NETX™

the high-performance real-time implementation
of TCP/IP standards

Simple Mail Transfer Protocol (SMTP)

User Guide

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

Part Number: 000-1054
Revision 5.0

Contents

Chapter 1

Introduction to NetX SMTP

The Simple Mail Transfer Protocol (SMTP) is a protocol designed for transferring mail across networks and the Internet. SMTP is a simple protocol that utilizes reliable Transmission Control Protocol (TCP) services to perform its content transfer function. Because of this, SMTP is a highly reliable content transfer protocol. Nearly all commercial email clients and servers utilize the SMTP protocol.

NetX SMTP Requirements

Client Requirements

The NetX SMTP Client API requires a creation of a NetX IP instance and NetX packet pool. Because the NetX SMTP Client utilizes NetX TCP services, TCP must be enabled with the *nx_tcp_enable* call prior to using the NetX SMTP API on that same IP instance. The SMTP Client uses a TCP socket to connect to an SMTP Server on the Server's SMTP port. This is typically set at the well-known port 25, though neither SMTP Client nor Server is required to use this port.

The size of the packet pool in terms of packet payload and number of packets available is user configurable, and depends on the anticipated volume of mail transmission and size of mail message content. The packet size associated with the SMTP Server or Client can be any size but it is recommended the application use the Ethernet device MTU (mean transfer unit) size as the packet size.

The Client creates a TCP socket for its session(s) for connecting to and transmitting mail to an SMTP Server. It is recommended that the Client use the well-known port 25 normally reserved for SMTP.

The NetX SMTP Client does not require creation of a byte and block pool for storing session and mail message data unless the host application desires this option. Access to the Client byte and block pools are mutex protected. Client memory is shared among all its sessions in a multi-session Client application. The size of these memory pools as well as the packet pool depends on the anticipated mail traffic for the Client and available resources. See **NetX SMTP Memory Allocation** elsewhere in this document for more information about using Client memory pools.

The SMTP Client requires a properly formatted mail message for mail transmission. It is not mail browser ("mail user") for creating the mail message. It is a "mail transfer agent" only.   It will provide any necessary processing of the mail message body for SMTP transport, such as data transparency (see Appendix C), as specified in RFC 2821. Mail message format, specified in RFC 2822 for including headers and message body is beyond the scope of the SMTP Client API.

Server Requirements

The NetX SMTP Server API also requires a NetX IP instance and NetX packet pool.   The packet pool is used by the Server to transmit messages back to the Client. The packet size associated with the SMTP Server or Client can be any size but it is recommended the application use the Ethernet device MTU (mean transfer unit) size as the packet size. The IP instance must be TCP enabled before using the NetX SMTP Server API. The Server creates a TCP socket for its session(s). The session socket(s) is assigned an unused port to listen for SMTP Client requests.   It is recommended that the Server use the well-known port 25 normally reserved for SMTP.

The NetX SMTP Server requires creation of a byte and block pool for storing session and mail message data, and mutexes to protect access to each one.   Server memory is shared among all its sessions in a multi-session Server application. The size of these memory pools as well as the packet pool depends on the anticipated mail traffic for the Server and available resources.

The SMTP Server typically requires a means to spool received SMTP Client mail to hard disk and release allocated memory back to the Server memory pools. The NetX SMTP package can take advantage of the ExpressLogic FileX file system package to meet this requirement.

NetX SMTP Server and NetX SMTP Client applications can work with non NetX SMTP applications. A NetX SMTP Client and Server can run as concurrent tasks on the same system. This is a typical configuration of most commercial mail servers, where a received mail queue is set up for the two tasks to coordinate receiving and transmitting mail.

NetX SMTP Constraints

While the NetX SMTP protocol implements the RFC 2821, 2822 and 2554 standards, there are some constraints:

1. The SMTP Client does not support mail data over 64k. It does support ASCII text mail, file attachments, and multi part MIME message data.

2. The SMTP Server and Client support only Login authentication, but not CRAM-MD5, and DIGEST-MD5 (with DES support).

3. Domain Name Server (DNS) is supported by NetX but not directly by SMTP API services.

4. The NetX SMTP API does not provide mail spooling or mail user agent services. A default mail spooler function is provided as a stub, but it is recommended the application define its own mail spooling function.

5. NetX SMTP Client expects mail message to be fully formatted with the necessary header fields and message body for display by a mail user agent. Likewise the NetX SMTP Server does not provide mail browser capability for parsing mail header data and viewing mail content.

6. NetX SMTP Client and Server API support a limited set of protocol extensions (AUTH) s well as legacy services (HELO). VRFY, SEND, SOML, EXPN, SAML, and SIZE requests are not supported.

7. SMTP Client services are for sending email only—there are no email download utilities (ETRN, TURN) provided in this package. See the NetX POP3 package for receiving mail.

8. The SMTP Server API does not support notifying the Client of failed mail message deliver, for example if the address cannot be found or the recipient host is not in service.

NetX SMTP IP Addresses (Mailboxes)

A NetX SMTP Client wishing to send mail must provide a reverse mailbox path to the sending localhost, and a mailbox for each recipient. All mailboxes must be fully qualified domain names.

A fully qualified domain name contains a local-part and a domain name, separated by an ‘@’ character.

For authentication, usernames can either be fully qualified domain names, or display user names. This depends on how the Server performs authentication. In the NetX SMTP Server package, authentication check is provided as a stub, but should be replaced with the custom application authentication check to suit your application requirements.

NetX SMTP Client Commands

The NetX SMTP Client uses the following commands to set up a mail session with an SMTP Server (not necessarily the NetX SMTP Server).

See Appendix C for details on Client command syntax.

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.
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.
RSET	The Client would like to restart the current mail transaction.
NOOP	The Client would like an OK-acknowledgement from the Server.

NetX SMTP Server Reply Codes

Reply codes are categorized by the first digit. This digit indicates the general acceptance or rejection of the Client command.

See Appendix A for a complete listing of Server reply codes. See Appendix C for details on Server reply syntax.

Reply code categories

2xx *Positive completion reply*. The Client command was acknowledged, had no errors, was executed by the Server, and the Client SMTP state may progess to the next step in the mail transaction process.

220 successful connection with Client

221 acknowledge Client quit command

235 successful authentication of Client data

250 command successfully processed

3xx *Positive intermediate reply*. The Client command was acknowledged, had no errors, but execution is pending further information or action on the part of the Client.

334 Server is prompting the Client for further authentication data.

354 Server wants Client to start transmitting message data

4xx *Temporary negative reply*. The Client command was accepted but the Server was unable to carry it out. The error condition is temporary, not the fault of the Client, and the Client should retry the command at some later time.

421 Server has detected an internal fatal error or transmission failure which requires it to abort the current session

451 Server is experiencing insufficient storage, mailbox unavailability, or processing errors on its side

5xy *Permanent negative reply*. The Client command was not accepted and the requested action did not occur. The error is considered to be on the Client side. However, “permanent” conditions can be corrected depending on the Client error handling, such as retrying a command or resetting a session.

50x indicates syntax either with the command or its parameters

55x indicates a problem with the mail system. Examples would be mailbox address not permitted (not on the local network), mailbox not found on the network, or the Client has exceeded storage allocation (“policy” limitations).

NetX SMTP Model

The TCP/IP layer activities such as packet acknowledgement, detecting dropped packets, and IP packet chaining are not visible to the SMTP protocol.

The NetX SMTP Client creates one or more sessions.   Each session can run within the Client thread, or execute as separate threads.   After the Client session receives a fully formatted mail message to deliver from the host environment, the Client session then executes the SMTP protocol to send out Client mail. The Server meanwhile listens for Client connection requests on one of its available session socket(s).

Once the SMTP Client and the Server connect, each maintains a ‘state’ of the SMTP protocol. The next Client command depends on acceptance of its previous command by the Server. The next set of commands the Server will accept from the Client depends on the previous command it accepted from the Client. The mail data being transmitted and received respectively are preserved between messages exchanged between the Client and Server. If the Client wishes to reset the session, the mail data is cleared and both Client and Server restart the state machines at the ‘MAIL’ state.

The Client and Server exchange commands and reply codes one to one. If this one to one exchange is disrupted, the Client and Server may end up waiting to receive the other’s next transmission and appear to hang up until their respective session eventually times out.

While the SMTP protocol claims to be very simple, there is wide interpretation and variability among different SMTP Servers and Clients as well as disruption and variation in network traffic. Therefore, the SMTP protocol requires that Servers and Clients make a ‘best effort’ for the mail transaction to succeed.   To meet specific network and application requirements, the NetX SMTP Server API provides enough services for the application to write its own ‘session run’ service, using the SMTP Server API client command handlers to best facilitate Server mail transport. Likewise, the NetX SMTP Client API provides enough services for the application to write its own ‘session run’ service, using the SMTP Client API server reply handlers to best facilitate Client mail transport.

However the NetX SMTP API does provide a default ‘session run’ services for the SMTP Client and Server to execute the SMTP protocol. This service performs the complete SMTP protocol to initiate and complete SMTP mail transactions.   The demo file as shown in the **Small Example System** section demonstrates the use of the built in session run service offered by the SMTP Client API, *nx_smtp_session_run()* and by the SMTP Server API*, nx_smtp_server_session_run().*

The SMTP Protocol

The general sequence of SMTP protocol is as follows:

1. The Client greets the Server by way of connecting over a TCP socket.

2. The Server responds to the Client greeting with a 221 code and optionally additional text indicating the Server domain.

3. The Client sends the hello command, EHLO, to the Server and waits for Server acknowledgment and list of supported services.

    If the EHLO command is not accepted, the Client sends the older form of hello command, HELO. This would be the case if the SMTP Server is a legacy Server before extended services were created, or if it can only offer basic SMTP services.

4. The Server responds with the 250 code and usually a simple acknowledgment such as “OK”. Typically it will also include a list of extended services supported.

5. At this point, the Client can optionally send an AUTH command with an authentication mechanism as the only parameter.

6. The Server will either accept the AUTH command with the 334 code, or reject the command with a 5xy code if it does not support or recognize the authentication mechanism. The Client and Server then exchange challenges and replies to each challenge until the Server sends the 235 code to the Client indicating the Client is successfully authenticated.

   See section **SMTP Authentication** for more details.

7. If the authentication succeeds, or if the SMTP Server does not require authentication, the Client next sends a MAIL FROM command with the original mail author’s mailbox address.

   Example: MAIL FROM: <skielbooms@expresslogic.com>.

   The NetX SMTP Client and Server packages do not support adding authentication parameters onto the MAIL FROM command, although this can be accommodated by some SMTP servers.

8. The Server if it accepts the MAIL command will acknowledge with another 250 OK. If it rejects the mail command, it sends a 5xy command.

Note that the Client can often get around a rejected MAIL FROM command (or any other command) simply be resending the same command. The NetX SMTP Client package provides a default reply code handler which has a mechanism for doing this. See section **NetX SMTP Client Reply Code Handling** for more details.

9. The Client sends a RCPT TO command with the recipient address as the only parameter.

   Example: RCPT TO:christiansen@expresslogic.com

   The mail recipient list must contain at least one RCPT TO: recipient and any number of RCTP CC and RCPT BCC commands.

10. The Server acknowledges with a 250 OK acknowledgment if there are no errors in the RCPT command or RCPT parameters. If it rejects the mail command, it sends a 5xy reply code and optionally some text with explanation.

11. The Client sends a DATA command with no parameters.

12. The Server responds with a 354 reply code if it accepts the command and awaits Client data. If it rejects the mail command, it sends a 5xy reply code.

13. The Client must ensure the mail message data has the proper message format as specified by SMTP protocol. See Appendix C Mail Message Formatting for specific details.

14. The Client transmits mail data in packets using the NetX API and the maximum packet size specified during packet pool creation. Packet size and packet pool creation is discussed in the section **Client Configuration Options**.

15. The Server receives one or more packets of the message data until the end of message sequence (EOM) is detected or a syntax error in the Client message is found such as total message length too long. If no errors are found, the Server sends the 250 reply code. Otherwise it sends a 5xy reply code.

16. The Client then sends the QUIT command and assumes the Server will successfully deliver the mail.

17. The Server acknowledges the QUIT command with the 221 reply code and closes the connection.

18. If the Server at any time has to terminate the session, it should wait for the Client QUIT command before actually closing the connection, if possible.

The syntax for SMTP commands and command parameters is very specific. See Appendix C for specific command syntax.

An SMTP Client must not advance to the next command without receiving an OK acknowledgement from the SMTP Server (see SMTP Server responses).

The Client can send the QUIT, RSET and NOOP commands at any time during a session.

If a command is rejected by the Server, the NetX SMTP Client reply handlers can resend the command one or more times until the command succeeds or the limit on resending a command is reached. The NetX SMTP Client has a configurable limit on the number of times the RSET command can sent. See Chapter 3 for the NX_SMTP_CLIENT_SESSION_COMMAND_RETRIES configuration option. Often the second command is accepted by the Server.

Sample SMTP Client - Server Exchanges

Basic SMTP example:

```
S: 220 www.example.com
C: HELO mydomain.com
S: 250 Hello mydomain.com
C: MAIL FROM:<sender@mydomain.com>
S: 250 Ok
C: RCPT TO:<friend@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C:
Subject: test message\r\n
From: sender@mydomain.com\r\n
To: friend@example.com\r\n
\r\n
Hello\r\n
This is a test\r\n
Goodbye.\r\n
.\r\n

S: 250 Ok: queued as 12345
```

```
C: QUIT
S: 221 Bye
```

Extended SMTP protocol example

In this example, the Server supplies the list of extended services it supports. The dashes after the reply code indicate more lines to follow in the current reply.

```
S: 220 www.example.com
C: EHLO mydomain.com
S: 250-serverdomain.com Hello mydomain.com [127.0.0.1]
S: 250-SIZE 14680064
S: 250-PIPELINING
S: 250 HELP
C: MAIL FROM:<sender@mydomain.com>
S: 250 Ok
C: RCPT TO:<friend@example.com>
Etc…
```

SMTP protocol Authentication example

In this example the Server only sends two challenges to the Client, “Username:” and “Password:” preceded by the 334 code. However, it can send as many as it feels is required to authenticate the Client.

```
S: 220 www.example.com
C: EHLO mydomain.com
S: 250-serverdomain.com Hello mydomain.com [1.2.3.4]
S: 250-AUTH LOGIN CRAM-MD5
S: 250-VRFY
S: 250 HELP
C: AUTH LOGIN
S: 334 VXNlcm5hbWU6
C: d2VsZG9u
S: 334 UGFzc3dvcmQ6
C: dzNsZDBu
S: 235 OK Authenticated
Etc…
```

Here are the base64 strings above decoded after `C: AUTH LOGIN:`

```
S: 334 Username:
C: weldon
S: 334 Password:
C: w3ld0n
```

NetX SMTP Client Reply Handlers

The NetX SMTP Client API reply handlers determine the next state of the SMTP protocol which in turn determines the next command (or mail message data) to send to the Server. This decision is based on acceptance or rejection of the previous command sent to the server and the state of the SMTP protocol and current mail item being processed. If, for example, the current mail item at least one valid recipient accepted by the Server but its last RCPT command was rejected, it will go ahead with the DATA command, since there is at least one valid recipient to receive it. If the Server rejects its message data as invalid, for example, if the message size is too big, the mail is marked for deletion and it will send the QUIT command next and exit the session.

The default handlers do not attempt to respond to each reply code specifically. To limit the reply handling to a manageable size, codes are grouped as much as makes sense by their first digit category and handled in the same manner.

In the example session below, the Client sends a DATA command before the RCPT command. The Server responds by sending an out-of-sequence error code.

```
C: MAIL FROM:<sender@mydomain.com>
S: 250 Ok
C: DATA
S: 503 Command out of sequence
C: RSET
S: 250 OK
Etc…
```

The NetX SMTP Client reply handler checks for code 503 in the Server reply. If it receives one, it will send the RSET command to restart the mail transaction. However, for all other 5xy codes the reply handler will simply retry the command up to a certain number of times the session is configured for before aborting the session.

Note that not all SMTP server applications apply the server reply codes similarly.

NetX SMTP Memory Allocation

The NetX SMTP Server API requires the host application to set up a byte and block memory pool for the Server. Byte and block pool size, including block size, are user configurable. The Server session run service, which handles communicating with the Client and receiving Client mail, automatically allocates and releases memory during an SMTP Client session and spools it to hard disk. If the SMTP Server spooling service, which is user defined, is unable to spool a mail item to hard disk, the Server will not mark that mail for deletion and log an error message. Potentially the Server can run out of memory, especially block pool memory, as mail not being deleted causes memory pool depletion. Hence the host application e.g. the spool service must be aware of this potential problem and handle it accordingly.

The SMTP Client API does not require the host application to use the Client byte and block pool memory for creating mail data. Optionally the host application can use stack memory to create mail data. This is well suited to an SMTP client application expecting a low volume of mail to transmit. To choose this option, the host application must define **NX_SMTP_CLIENT_NO_MEM_ALLOC**. This will enable the SMTP Client API services to know when mail creation and deletion requires the Client bye and block pool memory.

However, to use SMTP Client dynamic memory allocation, the host application must set up a byte and one block pool and leave NX_SMTP_CLIENT_NO_MEM_ALLOC undefined. This might be well suited to a multi threaded, multi session SMTP Client application handling a large volume or large variations in mail volume. The SMTP Client API will assume that the Client is configured with byte and block pool memory and invoke its memory services to allocate and release memory accordingly.

Below is a table of SMTP Client API services to be used depending on the host application's choice of memory allocation:

No Client memory allocation	Client memory allocation
nx_smtp_mail_create	nx_smtp_mail_create_memalloc
nx_smtp_recipient_create	nx_smtp_recipient_create_memalloc
nx_smtp_cmd_message	nx_smtp_cmd_message_memalloc
N/A	nx_smtp_mail_message_append

The **Small Example System** located elsewhere in this document does not use the Client memory allocation option. For an example of an application which does, see the demo file in the examples directory.

Note that Client and Server dynamic memory is not the same memory in the *tx_application_define()* where the system uses the system defined first unused memory (see the *Small Example System* section). That memory is defined in the linker directives file and is platform dependent.

Dynamic Client and Server memory services memory scheme (one byte and one block pool) is simple and practical, although it limits flexibility for customizing memory to suit the application's requirements. When invoked, the SMTP Client and Server API session run services use the byte pool for small memory allocation (e.g. authentication challenges, mailbox addresses, usernames). The SMTP Client and Server API use the block pool for storing mail message data, which typically requires larger chunks of memory. A small one sentence email with subject header is at least 300 bytes.

The NetX SMTP Client and Server API memory offer services for the host application to allocate and release byte and block pool memory directly. These services handle the task of acquiring and releasing the mutex for the memory pool being accessed.

SMTP Client and Server applications should free allocated memory as soon as it is not needed, especially block pool memory to avoid subsequent mail transactions from not having sufficient storage space.

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 its 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. A Client need only authenticate itself once during a mail session.

There are two flavors of authentication, namely *basic* and *digest*. *Digest* is not includes in the current release of NetX SMTP Client, so 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 encoded in the base64 format. 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. This makes it easier for the name and password to be stolen.

NetX SMTP Authentication Challenge

The NetX SMTP authentication process between Server and Client is very similar to most commercial mail applications. The specific details of the NetX SMTP authentication process are described below.

The SMTP Client parses Server reply text for authentication services provided by the Server. The Client must find a matching authentication type supported by the Server in its own authentication list which is set in the *nx_smtp_client_create()* call. See **Client Utility Services** elsewhere in this document for the SMTP Client API services providing this service.

The NetX SMTP Client supports the “LOGIN” mechanism, so it will send out the AUTH command with this parameter. If the NetX SMTP Server supports this type of authentication, it will reply with a 334 reply code and start the authentication ‘challenge’. It sends an base64 encoded prompt back to the Client which is typically “Username:”. The Client decodes the prompt, and replies with its base64 encoded username. If the Server accepts the Client username, it sends out another base64 encoded prompt for the Client password. The Client responds with its base64 encoded password. At this point, the Server will determine if the Client authentication is successful using the authentication check service it was configured with in the *nx_smtp_create_server()* call. If so, it responds with the 235 success code.

Example authentication exchange:

```
C:   AUTH LOGIN
S:   334 VXNlcm5hbWU6
C:   dGVzdEB3aXouZXhhbXBsZS5jb20=
S:   334 UGFzc3dvcmQ6
C:   dEVzdDQy
S:   235 OK Authenticated
```

The RFC imposes no limits on the number of challenges a Server can send to the Client before it considers the Client authenticated. At some point the Server will accept the Client as authenticated and send the 235 reply code. If it rejects the Client response to its challenge, it returns a 535 code. If the Client needs to cancel authentication it sends a ‘*’ to the Server. The Server should respond with a 5xy code (RFC 2554 specifies ‘501’).

NetX SMTP Event Logging

The NetX SMTP Client API includes a configurable logging service **NX_SMTP_CLIENT_EVENT_LOG** to log events during an SMTP Client operation. The NetX SMTP Server API includes a similar configurable logging service **NX_SMTP_SERVER_EVENT_LOG** to log events during an SMTP Server operation. These macros are defined in nx_smtp.h. The Client and Server event logging level are application defined parameters. See sections **Client Configuration Options** and **Server Configuration Options** for the configurable Client and Server options elsewhere in this document for more information.

There are four levels of event logging, ranging from NONE to ALL.

- NONE: no messages are logged

- SEVERE: only messages of SEVERE level severity. The Client session is not able to recover from these events and the Client session and possibly the Client must abort operation.

- MODERATE: messages of moderate or fatal level are logged. Moderate significance includes events such as authentication failing, invalid mail message data received, a lost TCP connection. The Client session is able to recover from these events, even if the mail transaction fails.

- ALL: all messages are logged. These events include successful mail transaction, successful authentication, and successful session initialization. This level of event logging is intended for testing and debugging the SMPT application. Normal mail traffic would produce too much data to be practically useful.

Below is an example NX_SMTP_CLIENT_EVENT_LOG logging call:

```
/* Log the event. */
NX_SMTP_CLIENT_EVENT_LOG(MODERATE, ("Session %d error connecting
              client %s to server. Status: 0x%x\n\r",
              session_ptr -> session_id,
              session_ptr -> client_ptr -> client_name,
              session_run_status));
```

The message typically includes the ID of the session in which the event occurs, and the error status, and may include additional identifying information.

The NetX SMTP Client `NX_SMTP_EVNT_LOG` macro uses a *printf* statement for displaying output. However, the application code can define its own printf.

NetX SMTP Multi-Session Support

A NetX SMTP Client application can be configured with multiple sessions to have concurrent mail transactions and greater mail output. The application code is responsible for defining both the Client and Client session thread entry functions. Each session thread runs the SMTP protocol state machine and mail transaction session with an SMTP Server.

The number of Client sessions is an application defined parameter. The default NetX SMTP Client session count is one. There is no limit other than the ThreadX limit and the system resources on the number of session threads an SMTP Client application can create. The Client creates the session threads and initializes its sessions so that they are ready to conduct an SMTP mail transaction. After each transaction, the session must be reinitialized before conducting another mail transaction.

In the example in **Small Example System** located elsewhere in this document, the Client creates one session which performs a single mail transaction in an iterative loop. This demo can easily be modified to create multiple sessions running concurrent mail transactions.

A NetX SMTP Server application can also create multiple Server sessions. Each session is assigned to a Server session thread, but unlike the SMTP Client *nx_smtp_server_session_thread_entry()* is part of the NetX SMTP Server API. The Server thread initializes all its sessions, then sets up an available Session socket to handle the first Client request. With each Client request, the assigned Server session accepts the request while the Server sets up the another available Session socket to listen for the next Client request. Like the Client session, the Server sessions must be reinitialized before they can accept another Client request.

The installation **NetX SMTP Examples** folder contains more advanced sample SMTP Client and SMTP Server applications running multiple sessions.

RFCs Supported by NetX SMTP

NetX SMTP API is compliant with RFC2821, 2822, and 2554.

Chapter 2 Installation and Use of NetX SMTP

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

NetX SMTP Installation

NetX SMTP Client is shipped on a single CD-ROM compatible disk. The package includes three source files, two include files, and a PDF file that contains this document, as follows:

nx_smtp_client.c C Source file for NetX SMTP Client API
nx_smtp_client.h C Header file for NetX SMTP Client API
nx_smtp_server.c C Source file for NetX SMTP Server API
nx_smtp_server.h C Header file for NetX SMTP Server API
nx_smtp.h C Header file for services and definitions common to NetX SMTP Server and Client

nx_smtp.pdf PDF description of NetX SMTP API for Client and Server applications

To use NetX SMTP Client API, the entire distribution mentioned previously should be copied to the same directory where NetX is installed. For example, if NetX is installed in the directory "*\threadx\mcf5272\green*" then the *nx_smtp.h*, *nx_smtp_client.h,* and *nx_smtp_client.c* files should be copied into this directory.

Using NetX SMTP

Using the NetX SMTP Client API is easy. The application must add *nx_smtp_client.c* to its build project. The application code must include *nx_smtp.h* and *nx_smtp_client.h* after *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX.

Using the NetX SMTP Server API is similar. The application must add *nx_smtp_server.c* in the build project. The application code must include *nx_smtp.h* and *nx_smtp_server.h* after *tx_api.h* and *nx_api.h*, in order to use ThreadX and NetX.

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 use the NetX SMTP Client and Server API.

Small Example Client Server System

An example of how easy it is to use NetX SMTP is described in Figure 1 that appears below. In this example, the SMTP Client and Server are created and their components set up in "*tx_application_define*". The SMTP Server is created first and its thread is allowed to start on line 241 ahead of the SMTP Client which is created next. After successful creation, the SMTP Client thread is then allowed to resume on line 245. In *demo_client_thread_entry*, the SMTP Client creates a mail item on line 341, and then connects to the Server and enters into an SMTP session by calling *nx_smtp_session_run* on line 401. The mail transaction is completed and the Client session reinitializes its socket for transmission, and deletes the mail just transmitted. After sending several mail items, the Client terminates its operations normally on line 448.

```
1    /*
2       demo_netx_smtp_client_server.c
3
4       This is a small demo of the NetX SMTP Client and Server on the
5       high-performance NetX TCP/IP stack.  This demo relies on Thread,
6       NetX and SMTP Client and SMTP Server APIs to perform simple SMTP mail
7       transfer between an SMTP client instance and the SMTP server instance.
8
9       Note that to adapt this demo to an embedded environment,
10      only one IP instance must be created and shared between Client and Server tasks,
11      and TCP, ARP, and ICMP enable service calls need only be made once.
12
13    */
14
15
16   #include  <stdio.h>
17   #include  "nx_api.h"
18   #include  "nx_ip.h"
19   #include  "nx_smtp.h"
20   #include  "nx_smtp_client.h"
21   #include  "nx_smtp_server.h"
22
23
24   /* Configure the Client to allocate all memory off the stack in this demo. */
25   #define NX_SMTP_CLIENT_NO_MEM_ALLOC
26
27   #define LOCALHOST                   "carolc2263@expresslogic.com"
28   #define LOCALHOST_PASSWORD          "cch42"
29   #define USERNAME_1                  "Janet Christiansen"
30   #define PASSWORD_1                  "jch42"
31   #define MAILBOX_1                   "jchristiansen@expresslogic.com"
32   #define CLIENT_DOMAIN                "expresslogic.com"
33
34
35   #define CLIENT_IPADR                IP_ADDRESS(1,2,3,4)
36   #define SERVER_IPADR                IP_ADDRESS(1,2,3,5)
37
38
39   /* Client control blocks and ThreadX and NetX resources used by client. */
40   NX_SMTP_CLIENT          demo_client;
41   NX_SMTP_CLIENT_SESSION  client_session;
42   CHAR                    mail_buffer[2500];
43   TX_THREAD               demo_client_thread;
44   NX_PACKET_POOL          client_packet_pool;
45   NX_IP                   client_ip;
46
47
48   /* Server control block and ThreadX and Netx resources used by server. */
49   NX_SMTP_SERVER          demo_server;
```

```
TX_THREAD               demo_server_thread;
TX_BYTE_POOL            server_byte_pool;
TX_BLOCK_POOL           server_block_pool;
TX_MUTEX                server_byte_pool_mutex;
TX_MUTEX                server_block_pool_mutex;
NX_PACKET_POOL          server_packet_pool;
NX_IP                   server_ip;


/* Server authentication check callback. */
UINT  authentication_check(CHAR *username_ptr, CHAR *password_ptr, UINT *authenticated);

/* Server mail spooler callback. */
UINT  server_mail_spooler(NX_SMTP_SERVER_MAIL *mail_ptr);

/* RAM 'network driver' enables SMTP server and client to communicate via RAM. */
void  _nx_ram_network_driver(struct NX_IP_DRIVER_STRUCT *driver_req);

/* Client thread entry point. */
void  demo_client_thread_entry(ULONG info);


/* Define main entry point.  */
int main()
{
    /* Enter the ThreadX kernel.  */
    tx_kernel_enter();
}


/* Define what the initial system looks like.  */

void    tx_application_define(void *first_unused_memory)
{

UINT      status;
CHAR    *free_memory_pointer;


    /* Setup the pointer to unallocated memory.  */
    free_memory_pointer =  (CHAR *) first_unused_memory;

    /* Initialize the NetX system. */
    nx_system_initialize();


  /* Configure the SMTP Client instance first. */

    /* Create Client packet pool (kind of a small one... */
    status =  nx_packet_pool_create(&client_packet_pool, "SMTP Clients Packet Pool",
                                    NX_SMTP_CLIENT_PACKET_SIZE, free_memory_pointer,
                                    NX_SMTP_CLIENT_PACKET_SIZE * 4);

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer = free_memory_pointer + NX_SMTP_CLIENT_PACKET_SIZE * 4;


    /* Create the Client thread */
    status = tx_thread_create(&demo_client_thread, "client_thread",
                              demo_client_thread_entry, 0, free_memory_pointer,
                              2048, NX_SMTP_CLIENT_PRIORITY,
                              NX_SMTP_CLIENT_PREEMPTION_THRESHOLD,
                              TX_NO_TIME_SLICE, TX_DONT_START);


    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer =  free_memory_pointer + 2048;

    /* Configure the session to be available (for an SMTP session). */
    client_session.available = NX_TRUE;

    /* Create an IP instance (to be shared by Client and Server). */
    status = nx_ip_create(&client_ip, "SMTP Client IP Instance", CLIENT_IPADR, 0xFFFFFF00UL,
                          &client_packet_pool, _nx_ram_network_driver, free_memory_pointer,
                          NX_SMTP_CLIENT_IP_STACK_SIZE, NX_SMTP_CLIENT_IP_THREAD_PRIORITY);


    free_memory_pointer =  free_memory_pointer + NX_SMTP_CLIENT_IP_STACK_SIZE;

    /* Enable ARP and supply ARP cache memory. */
    status =  nx_arp_enable(&client_ip, (void **) free_memory_pointer,
                                NX_SMTP_CLIENT_ARP_CACHE_SIZE);

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer = free_memory_pointer + NX_SMTP_CLIENT_ARP_CACHE_SIZE;

    /* Enable TCP for client. */
    status =  nx_tcp_enable(&client_ip);

    /* Enable ICMP processing. */
    status =  nx_icmp_enable(&client_ip);


    /* Now configure the SMTP Server instance. */

    /* Create the Server packet pool (kind of a small one). */
    status =  nx_packet_pool_create(&server_packet_pool, "SMTP Server Packet Pool",
                                    NX_SMTP_SERVER_PACKET_SIZE, free_memory_pointer,
                                    NX_SMTP_SERVER_PACKET_SIZE * 4);

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer = free_memory_pointer + NX_SMTP_SERVER_PACKET_SIZE * 4;

    /* Create an IP instance (to be shared by Client and Server). */
    status = nx_ip_create(&server_ip, "SMTP Server IP Instance", SERVER_IPADR, 0xFFFFFF00UL,
                          &server_packet_pool, _nx_ram_network_driver, free_memory_pointer,
                          NX_SMTP_SERVER_IP_STACK_SIZE, NX_SMTP_SERVER_IP_THREAD_PRIORITY);


    free_memory_pointer =  free_memory_pointer + NX_SMTP_SERVER_IP_STACK_SIZE;

    /* Enable ARP and supply ARP cache memory. */
    status =  nx_arp_enable(&server_ip, (void **) free_memory_pointer,
                            NX_SMTP_SERVER_ARP_CACHE_SIZE);

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer = free_memory_pointer + NX_SMTP_SERVER_ARP_CACHE_SIZE;

    /* Enable TCP for client. */
    status =  nx_tcp_enable(&server_ip);

    /* Enable ICMP processing. */
    status =  nx_icmp_enable(&server_ip);


    /* Set up the Server memory resources. */

    /* Create the Server byte pool.  */
    status = tx_byte_pool_create(&server_byte_pool, NX_SMTP_SERVER_BYTE_POOL_NAME,
                                 (VOID **)free_memory_pointer, 1024);

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer = free_memory_pointer + 1024;

    /* Create the Server block pool.  */
    status = tx_block_pool_create(&server_block_pool, NX_SMTP_SERVER_BLOCK_POOL_NAME,
                                  NX_SMTP_SERVER_BLOCK_SIZE, (VOID **)free_memory_pointer,
                                  NX_SMTP_SERVER_BLOCK_SIZE * 4);

    /* Update pointer to unallocated (free) memory. */
    free_memory_pointer = free_memory_pointer + NX_SMTP_SERVER_BLOCK_SIZE * 4;

    /* Create the Server byte pool mutex. */
    status = tx_mutex_create(&server_byte_pool_mutex,
                             NX_SMTP_SERVER_BYTE_POOL_MUTEX_NAME, TX_NO_INHERIT);

    /* Create the Server block pool mutex. */
    status = tx_mutex_create(&server_block_pool_mutex, NX_SMTP_SERVER_BLOCK_POOL_MUTEX_NAME,
                             TX_NO_INHERIT);


    /* Create the actual Server instance. */
    status = nx_smtp_server_create(&demo_server, NX_SMTP_SERVER_AUTHENTICATION_LIST, &server_ip,
                                    free_memory_pointer, 2048,
                                    NX_SMTP_SERVER_THREAD_PRIORITY,
                                    NX_SMTP_SERVER_PREEMPTION_THRESHOLD,
                                    NX_SMTP_SERVER_THREAD_TIME_SLICE, TX_DONT_START,
                                    &server_packet_pool, authentication_check,
                                    server_mail_spooler,
                                    &server_block_pool, &server_block_pool_mutex,
                                    NX_SMTP_SERVER_BLOCK_POOL_MUTEX_WAIT, &server_byte_pool,
                                    &server_byte_pool_mutex,
                                    NX_SMTP_SERVER_BYTE_POOL_MUTEX_WAIT);

     /* Check for error. */
     if (status != TX_SUCCESS)
     {
         NX_SMTP_SERVER_EVENT_LOG(SEVERE, ("Error creating Server. Status: 0x%x.\n\r", status));
         return;
     }

     free_memory_pointer += 2048;

     /* Create a Server session. This will be a single threaded SMTP Server. */
     status = nx_smtp_server_session_create(&demo_server,
                                            &(demo_server.nx_smtp_server_session_list[0]),
                                           1, free_memory_pointer,
                                           2048,
                                           NX_SMTP_SERVER_SESSION_THREAD_PRIORITY,
                                           NX_SMTP_SERVER_SESSION_THREAD_PRIORITY,
                                           NX_SMTP_SERVER_SESSION_THREAD_TIME_SLICE,
                                            TX_DONT_START);

     /* Check for error. */
     if (status != NX_SUCCESS)
     {
         /* Log the event. */
         NX_SMTP_SERVER_EVENT_LOG(SEVERE, ("Error creating server session. Status 0x%x\r\n",
                                  status));

         return;
     }

   /* Start the Server thread first so it can be waiting for Client mail. */

     /* Resume the SMTP server thread.  */
     printf("Starting the SMTP server thread...\n\r");

     nx_smtp_server_start(&demo_server);


     /* Now start the Client thread. */
     tx_thread_resume(&demo_client_thread);

    return;
}


/* Define the client thread task.

   This is a simple client thread which handles only one session at a time,
   sends off a single mail item with one recipient per session. All memory required
   for this operation is drawn from the stack.  After an arbitrary number of runs,
   the client task terminates and releases its ThreadX and NetX resources. */

void    demo_client_thread_entry(ULONG info)
{

UINT                        status;
UINT                        thread_counter = 1;
UINT                        continue_session = NX_TRUE;
NX_SMTP_CLIENT_SESSION      *client_session_ptr;
NX_SMTP_CLIENT_MAIL         session_mail;
NX_SMTP_CLIENT_RECIPIENT    mail_recipient;


    /* Create the actual SMTP Client instance. */
    status =  nx_smtp_client_create(&demo_client, LOCALHOST, LOCALHOST_PASSWORD,
                                    CLIENT_DOMAIN, NX_SMTP_CLIENT_AUTHENTICATION_LIST,
                                    NULL /* Optional date and time stamp */, NULL /* optional
                                    unique mail identifier */,
                                    &client_ip, &client_packet_pool,
                                    NULL /* client byte pool */,
                                    NULL /* client byte pool mutex */,
                                    NX_SMTP_CLIENT_BYTE_POOL_MUTEX_WAIT,
                                    NULL /* client block pool */,
                                    NULL /* client block pool mutex */,
                                    NX_SMTP_CLIENT_BLOCK_POOL_MUTEX_WAIT,
                                    NX_SMTP_GREETING_TIMEOUT_TICKS,
                                    NX_SMTP_ENVELOPE_TIMEOUT_TICKS,
                                    NX_SMTP_MESSAGE_TIMEOUT_TICKS,
                                    NX_SMTP_CLIENT_WINDOW_SIZE);

    if (status != NX_SUCCESS)
    {
        printf("Error creating the client. Status: 0x%x.\n\r", status);
        return;
    }

    /* The demo client username and password is the authentication
       data used when the server attempts to authentication the client. */


    /* Initialize the Client's only session with server ip address, port and Client. */
    status = nx_smtp_session_initialize(&client_session, 1, &demo_client,
                                         SERVER_IPADR, NX_SMTP_CLIENT_SESSION_PORT);

    /* Check for errors. */
    if (status != NX_SUCCESS)
    {
        printf("Error initializing client session. Status: 0x%x.\n\r", status);
        return;
    }

    /* Create and transmit mail items. */
    while (continue_session)
    {

        client_session_ptr = &(client_session);

        /* This mail message was taken from a Tour de France 2006 web site. */
        sprintf(mail_buffer,
        "To: %s\r\n"
        "From: %s\r\n"
        "Subject: Time Trial Stage 18\r\n"
        "Date: %s\r\n"
        "Message-ID: <%s>\r\n"
        "MIME-Version: 1.0\r\n"
        "Content-Type: text/plain;\r\n"
        "  charset=\"utf-8\"\r\n"
        "Content-Transfer-Encoding: 8bit\r\n"
        "\r\n"
        "The stage is a twenty five-mile individual time trial from\r\n"
        "the base of Mount Ventoux to the plains of Montceau-les-Mines\r\n"
        "(which, amazingly, was first settled in the first ice age).\r\n"
        "The route is very technical, with uneven pavement and an undulating profile.\r\n"
        "The RomMailer Embedded Email Toolkits are Simple Mail Transport Protocol (SMTP) clients"
        "that allow embedded devices to send email to any SMTP server. This allows any embedded"
        "device to send asynchronous status reports using the Internet application email.\r\n"


        "For example, a network printer using RomMailer could send an email to a system \r\n"
        "administrator when consumables such as toner or paper needed to be replenished.\r\n"
        "Or to the printer's service organizations when maintenance needs to be scheduled. \r\n"
        "Another use of email push technology is to manage networked populations of machines."

        "Individual vending machines have been attached to Web servers, but a large"
        "scale population benefit from email technology. RomMailer enables a network of vending"
        "machines to be managed remotely so that reports on each machine's sales and inventory "
        "can be sent via email to a central site. This data can be used to determine \r\n"
        "the profitability of each machine, the best mix of products for each machine, and when
        "to restock a machine. In addition, status alerts for coin jams, internal temperature,
        "and stock outages can be sent as needed. \r\n"


         "RomMailer sends pure text messages, messages with attachments, HTML messages, or HTML
         "with embedded graphics messages (MHTML) and is compliant with all the relevant RFCs.
         "The mail readers that come with Netscape Communicator and MS Internet Explorer 4.0 are
         "fully capable of reading HTML mail as is the newest version of Eudora. \r\n"
         "Messages may be constructed with static information and/or dynamic device data.
         "If RomMailer is used with Allegro's industry leading RomPager embedded Web server \r\n"
         "toolkit, the insertion of dynamic device data into a message uses the same field-proven
         "memory saving technology. In fact, the same page may be sent both as a response to a
         "browser request and as a spontaneous status report sent via email. \r\n",
          MAILBOX_1, LOCALHOST, NULL /* Optional date stamp */, NULL /* Optional unique ID */);


        /* Create a mail instance with the above text message. */
        status =  nx_smtp_mail_create(client_session_ptr, &session_mail,
                                      MAILBOX_1, NX_SMTP_MAIL_PRIORITY_NORMAL,
                                      mail_buffer, strlen(mail_buffer));

        /* Check for errors. */
        if (status != NX_SUCCESS)
        {
            printf("Error creating a mail item in client session 1. "
                             "Status: 0x%x.\n\r", status);

             break;
        }


         /* Start a session list of mail items to send. */
        nx_smtp_mail_add(client_session_ptr, &session_mail);


        /* Create mail recipient and set as this mail's current recipient. */
        status =  nx_smtp_recipient_create(client_session_ptr, &session_mail, &mail_recipient,
                                          USERNAME_1, MAILBOX_1, NX_SMTP_RECIPIENT_TO);


        /* Check for error. */
        if (status != NX_SUCCESS)
        {
             printf("Error creating client recipient %s. Status: 0x%x.\n\r", USERNAME_1, status);
             break;
        }

        /* Add newly created recipient to current mail's list of recipients. */
        status =  nx_smtp_recipient_add(&session_mail, &mail_recipient);

        /* Check for error. */
        if (status != NX_SUCCESS)
        {
            printf("Error adding client recipient to session %d mail. Status: 0x%x.\n\r",
                   status, client_session_ptr -> session_id);
            break;
        }

        /* Reset this mail item's current recipient to the start of the list. */
        session_mail.current_recipient_ptr = session_mail.start_recipient_ptr;

        status =  nx_smtp_session_run(client_session_ptr);

        if (status == NX_SUCCESS)
            printf("Client mail transaction %d status: successful!\r\n", thread_counter);
        else
            printf("Client error with mail %d transaction.  Status: 0x%x!\r\n", thread_counter,
                   status);

        /* Keep count of client sessions. */
        thread_counter++;

        /* Arbitrarily limit the Client to 10 SMTP sessions. */
        if (thread_counter > 3)
        {
            /* Terminate the client thread after 3 sessions executed. */
            continue_session = NX_FALSE;
        }

        /* Delete current mail item, close connection and update session availability status. */
        status = nx_smtp_session_reinitialize(client_session_ptr, continue_session);

        /* Wait for request to send mail out. Let other threads run. */
        tx_thread_sleep(200);
    }

    /* Check for errors during the SMTP session. */
    if (status != NX_SUCCESS)
    {

        /* Session run aborted. */
        printf("Client session terminated abnormally after %d run(s). "
               "Status 0x%x\n\r", thread_counter, status);
    }
    else
    {

        /* Normal session completion. */
        printf("Client session terminated normally after %d run(s). \r\n", thread_counter);
    }


    /* Release network resources used in the client session. */
    status = nx_smtp_session_delete(client_session_ptr);

    /* Release threadx resources used by client. */
    status = nx_smtp_client_delete(&demo_client);

    return;
}

/* Normally a mail server queue tasks notifies spools mail to hard disk
   for clients on its subnet, or else it forwards mail to the next 'hop'
   or ultimate destination, in which it might spool mail to an outgoing
   mail queue for a Client task to eventually process.

   This default spooler simply displays the mail and returns successful
   completion. The SMTP server will then delete memory resources allocated
   for this mail transaction.*/

UINT server_mail_spooler(NX_SMTP_SERVER_MAIL *mail_ptr)
{


    /* Display mail item. */
    nx_smtp_utility_print_server_mail(mail_ptr);


    /* Return successful server session. */
    return NX_SUCCESS;
}



/* Define the application's authentication check.  This is called by
   the SMTP server when it receives an authentication request. It is not
   accessed directly by the client.

   The default authentication check returns non successful completion only
   if it receives null username/password pointers.  This check requires only
   username and password data.  The caller has to send decoded/decrypted
   username/password received from the client.  */

UINT  authentication_check(CHAR *username_ptr, CHAR *password_ptr, UINT *authenticated)
{

UINT i;
CHAR name[] =     LOCALHOST;
CHAR password[] = LOCALHOST_PASSWORD;

    *authenticated = NX_FALSE;

    /* Check for invalid parameters. */
    if (!username_ptr || !password_ptr || !strlen(username_ptr)  || !strlen(password_ptr))
    {
        return NX_PTR_ERROR;
    }

    i =  0;

    /* Compare username with database name byte per byte */
    while (username_ptr[i] == name[i] && username_ptr[i] != (CHAR) NX_NULL )
    {
        i++;
    }

    /* Did the comparison succeed? */
    if (username_ptr[i] == (CHAR) NX_NULL && i > 0)
    {
        /* Yes, now compare the passwords.  */
        i =  0;

        /* Compare password with database password byte per byte */
        while (password_ptr[i] == password[i] && password_ptr[i] != (CHAR) NX_NULL)
        {
            i++;
        }

        /* Did the comparison succeed? */
        if (password_ptr[i] == (CHAR) NX_NULL && i > 0)
        {
            /* Yes, the user is authenticated */
            *authenticated = NX_TRUE;
        }
    }

    /* Return successful completion status even if authentication failed. */
    return (NX_SUCCESS);
}
```

Figure 1. Example of SMTP 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_DISABLE_ERROR_CHECKING	Defined, this option removes the basic SMTP error checking. It is typically used after the application has been debugged. The default NetX SMTP Client status is enabled.
NX_SMTP_CLIENT_DEBUG	This option sets the level of SMTP Client event logging, from logging ALL messages, to only logging FATAL errors. To disable logging, set level to NONE. The default NetX SMTP Client level is set to MODERATE.
NX_SMTP_CLIENT_NO_MEM_ALLOC	Defined, this enables the SMTP Client to create and delete mail without using the Client byte and block pools for memory allocation and release.
NX_SMTP_CLIENT_SESSION_COUNT	This option sets the number of the Client sessions. The default NetX SMTP Client size is 1.
NX_SMTP_PRINT_CLIENT_MAIL_DATA	This option enables printing Client mail after a mail transaction session. The default NetX SMTP Client feature is disabled.
NX_SMTP_PRINT_CLIENT_RESERVES	

	This option enables printing available Client byte and block pool memory and remaining packets in the Client packet pool after a mail transaction session. The default NetX SMTP Client feature is disabled.
NX_SMTP_CLIENT_PRINT_TIMEOUT	This option sets the timeout to obtain the Client print mutex. The default NetX SMTP Client is 1 second.
NX_SMTP_CLIENT_STACK_SIZE	This option sets the size of the Client thread stack. The default NetX SMTP Client size is 4096.
NX_SMTP_CLIENT_SESSION_STACK_SIZE	This option sets the size of the Client session thread stack. The default NetX SMTP Client size is 4096.
NX_SMTP_CLIENT_PRIORITY	This option sets the set the Client thread priority. The default NetX SMTP Client value is 2.
NX_SMTP_CLIENT_THREAD_TIME_SLICE	This option sets the time slice of the scheduler allows for Client thread execution. The default NetX SMTP Client size is TX_NO_TIME_SLICE.
NX_SMTP_CLIENT_SESSION_THREAD_TIME_SLICE	This option sets the time slice of the scheduler allows for a Client session thread execution. The default NetX SMTP Client size is TX_NO_TIME_SLICE.
NX_SMTP_CLIENT_ PRIORITY	This option sets the Client thread priority. The NetX SMTP Client default value is 2.

NX_SMTP_CLIENT_SESSION PRIORITY	This option sets the Client session thread priority. The default NetX SMTP Client value is set to NX_SMTP_CLIENT_ PRIORITY.
NX_SMTP_CLIENT_PREEMPTION_THRESHOLD	This option sets the sets the level of priority at which the Client thread allows preemption. The default NetX SMTP Client value is 2.
NX_SMTP_CLIENT_SESSION_PREEMPTION_THRESHOLD	This option sets the sets the level of priority at which the Client session thread allows preemption. The default NetX SMTP Client value is 2.

Configure NetX SMTP Client Network Resources

NX_SMTP_CLIENT_DOMAIN	This option sets the SMTP sender domain name which is included with the Client greeting to the Server. The default NetX SMTP Client is ‘expresslogic.com’
NX_SMTP_CLIENT_IP_STACK_SIZE	This option sets the Client IP helper thread stack size. The default NetX SMTP Client size is 2048 bytes.
NX_SMTP_CLIENT_IP_PRIORITY	This option sets Client IP helper thread priority. The default NetX SMTP Client value is 2.
NX_SMTP_CLIENT_SESSION_PORT	This sets the port which the Client socket will connect to the SMTP Server on. The default NetX SMTP Client is 25.

NX_SMTP_CLIENT_PACKET_SIZE	This sets the size of the TCP packet which carries message data to the SMTP Server. This includes TCP, IP, and Ethernet (Frame) packet header data. The default NetX SMTP Client is 1500.
NX_SMTP_CLIENT_PACKET_HEADER_SIZE	This option sets aside the number of bytes of the packet size for header data. The default NetX SMTP Client is 60.
NX_SMTP_CLIENT_PACKET_POOL_SIZE	This option sets the size of the SMTP Client packet pool. The NetX SMTP Client default is (10 * NX_SMTP_CLIENT_PACKET_SIZE).
NX_SMTP_CLIENT_TCP_SOCKET_NAME	This option sets the TCP socket name. The NetX SMTP Client TCP socket name default is "SMTP Client socket."
NX_SMTP_CLIENT_ARP_CACHE_MEM_SIZE	This option sets the ARP cache memory size. Each ARP entry is 52 bytes, so the number of ARP entries is the memory size divided by 52. The default NetX SMTP Client ARP cache memory size is 1040 (20 entries).
NX_SMTP_CLIENT_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. The default NetX SMTP Client TCP window size is NX_SMTP_CLIENT_PACKET_SIZE.

.

NX_SMTP_PACKET_TIMEOUT	This option sets the timeout on NetX packet allocation. The

default NetX SMTP Client packet timeout is 2 seconds.

NX_SMTP_TCP_SOCKET_SEND_WAIT

This option sets the timeout on a NetX TCP socket send completion. The default NetX SMTP Client socket send timeout is 2 seconds.

NX_SMTP_CLIENT_CONNECTION_TIMEOUT

This option sets the Client TCP socket connection timeout. The default NetX SMTP Client connection timeout is 5 seconds.

NX_SMTP_CLIENT_DISCONNECTION_TIMEOUT

This option sets the Client TCP socket disconnect timeout. The default NetX SMTP Client connect timeout is 5 seconds.

Configure Client memory resources

NX_SMTP_CLIENT_BYTE_POOL_ NAME

This option sets the name of the Client byte pool. The NetX SMTP Client default is “SMTP Client bytepool.”

NX_SMTP_CLIENT_BYTE_POOL_SIZE

This option sets the NetX SMTP Client byte pool size. The NetX SMTP Client byte pool default size is 2048 bytes.

NX_SMTP_CLIENT_BYTE_POOL_MUTEX_NAME

This option sets the name of the Client byte pool mutex. The NetX SMTP Client byte pool mutex name default is “SMTP Client bytepool mutex.”

NX_SMTP_CLIENT_BYTE_POOL_MUTEX_WAIT

This option sets the Client byte pool mutex timeout to obtain the mutex. The NetX SMTP Client byte pool mutex timeout is 5 seconds.

NX_SMTP_CLIENT_BLOCK_POOL_ NAME

This option sets the name of the Client block pool. The NetX SMTP Client block pool name is default is "SMTP Client blockpool."

NX_SMTP_CLIENT_BLOCK_ SIZE

This option sets the NetX SMTP Client block pool's block size. The NetX SMTP Client block pool default size is NX_SMTP_CLIENT_PACKET_SIZE

NX_SMTP_CLIENT_BLOCK_POOL_SIZE

This option sets the NetX SMTP Client block pool size. The NetX SMTP Client block pool default size in bytes is

16 * X_SMTP_CLIENT_PACKET_SIZE.

Note this is not large enough to handle a 64k message, which is the RFC recommended maximum message size.

NX_SMTP_CLIENT_BLOCK_POOL_MUTEX_NAME

This option sets the name of the Client block pool mutex. The NetX SMTP Client block pool mutex name idefault is "SMTP Client blockpool mutex."

NX_SMTP_CLIENT_BLOCK_POOL_MUTEX_WAIT

This option sets the Client block pool mutex timeout to obtain the mutex. The NetX SMTP Client block pool mutex timeout default is 5 seconds.

Configure NetX SMTP Client Session

NX_SMTP_DATA_TRANSPARENCY_BYTES

This option sets aside the number of bytes allowed for data transparency processing where the mail message requires 'byte stuffing' to prevent accidental end of message detection. The default NetX SMTP Client reset limit is 5 bytes.

NX_SMTP_CLIENT_SESSION_RESETS

This option sets the maximum number of times a Client session may reset. The default NetX SMTP Client reset limit is 2 session resets.
.

NX_SMTP_CLIENT_SESSION_COMMAND_RETRIES

This option sets the maximum number of times a command may be retried consecutively in a Client session.. Typically if a command is not accepted by the Server by the second try, it will always fail. The default NetX SMTP Client limit is 2 retries

NX_SMTP_CLIENT_MESSAGE_ID_SIZE

This option sets the maximum size of the Client mail message ID. Typically this ID has the format timestamp @ domain. The default NetX SMTP Client value is

NX_SMTP_DATE_AND_TIME_STAMP_SIZE + 1 +
NX_SMTP_MAX_USERNAME

NX_SMTP_CLIENT_AUTHENTICATION_LIST

	This option defines the Client list of supported authentication types. Client authentication can be disabled setting this parameter to NULL in the *nx_smtp_client_create* service call. This default NetX SMTP Client list is “LOGIN”. Note: no other authentication types are currently supported by the SMTP Client.
NX_SMTP_REPLY_BUFFER_SIZE	This option sets the size of the SMTP Client session buffer for storing the text of the Server reply. The default NetX SMTP Client value is 512. This is the RFC mandated limit on the size of an SMTP Server reply.
NX_SMTP_GREETING_TIMEOUT_TICKS	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_TICKS	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_TICKS	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.

Server Configuration Options

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

Define	**Meaning**
Configure NetX SMTP Server Debug and Event Logging Parameters	
NX_DISABLE_ERROR_CHECKING	Defined, this option removes the basic SMTP error checking. It is typically used after the application has been debugged. The default NetX SMTP Server status is enabled.
NX_SMTP_SERVER_DEBUG	This option sets the level of SMTP Server event logging, from logging ALL messages, to only logging FATAL errors. To disable logging, set level to NONE. The default NetX SMTP Server level is set to MODERATE.
NX_SMTP_PRINT_SERVER_MAIL_DATA	This option enables printing Server mail after a mail transaction session. The default NetX SMTP Server feature is disabled.
NX_SMTP_PRINT_SERVER_RESERVES	This option enables printing available Server byte and block pool memory and remaining packets in the Server packet pool after a mail transaction session. The default NetX SMTP Server feature is disabled.
NX_SMTP_SERVER_PRINT_TIMEOUT	This option sets the timeout to obtain the Server print mutex. The default NetX SMTP Server is 1 second.
Configure NetX SMTP Server and Server Session Thread Parameters	

NX_SMTP_SERVER_THREAD_STACK_SIZE

This option sets the size of the Server thread on the stack. The default NetX SMTP Server value is 4096.

NX_SMTP_SERVER_THREAD PRIORITY

This option sets the set the Server thread priority. The default NetX SMTP Server value is 2.

NX_SMTP_SERVER_THREAD_TIME_SLICE

This option sets the time slice of the scheduler allows for Server thread execution. The default NetX SMTP Server value is TX_NO_TIME_SLICE.

NX_SMTP_SERVER_PREEMPTION_THRESHOLD

This option sets the sets the level of priority at which the Server thread allows preemption. The default NetX SMTP Server value is NX_SMTP_SERVER_THREAD PRIORITY.

NX_SMTP_SERVER_SESSION_THREAD_STACK_SIZE

This option sets the size of the Server session thread stack. The default NetX SMTP Server value is 4096.

NX_SMTP_SERVER_SESSION_THREAD_TIME_SLICE

This option sets the time slice of the scheduler allows for a Server session thread execution. The default NetX SMTP Server value is TX_NO_TIME_SLICE.

NX_SMTP_SERVER_ SESSION_THREAD_PRIORITY

This option sets the Server session thread priority. The

	default NetX SMTP Server value is NX_SMTP_SERVER_THREAD_PRIORITY.
NX_SMTP_SERVER_SESSION_PREEMPTION_THRESHOLD	This option sets the sets the level of priority at which the Server session thread allows preemption. The default NetX SMTP Server value is NX_SMTP_SERVER_ SESSION_THREAD_PRIORITY.

Configure NetX SMTP Server Network Resources

NX_SMTP_SERVER_DOMAIN	This option sets the SMTP Server domain name which is included with the Server greeting to the Client. The default NetX SMTP Server domain is ‘Server.com.’
NX_SMTP_SERVER_SESSION_PORT	This option sets the SMTP Server port on which to listen for Client requests. The default NetX SMTP Server value is 25.
NX_SMTP_SERVER_SOCKET_QUEUE_SIZE	This option sets the number of Client requests that can be queued in the SMTP Server socket. The default NetX SMTP Server value is 5.
NX_SMTP_SERVER_WINDOW_SIZE	This option sets the size of the Server TCP receive window. This should be set to below the MTU size of the underlying Ethernet hardware. The default NetX SMTP Server TCP window size is NX_SMTP_SERVER_PACKET_SIZE.

NX_SMTP_SERVER_PACKET_SIZE	This sets the size of the TCP packet which carries message data to the SMTP Client. This includes TCP, IP, and Ethernet (Frame) packet header data. The default NetX SMTP Server is 1500.
NX_SMTP_SERVER_PACKET_HEADER_SIZE	This option sets aside the number of bytes of the packet size for header data. The default NetX SMTP Server is 60.
NX_SMTP_SERVER_PACKET_POOL_SIZE	This option sets the size of the SMTP Server packet pool. The NetX SMTP Server default is (10 * NX_SMTP_SERVER_PACKET_SIZE).

.

NX_SMTP_SERVER_PACKET_TIMEOUT	This option sets the timeout on NetX packet allocation. The default NetX SMTP Serverpacket timeout is 1 second.
NX_SMTP_SERVER_IP_STACK_SIZE	This option sets the Server IP helper thread stack size. The default NetX SMTP Server size is 2048 bytes.
NX_SMTP_SERVER_IP_PRIORITY	This option sets Server IP helper thread priority. The default NetX SMTP Server value is 2.
NX_SMTP_SERVER_ARP_CACHE_MEM_SIZE	This option sets the ARP cache memory size. Each ARP entry is 52 bytes, so the number of ARP entries is the memory size

divided by 52. The default NetX SMTP Server ARP cache memory size is 1040 (20 entries).

NX_SMTP_SERVER_TCP_SOCKET_SEND_WAIT

This option sets the Server TCP socket send timeout. The default NetX SMTP Server connection timeout is 3 seconds.

NX_SMTP_SERVER_TCP_ RECEIVE_TIMEOUT

This option sets the Server TCP socket receive timeout. The default NetX SMTP Server connection timeout is 5 seconds.

NX_SMTP_SERVER_CONNECTION_TIMEOUT

This option sets the Server TCP socket connection timeout. The default NetX SMTP Server connection timeout is NX_WAIT_FOREVER (no timeout).

NX_SMTP_SERVER_DISCONNECTION_TIMEOUT

This option sets the Server TCP socket disconnect timeout. The default NetX SMTP Server connect timeout is 10 seconds.

Configure Server memory resources

NX_SMTP_SERVER_BYTE_POOL_ NAME

This option sets the name of the Server byte pool. The NetX SMTP Server default is "SMTP Server bytepool."

NX_SMTP_SERVER_BYTE_POOL_SIZE

This option sets the NetX SMTP Client byte pool size. The NetX SMTP Client byte pool default size is 4096 bytes.

NX_SMTP_SERVER_BYTE_POOL_MUTEX_NAME

This option sets the name of the Server byte pool mutex. The NetX SMTP Server byte pool mutex name default is “SMTP Server bytepool mutex.”

NX_SMTP_SERVER_BYTE_POOL_MUTEX_WAIT

This option sets the Server byte pool mutex timeout to obtain the mutex. The NetX SMTP Server byte pool mutex timeout is 5 seconds.

NX_SMTP_SERVER_BLOCK_POOL_ NAME

This option sets the name of the Server block pool. The NetX SMTP Server block pool name is default is “SMTP Server blockpool.”

NX_SMTP_SERVER_BLOCK_ SIZE

This option sets the NetX SMTP Server block pool’s block size. The NetX SMTP Server byte pool default size is NX_SMTP_SERVER_PACKET_SIZE.

NX_SMTP_SERVER_BLOCK_POOL_SIZE

This option sets the NetX SMTP Server block pool size. The NetX SMTP Server block pool default size in bytes is

10 * X_SMTP_SERVER_PACKET_SIZE.

Note this is not large enough to handle a 64k message which is the RFC recommended maximum message size.

NX_SMTP_SERVER_BLOCK_POOL_MUTEX_NAME

This option sets the name of the Server block pool mutex. The NetX SMTP Server block pool mutex name idefault is "SMTP Server blockpool mutex."

NX_SMTP_SERVER_BLOCK_POOL_MUTEX_WAIT

This option sets the Server block pool mutex timeout to obtain the mutex. The NetX SMTP Server block pool mutex timeout default is 5 seconds.

Configure NetX SMTP Server Session Parameters

NX_SMTP_MAX_SERVER_SESSIONS

This option sets the number of the Server sessions. The default NetX SMTP Server value is 1.

NX_SMTP_SERVER_AUTHENTICATION_REQUIRED

This option if enabled requires the SMTP Client to be authenticated before the Server will accept a MAIL command from the Client. The default NetX SMTP Server setting is disabled.

NX_SMTP_SERVER_SESSION_MAIL_LIMIT

This option sets a limit on the number of mails the Server will transact in a single Client session. To disable this feature, set to zero. The default NetX SMTP Server value is 10.

NX_SMTP_SERVER_RECIPIENT_MAIL_LIMIT

This option sets a limit on the number of recipients allowed in a single SMTP mail item being transacted. To disable this feature, set to zero. The default NetX SMTP Server value is 100.

NX_SMTP_SERVER_AUTHENTICATION_LIST

This option defines the Server list of supported authentication types. Server authentication can be disabled setting this parameter to NULL in the *nx_smtp_server_create* service call.   This default NetX SMTP Server list is “LOGIN”. Note: no other authentication types are currently supported by the SMTP Server.

Chapter 3
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_session_initialize
Initialize an SMTP Client session instance

nx_smtp_session_reinitialize
ReInitialize an SMTP Client session instance for another mail session.

nx_smtp_session_delete
Delete an SMTP Client session instance

nx_smtp_mail_create
Create an SMTP Mail instance

nx_smtp_mail_delete
Delete an SMTP Mail instance

nx_smtp_mail_add
Add an SMTP mail instance to Client session

nx_smtp_recipient_create
Create an SMTP Recipient instance

nx_smtp_recipient_delete
Delete an SMTP Recipient instance

nx_smtp_recipient_add
Add recipient to SMTP Mail instance

nx_smtp_mail_message_append
Add message data to SMTP mail instance

nx_smtp_mail_message_process
Process message data to be valid SMTP data

Services for the SMTP Protocol and Client State Machine

nx_smtp_session_run
Run the SMTP protocol state machine to transmit mail to an SMTP Server and maintain Client state

nx_smtp_cmd_greeting
Send greeting to SMTP Server

nx_smtp_rsp_greeting
Respond to SMTP Server reply to greeting

nx_smtp_cmd_ehlo
Send EHLO command to SMTP Server

nx_smtp_rsp_ehlo
Respond to SMTP Server reply to EHLO command

nx_smtp_cmd_helo
Send HELO command to SMTP Server

nx_smtp_rsp_helo
Respond to SMTP Server reply to HELO command

nx_smtp_cmd_auth
Send AUTH command to SMTP Server

nx_smtp_rsp_auth
Respond to SMTP Server reply to AUTH command

nx_smtp_cmd_auth_challenge
Send authentication data after SMTP Server challenge

nx_smtp_rsp_ auth_challenge
Respond to SMTP Server reply to authentication data

nx_smtp_cmd_mail
Send MAIL command to SMTP Server

nx_smtp_rsp_mail
Respond to SMTP Server reply to MAIL command

nx_smtp_cmd_rcpt
Send RCPT command to SMTP Server

nx_smtp_rsp_rcpt
Respond to SMTP Server reply to RCPT command

nx_smtp_cmd_data
Send DATA command to SMTP Server

nx_smtp_rsp_data
Respond to SMTP Server reply to DATA command

nx_smtp_cmd_message
Send mail message data to SMTP Server

nx_smtp_rsp_message
Respond to SMTP Server reply to receiving message

nx_smtp_cmd_quit
Send QUIT command to SMTP Server

nx_smtp_rsp_quit
Respond to SMTP Server reply to QUIT command

nx_smtp_cmd_rset
Send RSET command to SMTP Server

nx_smtp_rsp_rset
Respond to SMTP Server reply to RSET command

nx_smtp_cmd_noop
Send NOOP command to SMTP Server

nx_smtp_rsp_noop
Respond to SMTP Server reply to NOOP command

Services for Client Memory Management

nx_smtp_bytepool_memory_get
Allocate memory from Client byte pool

nx_smtp__bytepool_memory_release
Release memory back to Client byte pool

nx_smtp_blockpool_memory_get
Allocate memory from Client block pool

nx_smtp__blockpool_memory_release
Release memory back to Client block pool

Client Utility services

nx_smtp_utility_read_server_code
Receive reply from SMTP Server; extract Server reply code and save reply text to session buffer

nx_smtp_utility_ *send_to_server*
Send data or command to SMTP Server

nx_smtp_utility_parse_server_services
Extract extension SMTP services from the Server EHLO reply

nx_smtp_utility_set_next_auth_type
Set next authentication type in Client list as the session authentication type

nx_smtp_utility_authentication_challenge
Decode Server challenge; send Client response

nx_smtp_utility_data_transparency
Format SMTP mail message for SMTP transport

nx_smtp_utility_print_client_mail_status
Display the contents and mail transaction data of a specified Client mail item

nx_smtp_utility_ print_client_reserves
Display available Client byte pool and block pool memory and remaining packets in Client packet pool

nx_smtp_client_create

Create an SMTP Client Instance

Prototype

```
UINT    nx_smtp_client_create(NX_SMTP_CLIENT *client_ptr,
                  CHAR *client_name, CHAR *client_password,
                  CHAR *client_domain, CHAR *authentication_list,
                  UINT (*client_date_stamp)(CHAR *date_stamp),
                  UINT (*client_create_unique_message_id)
                              (NX_SMTP_CLIENT_MAIL *mail_ptr),
                  NX_IP *ip_ptr, NX_PACKET_POOL *packet_pool_ptr,
                  TX_BYTE_POOL *bytepool_ptr,
                  TX_MUTEX *bytepool_mutex_ptr,
                  UINT bytepool_mutex_timeout,
                  TX_BLOCK_POOL *blockpool_ptr,
                  TX_MUTEX *blockpool_mutex_ptr,
                  UINT blockpool_mutex_timeout,
                  ULONG greeting_timeout, ULONG envelope_timeout,
                  ULONG message_timeout, ULONG window_size)
```

Description

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

Input Parameters

client_ptr	Pointer to SMTP Client control block
client_name	SMTP Client instance Client name
client_password	SMTP Client instance Client password
client_domain	Domain of SMTP Client instance
authentication_list	Pointer to list of Client authentication types
client_date_stamp	Pointer to user defined data stamp function
client_create_unique_message_id	Pointer to user defined function for creating unique mail message id
ip_ptr	Pointer to IP instance
packet_pool_ptr	Pointer to Client packet pool
bytepool_ptr	Pointer to Client byte pool
bytepool_mutex_ptr	Pointer to Client byte pool mutex
bytepool_mutex_timeout	Time to wait to obtain Client byte pool mutex
blockpool_ptr	Pointer to Client block pool
blockpool_mutex_ptr	Pointer to Client block pool mutex
blockpool_mutex_timeout	Time to wait to obtain Client block pool mutex
greeting_timeout	Time to wait to connect to Server
envelope_timeout	Time to wait for Server reply to command
message_timeout	Time to wait for Server reply to mail message
window_size	Size of the Client’s TCP socket receive window

Return Values

NX_SUCCESS	(0x00)	SMTP Client successfully created.
NX_PTR_ERROR	(0x16)	Invalid input pointer parameter.

Allowed From

Application Code

Example

```
/* Create the SMTP Client instance. */

status =  nx_smtp_client_create( &demo_client,
                                  LOCALHOST,
                                  LOCALHOST_PASSWORD,
                                  STP_CLIENT_DOMAIN,
                                  NX_SMTP_CLIENT_AUTHENTICATION_LIST,
                                  cient_date_stamp,
                                  client_create_unique_message_id,
                                  client_ip,
                                  &client_packet_pool,
                                  &client_byte_pool, &client_bytepool_mutex,
                                  bytepool_timeout, &client_block_pool,
                                  &client_blockpool_mutex, blockpool_timeout,
                                  NX_SMTP_GREETING_TIMEOUT_TICKS,
                                  NX_SMTP_ENVELOPE_TIMEOUT_TICKS,
                                  NX_SMTP_MESSAGE_TIMEOUT_TICKS,
                                  NX_SMTP_CLIENT_WINDOW_SIZE);

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

See Also

nx_smtp_client_delete, nx_smtp_session_initialize,
nx_smtp_session_reinitialize, nx_smtp_session_delete,
nx_smtp_mail_create, nx_smtp_mail_delete,
nx_smtp_mail_add, nx_smtp_recipient_create, nx_smtp_recipient_delete,
nx_smtp_recipient_add, nx_smtp_mail_message_append,
nx_smtp_mail_message_process, nx_smtp_session_run

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	(0x16)	Invalid input pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Delete the SMTP Client instance “my_client.” */
status = nx_smtp_client_delete(&my_client);

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

See Also

nx_smtp_client_create, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_delete, nx_smtp_mail_add, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_recipient_add, nx_smtp_mail_message_append, nx_smtp_mail_message_process, nx_smtp_session_run

nx_smtp_session_initialize

Initialize an SMTP Client session Instance

Prototype

```
UINT    nx_smtp_session_initialize(
                  NX_SMTP_CLIENT_SESSION *session_ptr,
                  ULONG session_id,
                  NX_SMTP_CLIENT *client_ptr,
                  ULONG ip_addr, USHORT port)
```

Description

This service initializes an SMTP session instance to connect with an SMTP Server and conduct a mail transaction.

Input Parameters

session_ptr	Pointer to SMTP Client session to initialize
session_id	Unique session ID
client_ptr	Pointer to session's SMTP Client
ip_addr	SMTP Server IP address in Big Endian format
port	SMTP Server port to connect to

Return Values

NX_SUCCESS	(0x00)	Client session successfully initialized
NX_PTR_ERROR	(0x16)	Invalid input pointer parameter

Allowed From

Application code, Threads

Example

```
/* Initialize the SMTP session instance. */

status =  nx_smtp_session_initialize(&client_session, 0x01, &demo_client,
                                server_ip_address,  smtp_session_port);


/* If an SMTP session instance was successfully initialized, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_delete, nx_smtp_mail_add, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_recipient_add, nx_smtp_mail_message_append, nx_smtp_mail_message_process, nx_smtp_session_run

nx_smtp_session_reinitialize

Reinitialize an SMTP Client session Instance

Prototype

```
UINT    nx_smtp_session_reinitialize(
                   NX_SMTP_CLIENT_SESSION *session_ptr,
                   UINT session_availability)
```

Description

This service reinitializes an SMTP session after a mail transaction is completed to connect with an SMTP Server again and conduct another mail transaction.

Input Parameters

session_ptr	Pointer to SMTP session to reinitialize
session_availability	Session availability status

Return Values

NX_SUCCESS	(0x00)	Client session successfully reinitialized
NX_PTR_ERROR	(0x16)	Invalid input pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Initialize the SMTP session instance. */

   status =  nx_smtp_session_reinitialize(session_ptr, NX_TRUE);


/* If an SMTP session instance was successfully reinitialized, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_delete, nx_smtp_mail_add, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_recipient_add, nx_smtp_mail_message_append, nx_smtp_mail_message_process, nx_smtp_session_run

nx_smtp_session_delete

Delete an SMTP session Instance

Prototype

```
UINT    nx_smtp_session_delete(NX_SMTP_SESSION *session_ptr)
```

Description

This service deletes an SMTP session instance and all mail and recipients associated with the session. It releases all memory dynamically allocated for the session instance back to the Client's memory pools.

Input Parameters

session_ptr	Pointer to SMTP session to delete

Return Values

NX_SUCCESS	(0x00)	Client session successfully deleted
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Delete the SMTP session instance. */
 status =  nx_smtp_session_delete(session_ptr);

/* If an SMTP session instance was successfully deleted, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_mail_create, nx_smtp_mail_delete, nx_smtp_mail_add, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_recipient_add, nx_smtp_mail_message_append, nx_smtp_mail_message_process, nx_smtp_session_run

nx_smtp_mail_create

Create an SMTP mail Instance

Prototype

```
UINT    nx_smtp_mail_create(NX_SMTP_CLIENT_SESSION *session_ptr,
                            NX_SMTP_CLIENT_MAIL **mail_ptr,
                            CHAR *reverse_path_ptr,
                            UINT priority, CHAR *mail_buffer, UINT
                            buffer_length)
```

Description

This service creates an SMTP mail instance including session the mail is associated with, mail delivery priority and mailbox path of the original author. It does not allocate memory for the mail message text, assuming the application code has already done this.

Input Parameters

session_ptr	Pointer to SMTP session creating the mail
mail_ptr	Pointer to SMTP mail instance being created
reverse_path_ptr	Pointer to string containing sender's mailbox
priority	Priority level at which mail is delivered
mail_buffer	Pointer to mail message text
buffer_length	Size of mail message text

Return Values

NX_SUCCESS	(0x00)	Mail successfully created
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
 /* Create Client mail. */
  status =  nx_smtp_mail_create(session_ptr, &mail_ptr, reverse_path_mailbox_ptr,
                                priority, buffer, length);

/* If an SMTP mail instance was successfully created, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_delete, nx_smtp_mail_add, nx_smtp_recipient_create,

nx_smtp_recipient_delete, nx_smtp_recipient_add,
nx_smtp_mail_create_memalloc,
nx_smtp_session_run

nx_smtp_mail_create_memalloc

Create an SMTP mail Instance

Prototype

```
UINT    nx_smtp_mail_create_memalloc(NX_SMTP_CLIENT_SESSION *session_ptr,
                                     NX_SMTP_CLIENT_MAIL **mail_ptr,
                                     CHAR *reverse_path_ptr,
                                     UINT priority)
```

Description

This service creates an SMTP mail instance including session the mail is associated with, mail delivery priority and mailbox path of the original author. This service uses the Client memory byte and block pool to store mail data.

Input Parameters

session_ptr	Pointer to SMTP session creating the mail
mail_ptr	Pointer to SMTP mail instance being created
reverse_path_ptr	Pointer to string containing sender's mailbox
priority	Priority level at which mail is delivered

Return Values

NX_SUCCESS	(0x00)	Mail successfully created
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
 /* Create Client mail. */
  status =  nx_smtp_mail_create_memalloc(session_ptr, &mail_ptr,
                                   reverse_path_mailbox_ptr, priority);

/* If an SMTP mail instance was successfully created, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_bytepool_memory_get, nx_smtp_mail_delete, nx_smtp_mail_add, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_recipient_add, nx_smtp_mail_message_append, nx_smtp_mail_message_process, nx_smtp_session_run, nx_smtp_bytepool_memory_release

nx_smtp_mail_delete

Delete an SMTP mail Instance

Prototype

```
UINT    nx_smtp_mail_delete(NX_SMTP_SESSION *session_ptr,
                            NX_SMTP_MAIL *mail_ptr)
```

Description

This service deletes an SMTP mail instance and all recipient and message data associated with the mail. It releases all memory dynamically allocated for the mail instance back to the Client's byte and block pools.

Input Parameters

session_ptr	Pointer to SMTP Session deleting the mail
mail_ptr	Pointer to SMTP mail instance being deleted

Return Values

NX_SUCCESS	(0x00)	Mail successfully deleted
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
  /* Delete the current session mail instance. */
    status = nx_smtp_mail_delete(session_ptr, mail_ptr);

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

See also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_add, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_recipient_add, nx_smtp_mail_message_append, nx_smtp_mail_message_process, nx_smtp_session_run, nx_smtp_bytepool_memory_release, nx_smtp_blockpool_memory_release

nx_smtp_mail_add

Add an SMTP mail Instance to an SMTP session

Prototype

```
UINT    nx_smtp_mail_add(NX_SMTP_SESSION *session_ptr,
                         NX_SMTP_MAIL *mail_ptr)
```

Description

This service adds an SMTP mail instance to the session mail.

Input Parameters

session_ptr	Pointer to SMTP Session adding the mail item
mail_ptr	Pointer to SMTP mail instance being added

Return Values

NX_SUCCESS	(0x00)	Mail to successfully added to session
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_SMTP_SESSION_MAIL_ERROR	(0xB4)	Error in session mail linked list

Allowed From

Threads

Example

```
/* Add mail to Client session. */
status = nx_smtp_mail_add(session_ptr, mail_ptr);

/* If mail instance was successfully added, then status = NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_add, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_recipient_add, nx_smtp_mail_message_append, nx_smtp_mail_message_process

nx_smtp_recipient_create

Create an SMTP recipient Instance

Prototype

```
UINT    nx_smtp_recipient_create(NX_SMTP_CLIENT_SESSION *session_ptr,
                          NX_SMTP_CLIENT_MAIL *mail_ptr,
                          NX_SMTP_CLIENT_RECIPIENT *recipient_ptr,
                          CHAR *recipient_name, CHAR *recipient_mailbox,
                          UINT type)
```

Description

This service configures an SMTP recipient instance with the input parameters and additional initial values for belonging to an SMTP Mail instance.   It does not allocate memory for the recipient data, assuming the caller has done this already.

Input Parameters

session_ptr	Pointer to the session the mail belongs to
mail_ptr	Pointer to SMTP mail creating the recipient
recipient_ptr	Pointer to recipient being configured
recipient_name	Pointer to recipient's name
recipient_mailbox	Pointer to recipient mailbox
type	Type of delivery: TO, CC or BCC

Return Values

NX_SUCCESS	(0x00)	Recipient successfully created
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Create mail recipient. */
status =  nx_smtp_recipient_create(session_ptr, mail_ptr,
                    &recipient_ptr, recipient_mailbox,
                    recipient_name, NX_SMTP_RECIPIENT_TO);

/* If an SMTP recipient instance was successfully created, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete,

nx_smtp_mail_create, nx_smtp_mail_add, nx_smtp_mail_delete,
nx_smtp_recipient_delete, nx_smtp_recipient_add,
nx_smtp_mail_message_append, nx_smtp_create_recipient_memalloc

nx_smtp_recipient_create_memalloc

Create an SMTP recipient Instance

Prototype

```
UINT    nx_smtp_recipient_create_memalloc(NX_SMTP_CLIENT_SESSION
                          *session_ptr, NX_SMTP_CLIENT_MAIL *mail_ptr,
                          NX_SMTP_CLIENT_RECIPIENT **recipient_ptr,
                          CHAR *recipient_name, CHAR *recipient_mailbox,
                          UINT type)
```

Description

This service creates an SMTP recipient instance belonging to an SMTP Mail instance.   It uses Client byte pool memory to store the new recipient being created.

Input Parameters

session_ptr	Pointer to the session the mail belongs to
mail_ptr	Pointer to SMTP mail creating the recipient
recipient_ptr	Pointer to location allocated for new recipient
recipient_name	Pointer to recipient's name
recipient_mailbox	Pointer to recipient mailbox
type	Type of delivery: TO, CC or BCC

Return Values

NX_SUCCESS	(0x00)	Recipient successfully created
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Create mail recipient. */
status =  nx_smtp_recipient_create_alloc(session_ptr, mail_ptr,
                    &recipient_ptr, recipient_mailbox,
                    recipient_name, NX_SMTP_RECIPIENT_TO);

/* If an SMTP recipient instance was successfully created, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_add, nx_smtp_mail_delete,

nx_smtp_recipient_delete, nx_smtp_recipient_add,
nx_smtp_mail_message_append, nx_smtp_mail_message_process,
nx_smtp_session_run, nx_smtp_bytepool_memory_get,
nx_smtp_bytepool_memory_release, nx_smtp_create_recipient

nx_smtp_recipient_delete

Delete an SMTP recipient Instance

Prototype

```
UINT    nx_smtp_recipient_delete(NX_SMTP_SESSION *session_ptr,
                                 NX_SMTP_RECIPIENT *recipient_ptr
```

Description

This service deletes an SMTP recipient instance. It releases all memory dynamically allocated for the recipient instance back to the Client's memory pool.

Input Parameters

session_ptr	Pointer to the session deleting the recipient
recipient_ptr	Pointer to the recipient being deleted

Return Values

NX_SUCCESS	(0x00)	Recipient successfully deleted
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Delete the current recipient in this session. */
status = nx_smtp_recipient_delete(session_ptr, next_recipient_ptr);

/* If an SMTP recipient instance was successfully deleted, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_add, nx_smtp_mail_delete, nx_smtp_recipient_create, nx_smtp_recipient_add, nx_smtp_mail_message_append, nx_smtp_mail_message_process, nx_smtp_session_run, nx_smtp_bytepool_memory_release

nx_smtp_recipient _add

Add SMTP recipient Instance to an SMTP Session mail instance

Prototype

```
UINT    nx_smtp_recipient_add(NX_SMTP_MAIL *mail_ptr,
                              NX_SMTP_RECIPIENT *recipient_ptr
```

Description

This service adds an SMTP recipient instance to the recipient list of an SMTP mail instance.

Input Parameters

mail_ptr	Pointer to SMTP mAIL adding the recipient
recipient_ptr	Pointer to the recipient instance being added

Return Values

NX_SUCCESS	(0x00)	Recipient successfully added
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_SMTP_MAIL_RECIPIENT_ERROR	(0xB5)	Error in mail recipient linked list

Allowed From

Threads

Example

```
   /* Add recipient to this mail instance. */
   status =  nx_smtp_recipient_add(mail_ptr, recipient_ptr);

  /* If an SMTP recipient instance was added to the current mail, status is NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_add, nx_smtp_mail_delete, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_mail_message_append, nx_smtp_mail_message_process, nx_smtp_session_run

nx_smtp_mail_message _append

Append mail message to an SMTP Mail Instance

Prototype

```
UINT    nx_smtp_mail_message_append(NX_SMTP_CLIENT_SESSION *session_ptr,
                                    NX_SMTP_CLIENT_MAIL *mail_ptr,
                                    CHAR *message_in_mail, ULONG message_length)
```

Description

This service adds the mail message, including header and body, to the mail message member of an SMTP Mail instance. Message data is stored in chained message segments which allocate blocks of memory from the Client block pool.

Input Parameters

session_ptr	Pointer to session containing the mail instance
mail_ptr	Pointer to SMTP MAIL containing message data
message_in_mail	Pointer to message data
message_length	Length of message data in bytes

Return Values

NX_SUCCESS	(0x00)	Message data successfully added
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Attach message text of specified length to this mail. */
status =  nx_smtp_mail_message_appendd(session_ptr, mail_ptr, message, message_length);

/* If message data was added to the mail item, status = NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_add, nx_smtp_mail_delete, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_mail_message_process, nx_smtp_session_run, nx_smtp_bytepool_memory_get, nx_smtp_bytepool_memory_release, nx_smtp_blockpool_memory_get

nx_smtp_mail_message _process

Process SMTP Mail Instance message data

Prototype

```
UINT    nx_smtp_mail_message_process(NX_SMTP_CLIENT_SESSION *session_ptr,
                                     NX_SMTP_CLIENT_MAIL *mail_ptr)
```

Description

This service checks for the required SMTP end of message sequence at the end of the mail message. If not found, the message data is invalid, and the mail cannot be sent.

Input Parameters

session_ptr	Pointer to session containing the mail instance
mail_ptr	Pointer to SMTP MAIL containing message data

Return Values

NX_SUCCESS	(0x00)	Message data is valid
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_SMTP_CODE_PARAMETER_SYNTAX_ERROR	(501)	Mail message data is invalid

Allowed From

Application code, Threads

Example

```
/* Verify mail message data is valid (terminated with an end of message sequence) */
status =  nx_smtp_mail_message_process(session_ptr, mail_ptr);

/* If the message data is terminated with the correct end of message sequence, status =
    NX_SUCCESS. */
```

See Also

nx_smtp_client_create, nx_smtp_client_delete, nx_smtp_session_initialize, nx_smtp_session_reinitialize, nx_smtp_session_delete, nx_smtp_mail_create, nx_smtp_mail_add, nx_smtp_mail_delete, nx_smtp_recipient_create, nx_smtp_recipient_delete, nx_smtp_mail_message_append, nx_smtp_session_run

nx_smtp_session_run

Run an SMTP Session to deliver mail

Prototype

```
UINT    nx_smtp_session_run(NX_SMTP_SESSION *session_ptr)
```

Description

This service is the default NetX SMTP Client protocol engine. It conducts an SMTP session after connecting to an SMTP Server and carries out mail transaction with the Server. A successful session status is one that completes normally by issuing the QUIT command, regardless whether the actual mail transaction is accepted by the SMTP Server.

The application code can write its own session run function and access NetX SMTP command and reply handling services directly.

Input Parameters

session_ptr	Pointer to SMTP Session instance

Return Values

NX_SUCCESS	(0x00)	Session successfully completed
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Execute an SMTP session for Client requesting mail delivery. */
status = nx_smtp_session_run(NX_SMTP_SESSION* session_ptr);

/* If an SMTP Session completes by executing a QUIT command, status = NX_SUCCESS. */
```

See Also

nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_mail_message_append, nx_smtp_mail_message_process

nx_smtp_cmd_greeting

Send greeting to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_greeting(NX_SMTP_SESSION *session_ptr)
```

Description

This service sets the SMTP session state to await the Server greeting after Client session connects with an SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Successful connection to Server
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the GREETING command. */
status = nx_smtp_cmd_greeting(session_ptr);

/* If the SMTP Client connects with the SMTP Server, status = NX_SUCCESS. */
```

See Also

nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run

nx_smtp_rsp_greeting

Handle SMTP Server response to greeting

Prototype

```
UINT    nx_smtp_rsp_greeting(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server greeting, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Successful reply handling
NX_PTR_ERROR	(0x16)	Invalid SMTP Session pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Receive Server reply to Client greeting. */
status = nx_smtp_rsp_greeting(session_ptr);

/* If the session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_cmd_ehlo

Send EHLO to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_ehlo(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the EHLO command to the SMTP Server and sets the session state to await the Server reply to the EHLO command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	EHLO command successfully sent
NX_PTR_ERROR	(0x16)	Invalid SMTP Session pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the EHLO command. */
status = nx_smtp_cmd_ehlo(session_ptr);

/* If the EHLO command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_ehlo

Handle SMTP Server response to EHLO command

Prototype

```
UINT    nx_smtp_rsp_ehlo(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the EHLO command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client EHLO command. */
status = nx_smtp_rsp_ehlo (session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_cmd_helo

Send HELO to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_helo(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the HELO command to the SMTP Server and sets the session state to await the Server reply to the HELO command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	HELO command successfully sent
NX_PTR_ERROR	(0x16)	Invalid SMTP Session pointer
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the HELO command. */
status = nx_smtp_cmd_helo(session_ptr);

/* If the HELO command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_rsp_helo, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_helo

Handle SMTP Server response to HELO command

Prototype

```
UINT    nx_smtp_rsp_helo(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the HELO command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client HELO command. */
status = nx_smtp_rsp_helo (session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_helo, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_cmd_auth

Send AUTH command to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_auth(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the AUTH command to the SMTP Server and sets the session state to await the Server reply to the AUTH command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	MAIL command successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Send the AUTH command. */
status = nx_smtp_cmd_auth(session_ptr);

/* If the AUTH command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_auth

Handle SMTP Server response to AUTH command

Prototype

```
UINT    nx_smtp_rsp_auth(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the AUTH command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will find a matching authentication with the Server, initiate the authentication process, and determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client AUTH command. */
status = nx_smtp_rsp_auth(session_ptr);

 /* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code, nx_smtp_utility_set_next_auth_type, nx_smtp_utility_authentication_challenge

nx_smtp_cmd_auth_challenge

Send response to SMTP Server authentication challenge

Prototype

```
UINT    nx_smtp_cmd_auth_challenge(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends a response to the authentication challenge from the SMTP Server and sets the session state to await the Server reply to the session authentication response.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Response successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Send response to Server's authentication challenge. */
status = nx_smtp_cmd_auth_challenge(session_ptr);

 /* If the resposne was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server, nx_smtp_base64_encode

nx_smtp_rsp_auth_challenge

Handle next SMTP Server authentication challenge

Prototype

```
UINT    nx_smtp_rsp_auth_challenge(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the next authentication challenge, extracts the Server reply code and decodes the Server prompt. If the session *default_reply_handler* is enabled, it will respond to the challenge, and determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server authentication challenge. */
status = nx_smtp_rsp_auth_challenge(session_ptr);

 /* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code,

nx_smtp_utility_set_next_auth_type,
nx_smtp_utility_authentication_challenge

nx_smtp_cmd_mail

Send MAIL command to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_mail(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the MAIL command to the SMTP Server and sets the session state to await the Server reply to the MAIL command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	MAIL command successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Send the MAIL command. */
status = nx_smtp_cmd_mail(session_ptr);

/* If the MAIL command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_mail

Handle SMTP Server response to MAIL command

Prototype

```
UINT    nx_smtp_rsp_rcpt(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the MAIL command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client MAIL command. */
status = nx_smtp_rsp_rcpt(session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_cmd_rcpt

Send RCPT command to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_rcpt(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the RCPT command to the SMTP Server and sets the session state to await the Server reply to the RCPT command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	RCPT command successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the RCPT command. */
status = nx_smtp_cmd_rcpt(session_ptr);

 /* If the RCPT command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_rcpt

Handle SMTP Server response to RCPT command

Prototype

```
UINT    nx_smtp_rsp_rcpt(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the RCPT command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client RCPT command. */
status = nx_smtp_rsp_rcpt(session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_cmd_data

Send DATA command to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_data(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the DATA command to the SMTP Server and sets the session state to await the Server reply to the DATA command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	DATA command successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the DATA command. */
status = nx_smtp_cmd_data(session_ptr);

/* If the DATA command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_data

Handle SMTP Server response to DATA command

Prototype

```
UINT    nx_smtp_rsp_data(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the DATA command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client DATA command. */
status = nx_smtp_rsp_data (session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop,nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_cmd_message

Send message data to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_message(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the mail message data to the SMTP Server and sets the session state to await the Server reply to receiving the message data. It assumes mail message is stored in the SMTP Client mail mail_buffer_ptr parameter, not in a chained linked list of message struct items (see *nx_smtp_cmd_message_memalloc* and *nx_smtp_mail_message_append*), when loading Client packets with the mail data to send to the Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Message data successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the message data. */
status = nx_smtp_cmd_message(session_ptr);

/* If the message data was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run

nx_smtp_cmd_message_memalloc

Send message data to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_message_memalloc(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the mail message data to the SMTP Server and sets the session state to await the Server reply to receiving the message data. It assumes mail message is stored in a chained linked list of message struct items (see *nx_smtp_cmd_message_memalloc* and *nx_smtp_mail_message_append*), not the SMTP Mail mail_buffer_ptr parameter, when loading Client packets with the mail data to send to the Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Message data successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the message data. */
status = nx_smtp_cmd_message_memalloc(session_ptr);

 /* If the message data was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run

nx_smtp_rsp_message

Handle SMTP Server response to message data

Prototype

```
UINT    nx_smtp_rsp_message(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to receiving the mail message data, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to receiving mail message data. */
status = nx_smtp_rsp_message(session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop,nx_smtp_rsp_noop, nx_smtp_session_run, , nx_smtp_utility_read_server_code

nx_smtp_cmd_quit

Send QUIT command to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_quit(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the QUIT command to the SMTP Server and sets the session state to await the Server reply to receiving the QUIT command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	QUIT command successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the QUIT command. */
status = nx_smtp_cmd_quit(session_ptr);

/* If the QUIT command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_quit

Handle SMTP Server response to QUIT command

Prototype

```
UINT    nx_smtp_rsp_quit(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the QUIT command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it sets the session termination status (normal vs abnormal).

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client QUIT command. */
status = nx_smtp_rsp_quit(session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop,nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_cmd_rset

Send RSET command to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_rset(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the RSET command to the SMTP Server and sets the session state to await the Server reply to receiving the RSET command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	RSET command successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the RSET command. */
status = nx_smtp_cmd_rset(session_ptr);

/* If the RSET command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_rset

Handle SMTP Server response to RSET command

Prototype

```
UINT    nx_smtp_rsp_rset(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the RSET command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client RSET command. */
status = nx_smtp_rsp_rset(session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_cmd_noop, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_cmd_noop

Send NOOP command to an SMTP Server

Prototype

```
UINT    nx_smtp_cmd_noop(NX_SMTP_SESSION *session_ptr)
```

Description

This service sends the NOOP command to the SMTP Server and sets the session state to await the Server reply to receiving the NOOP command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	NOOP command successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the NOOP command. */
status = nx_smtp_cmd_noop(session_ptr);

/* If the NOOP command was sent successfully, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting, nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_rsp_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_rsp_noop, nx_smtp_session_run, nx_smtp_utility_send_to_server

nx_smtp_rsp_noop

Handle SMTP Server response to NOOP command

Prototype

```
UINT    nx_smtp_rsp_noop(NX_SMTP_SESSION *session_ptr)
```

Description

This service receives the Server reply to the NOOP command, extracts the Server reply code and stores the text of the reply to the session. If the session *default_reply_handler* is enabled, it will determine the next session command to send to the SMTP Server.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully received
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Receive Server reply to Client NOOP command. */
status = nx_smtp_rsp_noop(session_ptr);

/* If session receives a valid Server reply, status = NX_SUCCESS. */
```

See Also

nx_smtp_cmd_greeting, nx_smtp_rsp_greeting nx_smtp_cmd_ehlo, nx_smtp_rsp_ehlo, nx_smtp_cmd_auth, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_cmd_mail, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_cmd_data, nx_smtp_rsp_data, nx_smtp_cmd_message, nx_smtp_rsp_message, nx_smtp_cmd_quit, nx_smtp_cmd_rset, nx_smtp_rsp_rset, nx_smtp_cmd_noop, nx_smtp_session_run, nx_smtp_utility_read_server_code

nx_smtp_bytepool_memory_get

Allocate memory from Client byte pool

Prototype

```
UINT nx_smtp_bytepool_memory_get(VOID **memory_ptr,
                          ULONG memory_size, TX_BYTE_POOL *bytepool_ptr,
                          TX_MUTEX *mutex_ptr, UINT mutex_wait_option)
```

Description

This service obtains an exclusive lock on the previously created SMTP Client byte pool, then allocates the specified amount of memory from it.

Input Parameters

memory_ptr	Pointer to location where memory is allocated
memory_size	Amount of memory in bytes to allocate
bytepool_ptr	Pointer to Client byte pool
mutex_ptr	Pointer to Client byte pool mutex
mutex_wait_option	Timeout option on obtaining byte pool mutex

Return Values

NX_SUCCESS	(0x00)	Memory successfully allocated
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_SMTP_PARAM_ERROR	(0xB2)	Parsing error (zero bytes requested)

Allowed From

Threads

Example

```
* Allocate memory for a mail instance.  */
status = nx_smtp_bytepool_memory_get((VOID **)session_mail_ptr,
                sizeof(NX_SMTP_CLIENT_MAIL),
                client_ptr -> bytepool_ptr, client_ptr -> bytepool_mutex_ptr,
                client_ptr -> bytepool_mutex_timeout);

/* If memory was successfully allocated, status = NX_SUCCESS. */
```

See Also

nx_smtp_bytepool_memory_release, nx_smtp_blockpool_memory_get, nx_smtp_blockpool_memory_release

nx_smtp_bytepool_memory_release

Release memory back to Client byte pool

Prototype

```
UINT    nx_smtp_bytepool_memory_release(VOID *memory_ptr,
                               TX_BYTE_POOL *bytepool_ptr,
                               TX_MUTEX *mutex_ptr,
                               UINT mutex_wait_option)
```

Description

This service obtains an exclusive lock on the previously created SMTP Client byte pool, then releases the memory allocated to the specified location back to the byte pool.

Input Parameters

memory_ptr	Pointer to location of allocated memory
bytepool_ptr	Pointer to Client byte pool
mutex_ptr	Pointer to Client byte pool mutex
mutex_wait_option	Timeout option on obtaining byte pool mutex

Return Values

NX_SUCCESS	(0x00)	Memory successfully released
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
* Allocate memory for a mail instance.  */
status = nx_smtp_bytepool_memory_release(mail_ptr, client_ptr -> bytepool_ptr,
                              client_ptr -> bytepool_mutex_ptr,
                              client_ptr -> bytepool_mutex_timeout);

/* If memory was successfully released, status = NX_SUCCESS. */
```

See Also

nx_smtp_bytepool_memory_get, nx_smtp_blockpool_memory_get, nx_smtp_blockpool_memory_release

nx_smtp_blockpool_memory_get

Allocated memory from Client block pool

Prototype

```
UINT nx_smtp_blockpool_memory_get(VOID **memory_ptr,
                              TX_BLOCK_POOL *blockpool_ptr,
                              TX_MUTEX *mutex_ptr,
                              UINT mutex_wait_option)
```

Description

This service obtains an exclusive lock on the previously created SMTP Client block pool, then allocates a block of memory from the block pool.

Input Parameters

memory_ptr	Pointer to location where memory is allocated
blockpool_ptr	Pointer to Client block pool
mutex_ptr	Pointer to Client block pool mutex
mutex_wait_option	Timeout option on obtaining block pool mutex

Return Values

NX_SUCCESS	(0x00)	Memory successfully allocated
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Allocate a block of memory from Client block pool for segment data. */
status = nx_smtp_blockpool_memory_get(
                       (VOID **)&message_segment_ptr -> message_ptr,
                       client_ptr -> blockpool_ptr,
                       client_ptr -> blockpool_mutex_ptr,
                       client_ptr -> blockpool_mutex_timeout);

/* If memory was successfully allocated, status = NX_SUCCESS. */
```

See Also

nx_smtp_bytepool_memory_get, nx_smtp_bytepool_memory_release, nx_smtp_blockpool_memory_release

nx_smtp_blockpool_memory_release

Release memory back to Client block pool

Prototype

```
UINT    nx_smtp_blockpool_memory_release(VOID *memory_ptr,
                                 TX_BLOCK_POOL *blockpool_ptr,
                                 TX_MUTEX *mutex_ptr,
                                 UINT mutex_wait_option)
```

Description

This service obtains an exclusive lock on the previously created SMTP Client block pool, then releases the memory allocated to the specified location back to the block pool.

Input Parameters

memory_ptr	Pointer to location of allocated memory
blockpool_ptr	Pointer to Client block pool
mutex_ptr	Pointer to Client block pool mutex
mutex_wait_option	Timeout option on obtaining block pool mutex

Return Values

NX_SUCCESS	(0x00)	Memory successfully released
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Release memory for a mail instance.  */
status = nx_smtp_blockpool_memory_release(mail_ptr, client_ptr -> blockpool_ptr,
                              client_ptr -> blockpool_mutex_ptr,
                              client_ptr -> blockpool_mutex_timeout);

/* If memory was successfully released, status = NX_SUCCESS. */
```

See Also

nx_smtp_bytepool_memory_get, nx_smtp_bytepool_memory_release, nx_smtp_blockpool_memory_get

nx_smtp_utility_read_server_reply_code

Read the SMTP Server reply and extract the reply code

Prototype

```
UINT  nx_smtp_utility_read_server_code(
                        NX_SMTP_CLIENT_SESSION *session_ptr,
                        ULONG timeout, UINT  receive_all_lines)
```

Description

This service receives the Server reply in TCP packets, extracts the Server reply code from the packet buffer and stores the text of the reply to its session buffer.

Input Parameters

session_ptr	Pointer to Client session receiving reply
`timeout`	How long to wait to receive Server reply
`receive_all_lines`	Option to receive one line at a time or all.

Return Values

NX_SUCCESS	(0x00)	Server reply successfully extracted
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Get the Server greeting message. */
status = nx_smtp_utility_read_server_code(session_ptr, timeout, NX_TRUE);

/* If a reply was successfully received and parsed, status = NX_SUCCESS. */
```

See Also

nx_smtp_utility_send_to_server, nx_smtp_rsp_greeting, nx_smtp_rsp_ehlo, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_rsp_data, nx_smtp_rsp_message, nx_smtp_rsp_quit, nx_smtp_rsp_rset, nx_smtp_rsp_noop

nx_smtp_utility_send_to_server

Send message or command to the SMTP Server

Prototype

```
UINT  nx_smtp_utility_send_to_server (
                        NX_SMTP_CLIENT_SESSION *session_ptr,
                        CHAR *buffer_ptr,
                        UINT buffer_length, ULONG timeout)
```

Description

This service receives the Server reply in TCP packets, extracts the Server reply code from the packet buffer and stores the text of the reply to its session buffer.

Input Parameters

session_ptr	Pointer to Client session sending message
buffer_ptr	Pointer to message to send
buffer_length	Size of message in bytes
timeout	How long to wait to send message

Return Values

NX_SUCCESS	(0x00)	Message was successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send the EHLO command.  */
status =  nx_smtp_utility_send_to_server(session_ptr, message,
                                          strlen(message), timeout);

/* If the message was successfully sent, status = NX_SUCCESS. */
```

See Also

nx_smtp_utility_send_to_server, nx_smtp_rsp_greeting, nx_smtp_rsp_ehlo, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_rsp_data, nx_smtp_rsp_message, nx_smtp_rsp_quit, nx_smtp_rsp_rset, nx_smtp_rsp_noop

nx_smtp_utility_parse_server_services

Parse Server extension services in SMTP Server message

Prototype

```
UINT nx_smtp_utility_parse_server_services(
                              NX_SMTP_CLIENT_SESSION *session_ptr)
```

Description

This service extracts extension SMTP services, such as authentication, from the SMTP Server EHLO reply that is stored in its session buffer and updates the Client session that these services are available.

Input Parameters

session_ptr	Pointer to Client session receiving Server services

Return Values

NX_SUCCESS	(0x00)	Server reply parsed successfully
NX_PTR_ERROR	(0x16)	Invalid pointer parameter

Allowed From

Application code

Example

```
/* Update session with specific services offered, if any, by Server. */
status = nx_smtp_utility_parse_server_services(session_ptr);

/* If buffer parsed successfully with no errors, status = NX_SUCCESS. */
```

See Also

nx_smtp_utility_read_server_code, nx_smtp_utility_send_to_server, nx_smtp_rsp_ehlo

nx_smtp_utility_data_transparency

Apply data transparency processing to Client mail message

Prototype

```
UINT  nx_smtp_utility_data_transparency (CHAR *buffer, UINT *length,
                                         UINT additional_buffer)
```

Description

This service applies data transparency processing on the Client mail message. In this process, lines beginning with a period (0x2E) have a duplicate period inserted to avoid being misinterpreted as part of an end of message sequence. The duplicate 'dot' is removed by the receiving mail user agent.

Input Parameters

buffer_ptr	Pointer to message to process
length	Size of message in bytes
additional_buffer	Number of bytes the caller allows for inserting duplicate periods

Return Values

NX_SUCCESS	(0x00)	Buffer processed successfully
NX_PTR_ERROR	(0x16)	Invalid pointer parameter

Allowed From

Application code

Example

```
/* Process next chunk of text. This may add a couple bytes to message size. */

status = nx_smtp_utility_data_transparency(message_segment_ptr -> message_ptr,
                              &bytes_to_store,
                              session_ptr ->data_transparency_bytes);

/* If message successfully processed, status = NX_SUCCESS. */
```

See Also

nx_smtp_rsp_greeting, nx_smtp_rsp_ehlo, nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge, nx_smtp_rsp_mail, nx_smtp_cmd_rcpt, nx_smtp_rsp_rcpt, nx_smtp_rsp_data, nx_smtp_rsp_message, nx_smtp_rsp_quit,

nx_smtp_rsp_rset, nx_smtp_rsp_noop, nx_smtp_session_run,
nx_smtp_mail_message_append, nx_smtp_mail_message_process

nx_smtp_utility_authentication_challenge

Handles the Server authentication challenge to the SMTP Client

Prototype

```
UINT nx_smtp_utility_authentication_challenge(
                    NX_SMTP_CLIENT_SESSION *session_ptr,
                    CHAR *server_challenge, UINT length)
```

Description

This service handles parsing and decoding the SMTP Server challenge and setting up the Client response to the challenge.

Input Parameters

session_ptr	Pointer to Client session being authenticated
server_challenge	Pointer to authentication challenge from Server
length	Size in bytes of the authentication string

Return Values

NX_SUCCESS	(0x00)	Server reply parsed successfully
NX_SMTP_PARAM_ERROR	(0xB2)	Parsing error (empty challenge buffer)

Allowed From

Application code

Example

```
/* Process the Server's challenge. */
status = nx_smtp_utility_authentication_challenge(session_ptr,
                    session_ptr -> reply_buffer,
                    CRLF_ptr - session_ptr -> reply_buffer);

/* If the challenge was successfully decoded, status = NX_SUCCESS. */
```

See Also

nx_smtp_utility_read_server_code, nx_smtp_utility_send_to_server, nx_smtp_cmd_auth , nx_smtp_rsp_auth, nx_smtp_cmd_auth_challenge, nx_smtp_rsp_auth_challenge

nx_smtp_utility_print_client_mail_status

Displays the contents and details of an SMTP Client mail

Prototype

```
UINT nx_smtp_utility_print_client_mail_status(
                              NX_SMTP_CLIENT_MAIL *mail_ptr)
```

Description

This service displays the contents of specified mail message including mail envelope, and miscellaneous mail transaction data. While it is not required, the application should obtain the Client print mutex if calling from a thread task to keep the display of session mail data discreet among concurrent Client session tasks.

Input Parameters

mail_ptr	Pointer to Client mail to display

Return Values

NX_SUCCESS	(0x00)	Mail data displayed successfully
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Application code

Example

```
/* Display mail item's data. */
status = nx_smtp_utility_print_client_mail_status(
                                             session_ptr -> current_mail_ptr);

/* If mail data displayed successfully, status = NX_SUCCESS.  */
```

See Also

nx_smtp_utility_print_client_reserves

nx_smtp_utility_print_client_reserves

Displays SMTP Client memory and packet pool reservers

Prototype

```
UINT nx_smtp_utility_print_client_reserves(
                                NX_SMTP_CLIENT *client_ptr)
```

Description

This service displays the remaining bytes and blocks available in the Client byte and block pools and remaining packets in the Client packet pool. While it is not required, the application should obtain the Client print mutex if calling from a thread task to keep the display of Client reserves discreet among concurrent tasks.

Input Parameters

client_ptr	Pointer to the SMTP Client

Return Values

NX_SUCCESS	(0x00)	Client data displayed successfully
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Application code

Example

```
/* Display the status of Client packet and memory pool reserves.  */
Status = nx_smtp_utility_print_client_reserves(session_ptr -> client_ptr);

/* If client data displayed successfully, status = NX_SUCCESS.  */
```

See Also

nx_smtp_utility_print_client_mail_status

Chapter 4 Description of SMTP Server Services

Services for Server Session and Mail Setup

nx_smtp_server_create
Create an SMTP Server Instance

nx_smtp_server_delete
Delete an SMTP Server instance

nx_smtp_server_session_create
Create an SMTP Server session instance

nx_smtp_server_session_delete
Delete an SMTP Server session instance

nx_smtp_server_mail_create
Create an SMTP Mail instance

nx_smtp_server_mail_delete
Delete an SMTP Mail instance

nx_smtp_server_mail_add
Add an SMTP mail instance to Server session

nx_smtp_server_recipient_create
Create an SMTP Recipient instance

nx_smtp_server_recipient_delete
Delete an SMTP Recipient instance

nx_smtp_server_recipient_add
Add recipient to SMTP Mail instance

Services for the SMTP Protocol and Server State Machine

nx_smtp_server_session_run
Run a SMTP session to transact SMTP mail delivery

nx_smtp_reply_to_greeting

Reply to greeting from SMTP Client

nx_smtp_reply_to_ehlo
Reply to SMTP Client EHLO command

nx_smtp_reply_to_helo
Reply to SMTP Client HELO command

nx_smtp_reply_to_auth
Reply to SMTP Client AUTH command

nx_smtp_reply_to_mail
Reply to SMTP Client MAIL command

nx_smtp_reply_to_rcpt
Reply to SMTP Client RCPT command

nx_smtp_reply_to_data
Reply to SMTP Client DATA command

nx_smtp_reply_to_message
Reply to receiving SMTP Client mail message

nx_smtp_reply_to_quit
Reply to SMTP Client QUIT command

nx_smtp_reply_to_rset
Reply to SMTP Client RSET command

nx_smtp_reply_to_noop
Reply to SMTP Client NOOP command

Services for Server Memory Management

nx_smtp_server_bytepool_memory_get
Allocate memory from Server byte pool

nx_smtp_server_bytepool_memory_release
Release memory back to Server byte pool

nx_smtp_server_blockpool_memory_get
Allocate memory from Server block pool

nx_smtp_server_blockpool_memory_release

Release memory back to Server block pool

Server Utility services

nx_smtp_utility_send_server_response
Send Server reply to SMTP Client

nx_smtp_utility_parse_client_request
Parse SMTP Client command

nx_smtp_server_parse_mailbox_address
Parse mailbox address from Client command

nx_smtp_utility_login_authenticate
Encode Server challenge; decode Client response

nx_smtp_utility_spool_session_mail
Spool Server session mail to hard disk

nx_smtp_utility_clear_session_mail
Delete Server session mail and release allocated memory

nx_smtp_utility_print_server_mail
Display the contents and mail transaction data of a specified Server session mail item

nx_smtp_utility_ print_server_reserves
Display available Server byte pool and block pool memory and remaining packets in Server packet pool

nx_smtp_server_create

Create an SMTP Server Instance

Prototype

```
UINT  nx_smtp_server_create(NX_SMTP_SERVER *server_ptr,
            CHAR *authentication_list, NX_IP *ip_ptr,
            VOID *stack_ptr, ULONG stack_size,
            UINT server_priority, UINT server_preempt_threshold,
            UINT server_time_slice, UINT auto_start,
            NX_PACKET_POOL *packet_pool_ptr,
            UINT (*authentication_check)
                  (CHAR *username, CHAR *password, UINT *result),
            UINT (*mail_spooler)(NX_SMTP_SERVER_MAIL *mail_ptr),
            TX_BLOCK_POOL *blockpool_ptr,
            TX_MUTEX *blockpool_mutex_ptr,
            UINT blockpool_mutex_timeout,
            TX_BYTE_POOL *bytepool_ptr,
            TX_MUTEX *bytepool_mutex_ptr,
            UINT bytepool_mutex_timeout)
```

Description

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

Input Parameters

server_ptr	Pointer to SMTP Server control block
authentication_list	Pointer to list of authentication type supported by the SMTP Server
ip_ptr	Pointer to Server IP instance
stack_ptr	Pointer to stack location of Server thread
stack_size	Size of stack memory for the Server
server_priority	Server thread priority
server_preemp_threshold	Server thread preemption threshold
server_time_slice	Time slice allocated by the scheduler for Server thread execution
auto_start	Server thread start option
packet_pool_ptr	Pointer to Server packet pool
authentication_check	Pointer to the Server authentication check service
mail_spooler	Pointer to the Server mail spooler service
blockpool_ptr	Pointer to Server block pool
blockpool_mutex_ptr	Pointer to Server block pool mutex
blockpool_mutex_timeout	Time to wait to Server Client block pool mutex
bytepool_ptr	Pointer to Server byte pool
bytepool_mutex_ptr	Pointer to Server byte pool mutex
bytepool_mutex_timeout	Time to wait to obtain Server byte pool mutex

Return Values

NX_SUCCESS	(0x00)	SMTP Server successfully created.
NX_PTR_ERROR	(0x16)	Invalid input pointer parameter.

Allowed From

Application Code

Example

```
/* Create the SMTP Server instance. */

status = nx_smtp_server_create(&demo_server, "LOGIN", &server_ip,
                    free_memory_pointer, 2048, NX_SMTP_SERVER_THREAD_PRIORITY,
                    NX_SMTP_SERVER_THREAD_PRIORITY, NX_SMTP_SERVER_THREAD_TIME_SLICE,
                    TX_DONT_START, &server_packet_pool, server_authentication_check,
                    server_mail_spooler, &server_block_pool,
                    &server_block_pool_mutex,
                    NX_SMTP_SERVER_BLOCK_POOL_MUTEX_WAIT, &server_byte_pool,
                    &server_byte_pool_mutex, NX_SMTP_SERVER_BYTE_POOL_MUTEX_WAIT);

/* If a Server was successfully created, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_delete, nx_smtp_server_session_create, nx_smtp_server_session_initialize, nx_smtp_server_session_reinitialize, nx_smtp_server_session_delete, nx_smtp_server_mail_create, nx_smtp_server_mail_delete, nx_smtp_server_mail_add, nx_smtp_server_recipient_create, nx_smtp_server_recipient_delete, nx_server_smtp_recipient_add, nx_smtp_server_message_segment_add, nx_smtp_server_session_run, nx_smtp_session_connection_present, nx_smtp_session_disconnect_present

nx_smtp_server_delete

Delete an SMTP Server Instance

Prototype

```
UINT nx_smtp_server_delete(NX_SMTP_SERVER *server_ptr);
```

Description

This service deletes a previously created SMTP Server instance.

Input Parameters

client_ptr	Pointer to SMTP Server instance.

Return Values

NX_SUCCESS	(0x00)	Server successfully deleted.
NX_PTR_ERROR	(0x16)	Invalid input pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Delete the SMTP Server instance "my_server." */
status = nx_smtp_server_delete(&my_server);

/* If the SMTP Server instance was successfully deleted, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_session_create, nx_smtp_server_session_delete, nx_smtp_server_mail_create, nx_smtp_server_mail_delete, nx_smtp_server_mail_add, nx_smtp_server_recipient_create, nx_smtp_server_recipient_delete, nx_server_smtp_recipient_add, nx_smtp_server_message_segment_add, nx_smtp_server_session_run, nx_smtp_session_connection_present, nx_smtp_session_disconnect_present

nx_smtp_server_session_create

Create an SMTP Server Session Instance

Prototype

```
UINT  nx_smtp_server_session_create(NX_SMTP_SERVER *server_ptr,
                NX_SMTP_SERVER_SESSION *session_ptr,
                UINT session_id, VOID *session_stack_ptr,
                ULONG session_stack_size, UINT session_priority,
                 UINT session_preempt_threshold,
                ULONG session_time_slice,
                UINT session_auto_start)
```

Description

This service creates an SMTP Session instance to accept an SMTP Client connection and conduct an SMTP mail transaction.

Input Parameters

server_ptr	Pointer to session's Server
session_ptr	Pointer to SMTP Server session to create
session_id	ID assigned to session
session_stack_ptr	Location of Server session on the stack
session_stack_size	Size of Server session stack memory
session_priority	Server session thread priority
session_preempt_theshold	Server session thread preemption threshold
session_time_slice	Time slice the scheduler allots for Server session thread execution
Session_auto_start	Server session thread start option

Return Values

NX_SUCCESS	(0x00)	Server session successfully created
NX_PTR_ERROR	(0x16)	Invalid input pointer parameter

Allowed From

Application code

Example

```
/* Create the SMTP Server Session instance. */
status =  nx_smtp_server_session_create(session_ptr, 0x01, server_ptr, server_ip_address,
                                        smtp_session_port);


/* If a SMTP Server session instance was successfully created, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_delete, nx_smtp_server_mail_create,
nx_smtp_server_mail_delete, nx_smtp_server_mail_add,
nx_smtp_server_recipient_create, nx_smtp_server_recipient_delete,
nx_server_smtp_recipient_add, nx_smtp_server_message_segment_add,
nx_smtp_server_session_run, nx_smtp_session_connection_present,
nx_smtp_session_disconnect_present

nx_smtp_server_session_delete

Delete an SMTP Server session Instance

Prototype

```
UINT    nx_smtp_server_session_delete(NX_SMTP_SERVER_SESSION
                                       *session_ptr)
```

Description

This service deletes an SMTP Server session instance and all mail and recipients associated with the session. It releases all memory dynamically allocated for the session instance back to the Server's memory pools.

Input Parameters

session_ptr	Pointer to SMTP Server session to delete

Return Values

NX_SUCCESS	(0x00)	Server session successfully deleted
NX_CALLER_ERROR	(0x11)	Invalid caller of this service
NX_PTR_ERROR	(0x16)	Invalid pointer parameter

Allowed From

Threads

Example

```
/* Delete the SMTP Server session instance. */
 status =  nx_smtp_server_session_delete(session_ptr);


/* If a Server session instance was successfully deleted, status is NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_mail_create,
nx_smtp_server_mail_delete, nx_smtp_server_mail_add,
nx_smtp_server_recipient_create, nx_smtp_server_recipient_delete,
nx_server_smtp_recipient_add, nx_smtp_server_message_segment_add,
nx_smtp_server_session_run, nx_smtp_session_connection_present,
nx_smtp_session_disconnect_present

nx_smtp_server_mail_create

Create an SMTP mail Instance

Prototype

```
UINT    nx_smtp_server_mail_create(NX_SMTP_SERVER_SESSION *session_ptr,
                                   NX_SMTP_SERVER_MAIL **mail_ptr,
                                   CHAR *reverse_path_ptr,
                                   UINT priority)
```

Description

This service creates an SMTP Server mail instance including session the mail is associated with, mail delivery priority and mailbox path of the original author.

Input Parameters

session_ptr	Pointer to SMTP Session creating the mail
mail_ptr	Pointer to location to create SMTP Mail instance
reverse_path_ptr	Pointer to string containing sender's mailbox
priority	Priority level at which mail is delivered

Return Values

NX_SUCCESS	(0x00)	Mail successfully created
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
 /* Create SMTP Server mail item. */
 status =  nx_smtp_server_mail_create(session_ptr, &mail_ptr,
                              reverse_path_mailbox_ptr, priority);

/* If an SMTP mail instance was successfully created, status is NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_session_delete,
nx_smtp_server_session_run, nx_smtp_server_mail_delete,
nx_smtp_server_mail_add, nx_smtp_server_recipient_create,
nx_smtp_server_recipient_delete, nx_server_smtp_recipient_add,
nx_smtp_server_message_segment_add,

nx_smtp_server_bytepool_memory_get,
nx_smtp_server_bytepool_memory_release

nx_smtp_server_mail_delete

Delete an SMTP Server mail Instance

Prototype

```
UINT    nx_smtp_server_mail_delete(NX_SMTP_SERVER_SESSION *session_ptr,
                           NX_SMTP_SERVER_MAIL *mail_ptr)
```

Description

This service deletes an SMTP Server mail instance and all recipient and message data associated with the mail. It releases all memory dynamically allocated for the mail instance back to the Server's byte and block pools.

Input Parameters

session_ptr	Pointer to SMTP Session deleting the mail
mail_ptr	Pointer to SMTP mail instance being deleted

Return Values

NX_SUCCESS	(0x00)	Mail successfully deleted
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Delete the current Server session mail instance. */
 status =  nx_smtp_server mail_delete(session_ptr, mail_ptr);

/* If SMTP Server mail instance was successfully deleted, status = NX_SUCCESS. */
```

See also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_session_delete,
nx_smtp_server_session_run, nx_smtp_server_mail_create,
nx_smtp_server_mail_add, nx_smtp_server_recipient_create,
nx_smtp_server_recipient_delete, nx_server_smtp_recipient_add,
nx_smtp_server_message_segment_add,
nx_smtp_server_bytepool_memory_get,
nx_smtp_server_bytepool_memory_release

nx_smtp_server_mail_add

Add an SMTP Server mail Instance to a Server session

Prototype

```
UINT    nx_smtp_server_mail_add(NX_SMTP_SERVER_SESSION *session_ptr,
                                NX_SMTP_SERVER_MAIL *mail_ptr)
```

Description

This service adds an SMTP Server mail instance to the Server session mail.

Input Parameters

session_ptr	Pointer to SMTP Session adding the mail item
mail_ptr	Pointer to SMTP mail instance being added

Return Values

NX_SUCCESS	(0x00)	Mail to successfully added to session
NX_PTR_ERROR	(0x16)	Invalid pointer parameter

Allowed From

Application code, Threads

Example

```
/* Add mail to Server session. */
status = nx_smtp_server_mail_add(session_ptr, mail_ptr);

/* If mail instance was successfully added, then status = NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_session_delete,
nx_smtp_server_session_run, nx_smtp_server_mail_delete,
nx_smtp_server_mail_create, nx_smtp_server_recipient_create,
nx_smtp_server_recipient_delete, nx_server_smtp_recipient_add,
nx_smtp_server_message_segment_add,
nx_smtp_server_bytepool_memory_get,
nx_smtp_server_bytepool_memory_release

nx_smtp_server_recipient_create

Create an SMTP Server recipient Instance

Prototype

```
UINT    nx_smtp_recipient_create(NX_SMTP_SERVER_SESSION *session_ptr,
                        NX_SMTP_SERVER_MAIL *mail_ptr,
                        NX_SMTP_SERVER_RECIPIENT **recipient_ptr,
                        CHAR *recipient_mailbox, UINT type)
```

Description

This service creates an SMTP Server recipient instance belonging to an SMTP Server mail instance.

Input Parameters

session_ptr	Pointer to the session the mail belongs to
mail_ptr	Pointer to SMTP mail creating the recipient
recipient_ptr	Pointer to location for creating recipient
recipient_mailbox	Pointer to recipient mailbox
type	Type of delivery: TO, CC or BCC

Return Values

NX_SUCCESS	(0x00)	Recipient successfully created
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Create instance of SMTP Server recipient. */
status =  nx_smtp_server_recipient_create(session_ptr, mail_ptr,
                    &recipient_ptr, recipient_mailbox,
                    NX_SMTP_RECIPIENT_TO);

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

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_session_delete,
nx_smtp_server_session_run, nx_smtp_server_mail_delete,
nx_smtp_server_mail_create, nx_smtp_server_mail_create,

nx_smtp_server_recipient_delete, nx_server_smtp_recipient_add,
nx_smtp_server_message_segment_add,
nx_smtp_server_bytepool_memory_get,
nx_smtp_server_bytepool_memory_release

nx_smtp_server_recipient_delete

Delete an SMTP Server recipient Instance

Prototype

```
UINT    nx_smtp_server_recipient_delete(
                       NX_SMTP_SERVER_SESSION *session_ptr,
                       NX_SMTP_SERVER_RECIPIENT *recipient_ptr
```

Description

This service deletes an SMTP Server recipient instance. It releases all memory dynamically allocated for the recipient instance back to the Client's memory pool.

Input Parameters

session_ptr	Pointer to the session deleting the recipient
recipient_ptr	Pointer to the recipient being deleted

Return Values

NX_SUCCESS	(0x00)	Recipient successfully deleted
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Delete the current recipient in this session. */
status = nx_smtp_server_recipient_delete(session_ptr, next_recipient_ptr);

/* If the SMTP Recipient instance was successfully deleted, status is NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_session_delete,
nx_smtp_server_session_run, nx_smtp_server_mail_delete,
nx_smtp_server_mail_create, nx_smtp_server_mail_create,
nx_smtp_server_recipient_create, nx_server_smtp_recipient_add,
nx_smtp_server_message_segment_add,
nx_smtp_server_bytepool_memory_get,
nx_smtp_server_bytepool_memory_release

nx_smtp_server_recipient _add

Add SMTP Server recipient Instance to an SMTP session mail

Prototype

```
UINT    nx_smtp_server_recipient_add(NX_SMTP_SERVER_MAIL *mail_ptr,
                                     NX_SMTP_SERVER_RECIPIENT *recipient_ptr
```

Description

This service adds an SMTP Server recipient instance to the recipient list of an SMTP Server Mail instance.

Input Parameters

mail_ptr	Pointer to SMTP MAIL adding the recipient
recipient_ptr	Pointer to the recipient instance being added

Return Values

NX_SUCCESS	(0x00)	Recipient successfully added
NX_PTR_ERROR	(0x16)	Invalid pointer parameter

Allowed From

Application code, Threads

Example

```
 /* Add recipient to this mail instance. */
 status =  nx_smtp_server_recipient_add(mail_ptr, recipient_ptr);

/* If an SMTP recipient instance was added to the current mail, status is NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_session_delete,
nx_smtp_server_session_run, nx_smtp_server_mail_delete,
nx_smtp_server_mail_create, nx_smtp_server_mail_create,
nx_smtp_server_recipient_create, nx_server_smtp_recipient_delete,
nx_smtp_server_message_segment_add,
nx_smtp_server_bytepool_memory_get,
nx_smtp_server_bytepool_memory_release

nx_smtp_server_message_segment_add

Add mail message segment to an SMTP Mail Instance

Prototype

```
UINT    nx_smtp_server_message_segment_add(NX_SMTP_SERVER_MAIL *mail_ptr,
                              NX_SMTP_MESSAGE_SEGMENT message_segment_ptr)
```

Description

This service adds an SMTP mail message segment containing part or all of the mail message data to an SMTP Server Mail instance. The complete mail message data is stored in one or more mail message segments chained together.

Input Parameters

mail_ptr	Pointer to SMTP Server MAIL instance
message_segment_ptr	Pointer to message segment

Return Values

NX_SUCCESS	(0x00)	Message segment successfully added
NX_PTR_ERROR	(0x16)	Invalid pointer parameter

Allowed From

Application code, Threads

Example

```
/* Attach message text of specified length to this mail. */
status =  nx_smtp_server_message_segment_add(mail_ptr, message_segment_ptr);

/* If message data was added to the mail item, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_session_delete,
nx_smtp_server_session_run, nx_smtp_server_mail_delete,
nx_smtp_server_mail_create, nx_smtp_server_mail_create,
nx_smtp_server_recipient_create, nx_server_smtp_recipient_delete,
nx_smtp_server_message_segment_add,
nx_smtp_server_blockpool_memory_get,
nx_smtp_server_blockpool_memory_release

nx_smtp_server_session_run

Run an SMTP session to receive mail

Prototype

```
UINT    nx_smtp_server_session_run(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service is the NetX SMTP Server protocol engine. It conducts an SMTP session after accepting an SMTP Client request for receiving Client mail. A successful session status is one that completes when the Client issues the QUIT command, regardless whether the actual mail transaction is accepted by the SMTP server.

Input Parameters

session_ptr	Pointer to SMTP Server session instance

Return Values

NX_SUCCESS	(0x00)	Session successfully completed
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
/* Execute an SMTP Server session with an SMTP Client. */
status = nx_smtp_server_session_run(NX_SMTP_SERVER_SESSION* session_ptr);

/* If an SMTP Session completes normally, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_create, nx_smtp_server_delete
nx_smtp_server_session_create, nx_smtp_server_session_delete,
nx_smtp_server_mail_create, nx_smtp_server_mail_delete,
nx_smtp_server_mail_add, nx_smtp_server_recipient_create,
nx_smtp_server_recipient_delete, nx_server_smtp_recipient_add,
nx_smtp_server_message_segment_add

nx_smtp_server_reply_to_greeting

Reply to SMTP Client greeting

Prototype

```
UINT    nx_smtp_server_reply_to_greeting(
                                   NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client greeting (connetion request), and replies with a Server reply code indicating if it can accept the connection, as well as additional text indicating the Server's name and domain.

Input Parameters

session_ptr	Pointer to SMTP Server session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid SMTP Session pointer.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service.

Allowed From

Threads

Example

```
   /* Reply to Client greeting. */
   status = nx_smtp_server_reply_to_greeting(session_ptr);

   /* If the session able to send a reply message to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response

nx_smtp_server_reply_to_ehlo

Reply to Client EHLO command

Prototype

```
UINT    nx_smtp_server_reply_to_ehlo(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client EHLO command, and replies with a Server reply code indicating if it accepts the command, as well as additional text indicating the Server's extension services, if any.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client EHLO command. */
status = nx_smtp_server_reply_to_ehlo (session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response

nx_smtp_server_reply_to_helo

Reply to Client HELO command

Prototype

```
UINT    nx_smtp_server_reply_to_helo(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client HELO command, and replies with a Server reply code indicating if it accepts the command. Note that the Server text does not list SMTP extension services, because HELO is only used in a Basic SMTP session (no extension services permitted).

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client HELO command. */
status = nx_smtp_server_reply_to_helo (session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response

nx_smtp_server_reply_to_auth

Reply to Client AUTH command

Prototype

```
UINT    nx_smtp_server_reply_to_auth(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client AUTH command, and replies with a Server reply code indicating if it accepts the command. If it does, the Server reply text contains the first of its encoded authentication challenges the Client must respond to.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client AUTH command. */
status = nx_smtp_server_reply_to_auth (session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response

nx_smtp_server_reply_to_mail

Reply to Client MAIL command

Prototype

```
UINT    nx_smtp_server_reply_to_mail(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client MAIL command, and replies with a Server reply code indicating if it accepts the command. It must be able to parse a valid sender mailbox address in the first command parameter before it will accept a Client MAIL command. If so it will create an SMTP Server Mail instance.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client MAIL command. */
status = nx_smtp_server_reply_to_mail (session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response, nx_smtp_server_mail_create

nx_smtp_server_reply_to_rcpt

Reply to Client RCPT command

Prototype

```
UINT    nx_smtp_server_reply_to_rcpt(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client RCPT command, and replies with a Server reply code indicating if it accepts the command. It must be able to parse a valid recipient mailbox address in the first command parameter before it will accept a Client RCPT command. If so it will create an SMTP Server Recipient instance.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client RCPT command. */
status = nx_smtp_server_reply_to_rcpt(session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response, nx_smtp_server_recipient_create

nx_smtp_server_reply_to_data

Reply to Client DATA command

Prototype

```
UINT    nx_smtp_server_reply_to_data(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client DATA command, and replies with a Server reply code indicating if it accepts the command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client DATA command. */
status = nx_smtp_server_reply_to_data(session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response

nx_smtp_server_reply_to_message

Reply to receiving Client mail message data

Prototype

```
UINT    nx_smtp_server_reply_to_message(
                             NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client message, and replies with a Server reply code indicating if it accepts the the mail message data. Note that the Server will wait to receive packets from the Client until it finds one which contains a valid end of message sequence in the packet buffer data. As it receives packets, it creates mail message segments for storing packet message data and chains these segments together to the Server mail instance.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client mail message data. */
status = nx_smtp_server_reply_to_message(session_ptr);

 /* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response, nx_smtp_server_bytepool_memory_get,

nx_smtp_server_bytepool_memory_release,
nx_smtp_server_blockpool_memory_get

nx_smtp_server_reply_to_quit

Reply to Client QUIT command

Prototype

```
UINT    nx_smtp_server_reply_to_quit(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client QUIT command, and replies with a Server reply code indicating if it acknowledges the Client is terminating the session.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client QUIT command. */
status = nx_smtp_server_reply_to_quit(session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_rset, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response

nx_smtp_server_reply_to_rset

Reply to Client RSET command

Prototype

```
UINT    nx_smtp_server_reply_to_rset(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client RSET command, and replies with a Server reply code indicating if it accepts the command. If so, it must delete the current mail transaction and reset the state of the SMTP protocol to the MAIL command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client RSET command. */
status = nx_smtp_server_reply_to_rset(session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_noop, nx_smtp_server_session_run, nx_smtp_utility_send_server_response, nx_smtp_server_mail_delete

nx_smtp_server_reply_to_noop

Reply to Client NOOP command

Prototype

```
UINT    nx_smtp_server_reply_to_noop(NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service receives the SMTP Client NOOP command, and replies with a Server reply code indicating if it acknowledging the command.

Input Parameters

session_ptr	Pointer to SMTP session instance

Return Values

NX_SUCCESS	(0x00)	Server reply successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Reply Client NOOP command. */
status = nx_smtp_server_reply_to_noop(session_ptr);

/* If session able to send reply to Client, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_session_run, nx_smtp_utility_send_server_response

nx_smtp_server_bytepool_memory_get

Allocate memory from Server byte pool

Prototype

```
UINT nx_smtp_server_bytepool_memory_get(VOID **memory_ptr,
                        ULONG memory_size, TX_BYTE_POOL *bytepool_ptr,
                        TX_MUTEX *mutex_ptr, UINT mutex_wait_option)
```

Description

This service obtains an exclusive lock on the previously created SMTP Server byte pool, then allocates the specified amount of memory from it.

Input Parameters

memory_ptr	Pointer to location where memory is allocated
memory_size	Amount of memory in bytes to allocate
bytepool_ptr	Pointer to Server byte pool
mutex_ptr	Pointer to Server byte pool mutex
mutex_wait_option	Timeout option on obtaining byte pool mutex

Return Values

NX_SUCCESS	(0x00)	Memory successfully allocated
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
* Allocate memory for a mail instance.  */
status = nx_smtp_server_bytepool_memory_get((VOID **)session_mail_ptr,
              sizeof(NX_SMTP_SERVER_MAIL),
              server_ptr -> bytepool_ptr, server_ptr -> bytepool_mutex_ptr,
              server _ptr -> bytepool_mutex_timeout);

/* If memory was successfully allocated, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_bytepool_memory_release,
nx_smtp_ server_blockpool_memory_get,
nx_smtp_ server_blockpool_memory_release

nx_smtp_server_bytepool_memory_release

Release memory back to Server byte pool

Prototype

```
UINT    nx_smtp_server_bytepool_memory_release(VOID *memory_ptr,
                                   TX_BYTE_POOL *bytepool_ptr,
                                   TX_MUTEX *mutex_ptr,
                                   UINT mutex_wait_option)
```

Description

This service obtains an exclusive lock on the previously created SMTP Server byte pool, then releases the memory allocated to the specified location back to the byte pool.

Input Parameters

memory_ptr	Pointer to location of allocated memory
bytepool_ptr	Pointer to Server byte pool
mutex_ptr	Pointer to Server byte pool mutex
mutex_wait_option	Timeout option on obtaining byte pool mutex

Return Values

NX_SUCCESS	(0x00)	Memory successfully released
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
* Allocate memory for a mail instance.  */
status = nx_smtp_server_bytepool_memory_release(mail_ptr,
                  server_ptr -> bytepool_ptr, server_ptr -> bytepool_mutex_ptr,
                  server_ptr -> bytepool_mutex_timeout);

/* If memory was successfully released, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_bytepool_memory_get,
nx_smtp_ server_blockpool_memory_get,
nx_smtp_ server_blockpool_memory_release

nx_smtp_server_blockpool_memory_get

Allocated memory from Server block pool

Prototype

```
UINT nx_smtp_server_blockpool_memory_get(VOID **memory_ptr,
                              TX_BLOCK_POOL *blockpool_ptr,
                              TX_MUTEX *mutex_ptr,
                              UINT mutex_wait_option)
```

Description

This service obtains an exclusive lock on the previously created SMTP Server block pool, then allocates a block of memory from the block pool.

Input Parameters

memory_ptr	Pointer to location where memory is allocated
blockpool_ptr	Pointer to Server block pool
mutex_ptr	Pointer to Server block pool mutex
mutex_wait_option	Timeout option on obtaining block pool mutex

Return Values

NX_SUCCESS	(0x00)	Memory successfully allocated
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Allocate a block of memory fromServerblock pool for segment data. */
status = nx_smtp_server_blockpool_memory_get(
                      (VOID **)&message_segment_ptr -> message_ptr,
                      server _ptr -> blockpool_ptr,
                      server _ptr -> blockpool_mutex_ptr,
                      server _ptr -> blockpool_mutex_timeout);

/* If memory was successfully allocated, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_bytepool_memory_get,
nx_smtp_ server_ bytepool_memory_release,
nx_smtp_ server_blockpool_memory_release

nx_smtp_server_blockpool_memory_release

Release memory back to Server block pool

Prototype

```
UINT    nx_smtp_server_blockpool_memory_release(VOID *memory_ptr,
                             TX_BLOCK_POOL *blockpool_ptr,
                             TX_MUTEX *mutex_ptr,
                             UINT mutex_wait_option)
```

Description

This service obtains an exclusive lock on the previously created SMTP Server block pool, then releases the memory allocated to the specified location back to the Server block pool.

Input Parameters

memory_ptr	Pointer to location of allocated memory
blockpool_ptr	Pointer to Server block pool
mutex_ptr	Pointer to Server block pool mutex
mutex_wait_option	Timeout option on obtaining block pool mutex

Return Values

NX_SUCCESS	(0x00)	Memory successfully released
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Release memory for a mail instance.  */
status = nx_smtp_server_blockpool_memory_release(mail_ptr,
                              server _ptr -> blockpool_ptr,
                              server_ptr -> blockpool_mutex_ptr,
                              server _ptr -> blockpool_mutex_timeout);

/* If memory was successfully released, status = NX_SUCCESS. */
```

See Also

nx_smtp_bytepool_memory_get, nx_smtp_bytepool_memory_release, nx_smtp_blockpool_memory_get

nx_smtp_utility_send_server_response

Send Server reply to SMTP Client

Prototype

```
UINT nx_smtp_utility_send_server_response(
                        NX_SMTP_SERVER_SESSION *session_ptr,
                        CHAR *reply_code, CHAR *info, CHAR *more_info,
                        UINT more_lines_to_follow)
```

Description

This service sends a Server reply to the SMTP Client. The reply should include a server code, and at least one text string to make up the message text, with an extra argument (‘more_info’) for additional text. The caller must specify when the Server reply contains multiple lines of text.

Input Parameters

session_ptr	Pointer to Server session replying to Client
reply_code	SMTP Server reply code
info	Pointer to Server text to add to message
more_info	Pointer to optional additional text to add
more_lines_to_follow	Server reply contains one line or multiple lines

Return Values

NX_SUCCESS	(0x00)	Message successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter.
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Send a sinlge line of server reply "250 serverdomain.com\r\n" */

status = nx_smtp_utility_send_server_response(session_ptr,
                              NX_SMTP_TEXT_CODE_OK_TO_CONTINUE, NX_SMTP_SERVER_OK,
                              NX_SMTP_SERVER_DOMAIN, NX_FALSE);


/* If a valid message was created and successfully sent, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_session_run

nx_smtp_utility_parse_client_request

Parses the SMTP command from Client message

Prototype

```
UINT nx_smtp_utility_parse_client_request (CHAR *buffer_ptr,
                              UINT *protocol_code, UINT length)
```

Description

This service parses the Client messages received during an SMTP session, extracts the SMTP command, and returns an enumerated command data back to the Server session engine.

Input Parameters

buffer_ptr	Pointer to Client message
protocol_code	Enumerated Client command data
length	Size in bytes of buffer

Return Values

NX_SUCCESS	(0x00)	Message successfully sent
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_SMTP_PARAM_ERROR	(0xB2)	Invalid message buffer parameter

Allowed From

Application code

Example

```
/* Parse client command and parameters from packet data. */

Status = nx_smtp_utility_parse_client_request(buffer_ptr, &protocol_code,
                              packet_ptr -> nx_packet_length);
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_session_run

nx_smtp_utility_parse_mailbox_address

Parses the mailbox address from Client command

Prototype

```
UINT nx_smtp_utility_parse_mailbox_address(HAR *start_buffer,
                                    CHAR *end_buffer,
                                    CHAR **mailbox_address,
                                    UINT addr_is_parameter)
```

Description

This service parses the mailbox address from a Client command received during an SMTP session, verifies it is a valid mailbox address, and returns a pointer to the extracted parameter.

Input Parameters

`start_buffer`	Pointer to start of buffer to parse
`end_buffer`	Pointer to end of buffer to parse
mailbox_address	Pointer to location where parsed mailbox
addr_is_parameter	Mailbox address is part of a command

Return Values

NX_SUCCESS	(0x00)	Message successfully sent
NX_PTR_ERROR	(0x16)	Message successfully sent
NX_SMTP_PARAM_ERROR	(0xB2)	Invalid mailbox address parsed

Allowed From

Application code

Example

```
/* Parse sender's mailbox address. */
Status = nx_smtp_utility_parse_mailbox_address(buffer, CRLF_location,
                                              &mailbox_address, NX_TRUE);

/* If a valid mailbox is found, mailbox_address points to a non NULL location
   and status = NX_SUCCESS. */
```

See Also

`nx_smtp_server_reply_to_mail,`
`nx_smtp_server_reply_to_rcpt, nx_smtp_server_session_run`

nx_smtp_utility_login_authenticate

Authenticates the SMTP Client

Prototype

```
UINT nx_smtp_utility_login_authenticate(
                                 NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service sends encoded authentication challenges to the SMTP Client in response to the AUTH command. It then decodes the Client responses, and forwards the Client data to the Server authentication check. It sends as many challenges as is required by the Server to mark the authentication process complete or until Client data is rejected.

Input Parameters

session_ptr	Pointer to Server session authenticating the Client

Return Values

NX_SUCCESS	(0x00)	Server reply successfully extracted
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Authenticate the Client. */
status = nx_smtp_utility_login_authenticate(session_ptr);

/* If Client data was received and decoded regardless if Client was successfully
    authenticated, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_auth, nx_smtp_server_session_run, nx_smtp_utility_send_server_response

nx_smtp_utility_spool_session_mail

Spool SMTP mail received from Client to hard disk

Prototype

```
UINT  nx_smtp_utility_spool_session_mail (
                              NX_SMTP_SERVER_MAIL *mail_ptr)
```

Description

This service calls the Server mail spooling service if one was supplied in the SMTP Server create call. The NetX SMTP Server mail spooler service marks SMTP mail as ready for deletion presuming it has been written to hard disk as it is only a stub for an actual mail spooler service.

Input Parameters

mail_ptr	Pointer to Server mail to spool

Return Values

NX_SUCCESS	(0x00)	Mail was successfully spooled
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Spool Server mail.  */
status =  nx_smtp_utility_spool_session_mail(session_ptr -> current_mail_ptr);

/* If the mail was successfully spooled, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_reply_to_greeting, nx_smtp_server_reply_to_ehlo, nx_smtp_server_reply_to_helo, nx_smtp_server_reply_to_auth, nx_smtp_server_reply_to_mail, nx_smtp_server_reply_to_rcpt, nx_smtp_server_reply_to_data, nx_smtp_server_reply_to_message, nx_smtp_server_reply_to_quit, nx_smtp_server_reply_to_rset, nx_smtp_server_session_run, nx_smtp_server_clear_session_mail

nx_smtp_utility_clear_session_mail

Delete all Server mail from the current session

Prototype

```
UINT nx_smtp_utility_clear_session_mail(
                              NX_SMTP_SERVER_SESSION *session_ptr)
```

Description

This service deletes all Server mail received in the current session and releases memory back to the Server memory pools. Typically this call is made after the Server spools the mail to hard disk.

Input Parameters

session_ptr	Pointer to Server session clearing mail

Return Values

NX_SUCCESS	(0x00)	Server mail deleted successfully
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Application code

Example

```
/* Clear Server mail from the specified session. */
status = nx_smtp_utility_clear_session_mail(session_ptr);

/* If the mail was cleared, status = NX_SUCCESS. */
```

See Also

nx_smtp_server_mail_delele, nx_smtp_server_recipient_delete, nx_smtp_server_bytepool_memory_release, nx_smtp_server_blockpool_memory_release

nx_smtp_utility_print_server_mail_status

Displays the contents and details of an SMTP Server mail

Prototype

```
UINT nx_smtp_utility_print_server_mail_status(
                                    NX_SMTP_SERVER_MAIL *mail_ptr)
```

Description

This service prints the contents of specified mail message including mail envelope, and miscellaneous mail transaction data. While it is not required, the application should obtain the Server print mutex if calling from a thread task to keep the display of session mail data discreet among concurrent Server session tasks.

Input Parameters

mail_ptr	Pointer to Server mail to display

Return Values

NX_SUCCESS	(0x00)	Server mail data displayed successfully
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
/* Print mail item's data. */
Status = nx_smtp_utility_print_server_mail_status(
                                    session_ptr -> current_mail_ptr);

/* if server mail item successfully displayed, status = NX_SUCCESS. */
```

See Also

nx_smtp_utility_print_server_reserves

nx_smtp_utility_print_server_reserves

Display SMTP Server memory and packet pool reserves

Prototype

```
UINT nx_smtp_utility_print_server_reserves(
                                    NX_SMTP_SERVER *server_ptr)
```

Description

This service displays the remaining bytes and blocks available in the Server byte and block pools and remaining packets in the Server packet pool. While it is not required, the application should obtain the Server print mutex if calling from a thread task to keep the display of Server reserves discreet among concurrent tasks.

Input Parameters

server_ptr	Pointer to the SMTP Server

Return Values

NX_SUCCESS	(0x00)	Server data displayed successfully
NX_PTR_ERROR	(0x16)	Invalid pointer parameter
NX_CALLER_ERROR	(0x11)	Invalid caller of this service

Allowed From

Threads

Example

```
       /* Display the status of Server packet and memory pool reserves.  */
        Status = nx_smtp_utility_print_client_reserves(session_ptr -> client_ptr);

        /* if server reserves successfully accessed and displayed, status = NX_SUCCESS.
*/
```

See Also

nx_smtp_utility_print_server_mail_status

nx_smtp_utility_packet_data_extract

Extract data from a chain of packets into a single buffer

Prototype

```
UINT nx_smtp_utility_packet_data_extract(NX_PACKET *packet_ptr,
                              VOID *buffer_start, ULONG buffer_length,
                              ULONG *bytes_copied)
```

Description

This service extracts data from a chain of IP packet data into a single buffer. The number of bytes that should be copied is derived from the packet pointer header; if the number of bytes actually copied to buffer is less than that, the service returns an invalid packet status.

Input Parameters

packet_ptr	Pointer to head of the packet chain
buffer_start	Pointer to start of buffer to copy data to
buffer_length	Size of buffer to copy data to
bytes_copied	Pointer to actual amount of data copied

Return Values

NX_SUCCESS	(0x00)	Server data displayed successfully
NX_INVALID_PACKET	(0x12)	Missing packet data; invalid packet
NX_PTR_ERROR	(0x16)	Invalid pointer parameter

Allowed From

Threads

Example

```
/* Extract chained packet data into mail message segment. */
status = nx_smtp_utility_packet_data_extract(packet_ptr,
                    mail_message_segment  -> message_ptr,
                    server_ptr -> blockpool_ptr -> tx_block_pool_block_size,
                    &bytes_copied);

/* If all packet data copied to the supplied buffer, status = NX_SUCCESS. */
```

See Also

nx_smtp_reply_to_message

Appendix A. SMTP Server Reply Codes and Client Commands

Below is a list of common SMTP Server reply codes, listed first in numerical order, then grouped by Client command.

211	System status or system help reply
214	Help message
220	<domain> Service ready
221	<domain> Service closing transmission channel
235	Client authentication successful
250	Requested mail action okay, completed
251	User not local; will forward to <forward-path>
252	Unable to verify members of mailing list
334	Server authentication challenge
354	Start mail input; end with <CRLF>.<CRLF>
421	<domain> Service not available, closing transmission channel
450	Requested mail action not taken: mailbox unavailable
451	Requested action aborted: local error in processing
452	Requested action not taken: insufficient system storage
500	Syntax error, command unrecognized
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
521	<domain> does not accept mail (see RFC 1846)
530	Authentication required
534	Authentication mechanism is too weak
535	Authentication failed
538	Encryption required for requested authentication mechanism
550	Requested action not taken: mailbox unavailable
551	User not local; please try <forward-path>
552	Requested mail action aborted: exceeded storage allocation
553	Requested action not taken: mailbox name not allowed
554	Transaction failed

Reply Codes Grouped By Command

Connecting:

220	<domain> Service ready
421	<domain> Service not available, closing transmission channel

HELO

250	Requested mail action okay, completed

	421	<domain> Service not available, closing transmission channel
	500	Syntax error, command unrecognized
	501	Syntax error in parameters or arguments
	504	Command parameter not implemented
	521	<domain> does not accept mail [RFC 1846]
EHLO		
	250	Requested mail action okay, completed
	421	<domain> Service not available, closing transmission channel
	500	Syntax error, command unrecognized
	501	Syntax error in parameters or arguments
	504	Command parameter not implemented
	550	Not implemented
AUTH		
	235	Client authentication successful
	334	Server authentication challenge
	530	Authentication required
	534	Authentication mechanism is too weak
	535	Authentication failed
MAIL		
	250	Requested mail action okay, completed
	421	<domain> Service not available, closing transmission channel
	451	Requested action aborted: local error in processing
	452	Requested action not taken: insufficient system storage
	500	Syntax error, command unrecognized
	501	Syntax error in parameters or arguments
	552	Requested mail action aborted: exceeded storage allocation
RCPT		
	250	Requested mail action okay, completed
	251	User not local; will forward to <forward-path>
	553	Requested action not taken: mailbox name not allowed
	421	<domain> Service not available, closing transmission channel
	450	Requested mail action not taken: mailbox unavailable
	451	Requested action aborted: local error in processing
	452	Requested action not taken: insufficient system storage
	500	Syntax error, command unrecognized
	501	Syntax error in parameters or arguments
	503	Bad sequence of commands
	521	<domain> does not accept mail [rfc1846]
	550	Requested action not taken: mailbox unavailable
	551	User not local; please try <forward-path>
	552	Requested mail action aborted: exceeded storage allocation
DATA		

354	Start mail input; end with <CRLF>.<CRLF>
421	<domain> Service not available, closing transmission channel
451	Requested action aborted: local error in processing
554	Transaction failed
500	Syntax error, command unrecognized
501	Syntax error in parameters or arguments
503	Bad sequence of commands

Message (Received Mail Data)

250	Requested mail action okay, completed
451	Requested action aborted: local error in processing
452	Requested action not taken: insufficient system storage
552	Requested mail action aborted: exceeded storage allocation
554	Transaction failed

RSET

250	Requested mail action okay, completed
500	Syntax error, command unrecognized
501	Syntax error in parameters or arguments
504	Command parameter not implemented
421	<domain> Service not available, closing transmission channel

NOOP

250	Requested mail action okay, completed
421	<domain> Service not available, closing transmission channel
500	Syntax error, command unrecognized

QUIT

221	<domain> Service closing transmission channel
500	Syntax error, command unrecognized

Client Commands

Command	Format
Greeting	None - This is the connection to TCP Server socket
EHLO	EHLO<SP>Domain<CRLF>
HELO	HELO<SP>Domain<CRLF>
MAIL	MAIL FROM:<reverse-path><CRLF>
RCPT	RCPT TO:<forward-path><CRLF>
RCPT	RCPT CC:<forward-path><CRLF>
RCPT	RCPT BCC:<forward-path>><CRLF>
AUTH	AUTH<SP><authentication type(s)><CRLF>
DATA	DATA<CRLF>

```
QUIT          QUIT<CRLF>
RSET          RSET<CRLF>
NOOP          NOOP<CRLF>

<SP>          Single space
<CRLF>        Carriage Return Line Feed
[ ]           Optional text
```

There are other formats for the Client to request and send authentication data, including appending authentication data onto the MAIL command, or including username and password data with the AUTH command itself. However they are not supported in the current NetX SMTP API.

Appendix B. Definition of Common Mail Transport Terms

SMTP Client - The Client is the agent that is responsible for transmitting mail to an SMTP Server. It is not necessarily the original host sender, nor is it the mail author. The Client is sometimes referred to as a "sender."

SMTP Client session – The Client session is responsible for connecting with an SMTP Server, maintaining the state of the SMTP session, and sending the commands necessary to complete a mail transaction.

SMTP Server - The Server is the agent that is responsible for receiving mail from an SMTP Client. It is not necessarily the final destination, nor does it necessarily deliver mail immediately to the recipient(s). The Server is sometimes referred to as a "receiver."

Mail user agent (MUA) – An MUA is not directly involved with sending or delivering mail. It is a mail browser that enables the user to compose mail and view mail sent to them.

Mail transfer agent (MTA) – An MTA can be either a Server or Client or relay gate. Its role is to deliver mail either by sending mail to another MTA or by receiving mail from another MTA.

Mail Server – A Mail Server is responsible both for receiving and delivering mail. Mail Server architecture typically contains a Client agent, a Server agent, and a mail queuing service all working together to efficiently provide mail service to clients. It must free up resources such as dynamic memory to avoid bottlenecks in mail service. Sendmail and Postfix are good examples of well known Mail Server architecture.

State machine – Both SMTP Client and Server maintain a state of the current mail transaction. State data includes mail recipients, mail message data, and user authentication. The state machine is the logical progression of mail session from requesting mail service, to supplying recipient list, and mail message data, and finally quitting the service on completion. The SMTP protocol requires the Server to provide a "reset" service so that a Client at any time can restart a mail transaction. All previous mail data is erased, and the mail transaction starts at the beginning with the MAIL FROM command.

Basic vs extended SMTP services – The basic SMTP protocol provides enough services to complete a mail transmission to a limited number of recipients and a limited size of the message. The mail transmission can include multipart MIME mail messages. Extended services include the SIZE command, authentication, EXPN and other services. See RFC 2821 and 2554 for more details. An SMTP Client indicates it would like extended services by using the EHLO command instead of the HELO (legacy) command. Similarly, the Server that does not support extended SMTP services will reject the EHLO command and the SMTP Client is obligated to send the HELO command and use only basic services.

Envelope - An Internet message consists of an envelope component and a mail body component. The mail browser never displays the envelope component. It is the data that is transmitted between Server and Client agents to facilitate mail delivery.

Header – The mail message consists of a header and message body, separated by a blank line. The header is a series of field:value pairs that specify who the mail is to and from, when it was sent by the original author, and the subject. It can contain additional fields such as MIME content for multi part MIME data.

Message body – The message body consists of a series of lines of message data, and ends with an 'end of message' sequence. Each line must end with a line terminating sequence. The RFC 2821 mandates that each line not exceed 1000 bytes, and *recommends* that lines of text not exceed 78 bytes. The entire message must not exceed 64k bytes, unless the sender and receiver can negotiate extended services to handle large messages.

Appendix C. SMTP Message Format and Syntax Rules

1. The mail message body of a mail item must be terminated by an 'end of message sequence' (EOM) sequence:

 <CR><LF>.<CR><LF>

(or 0x0D 0xOA 0x2e 0x0D 0x0A. Hexidecimal notation)

2. A data transparency procedure prevents the EOM sequence from interfering with the actual text. When sending a line of text data, the NetX SMTP Client checks the first character of the line. If it is a period (0x2E), one additional period is inserted at the beginning of the line. This is known as 'byte stuffing.'

3. Client commands and Server replies are limited to 512 characters and plain ASCII text characters. In practice, the 512 limit is rarely approached.

4. A Client command should be (not required) in upper case.

5. There must be a single space between the Client command and the succeeding parameter(s).

RFC 2821 has a complete list of Client commands for basic SMTP services. Numerous other RFC documents cover the growing number of extended SMTP services in extensive detail. RFC 2821 has a complete list of commands. See RFC 2821 for more details on message formatting

6. A Server reply must begin with a known SMTP 3 digit reply code followed by either a dash or single space and optional Server text.

An example multi-line reply follows below:

```
First packet sent to Client:      250-SMTP-Server1
Second packet sent to Client:     250-AUTH LOGIN 250 VRFY
```

Note that the '250' in the last line has no dash, indicating no more lines to follow. RFC 2821 has a complete list of server reply codes.

7. Each line in a multiline Server reply must begin with the same reply code.

8. All Client commands and Server replies must be terminated by a line terminating carriage return-line feed sequence (CRLF or *0x0D 0x0A* in Hexidecimal notation). Each line in a multiline reply must end with this line terminator sequence.

9. Domain names in mailbox addresses in Client command parameters are not case sensitive, but the localhost case must be preserved.