ESMTP Client Technical Reference Manual

Interniche Legacy Document

Version 1.00

Date: 09-May-2017 10:43

Table of Contents

1 Introduction

The ESMTP module uses the SMTP and (optionally) SSL protocols to send email from a NicheStack client application to standard email servers on the Internet such as those provided by Yahoo and Google. It could connect to a localized private email server, as long as that server uses the SMTP protocol and follows standard email conventions.

The porting engineer must provide an email application that collects the information required for an email (from, to, data, etc.) and passes that information to ESMTP via calls to the various APIs provided by InterNiche's ESMTP.

A typical use of ESMTP would be to send periodic reports and status messages from various applications. However, depending on the needs of your application, the ESMTP APIs could be used to send complex emails with multiple senders and recipients and multiple attachments of various types.

InterNiche's ESMTP only transmits email, it does not provide incoming email.

1.1 ESMTP Capabilities

ESMTP supports the following email header types:

- From
- ReplyTo
- To
- CC
- BCC
- Subject

There can be multiple entries for: From, TO, CC, and BCC.

ESMTP will accept **body** data from one of five different sources:

- a buffer;
- a file;
- a function that generates data;
- a CLI command;
- a script file that executes CLI commands.

As per the email specifications, all **body** data must be ASCII text.

ESMTP will accept multiple **attachments** from one of three different sources:

- a buffer;
- a file;
- a function that generates data.

It provides simple APIs for attachments containing standard ASCII data. It can send attachments in a very large variety of formats, as long as the user application provides the necessary stings for MIME-type, MIME-subtype, and MIME parameters.

1.2 ESMTP Protocol

ESMTP is capable of providing full TLS/SSL security and encryption for the entire email session. It also supports PLAIN authentication (username and password) as defined in RFC 4616 and RFC 4954

Supported RFCs

The ESMTP module provide support for the following RFCs:

Current base RFCs for SMTP

- RFC 5321 Simple Mail Transfer Protocol
- RFC 5322 Internet Message Format

Supporting RFCs for SMTP

- RFC 1035 Domain Names – Implementation and Specification
- RFC 2045 Multipurpose Internet Mail Extensions (MIME) Part one: Format of Internet Message Bodies
- RFC 2046 Multipurpose Internet Mail Extensions (MIME) Part two: Media Types
- RFC 2920 SMTP Service Extension for Command Pipelining
- RFC 3207 SMTP Service Extension for Secure SMTP over Transport Layer Secuirty
- RFC 4422 Simple Authentication and Security Layer (SASL)
- RFC 4616 The PLAIN Simple Authentication and Security Layer (SASL) Mechanism
- RFC 4954 SMTP Service Extension for Authentication

Note: Numerous older RFC have been superseded or made obsolete by those listed here: RFCs 821, 822, 851, 974, 1869, 2197, 2222, 2487, 2821, 2822 and others.

1.3 Terminology

MIME type

A MIME type is a standardized label for describing the type of payload data and allowing for its encoding and handling. It consists of a type and sub-type, formatted as "type/subtype". The list of media types is maintained by the IANA. For more information, please refer to RFC2045 and RFC6838.

Base64

Because the SMTP protocol uses certain characters within the data stream as control characters, it cannot be used to transfer arbitrary binary data without modification. Even some ASCII characters have special meaning within SMTP. Base64 encoding is a method for encoding a stream of arbitrary data (possibly binary) into a set of characters that cannot contain any control characters. This is not encryption as anyone who knows the algoritm can easily decode the data. Data that has been base64 encoded is about 1/3rd larger than the original stream.

2 The Path of an email

A user application uses ESMTP APIs to pass email addresses and data. The ESMTP module connects to a remote email server and uses the SMTP protocol to pass the email headers, body, and attachments to the email server. The email server forwards the email to one or more destination computers. The destination computer(s) will have an email program that accepts the email and presents it to the recipients.

3 General Structure of an Email session

Each part will be discussed in more detail below.

1. Application calls esmtp_startsession().
2. If this is first call to esmtp_startsession, ESMTP will allocate memory pools based on user settable variables: maxconcursess, esm_membufsize, esm_txbufsize.
3. ESMTP uses DNS to get the IP address of the server.
4. Application makes multiple calls to `esmtp_param()` passing the parameters that will be used in the header of the message. All parameters are saved in a session-specific memory buffer.
5. Application calls one of ESMTP's APIs to pass or specify the source of data for the email body.
6. The application may optionally call any of ESMTP's attachment APIs to pass attachment parameters and data.
7. Application calls esmtp_exec() to indicate that it has passed all of the parameters and ESMTP should now send the email.
8. ESMTP will establish a connection to the email server and through a series of SMTP requests and responses, it will pass all of the information to the server. At the time when it sends the data for the body or an attachment:
    a. If the data comes from a file it will open, read, and transmit the file data.
    b. If the data comes from an application callback function, it will make one or more calls to that function, reading and transmitting the data, until the function returns a 0 indicating that it has passed all of the data.
9. ESMTP sends a Data Done message to the email server and reads the response which reports whether or not the email server has accepted responsibility for delivering the email.
10. ESMTP calls the application's main callback function passing this final status.
11. ESMTP closes the session with the email server and makes the space in the memory pools used by this session available for a new session.

Generally, any error will cause the email session to close. However, if an `esmtp_param()` call returns an error, the application can either close the session or make another API call.

An email session starts with the applications call to esmtp_startsession(). It lasts until ESMTP calls the application's callback function to give the final status of the email. Once ESMTP establishes a connection to the email server, the session consists of a series of requests from ESMTP and responses from the email server. The email server may be busy causing a substantial delay between any request and response. Today, an email session typically lasts only a few seconds. However, the length is unpredictable. It may last a minute or more.

Note: The required minimum timeout values specified by RFC 5321 imply that a session could last much longer than a few minutes.

4 ESMTP Sources, Options

4.1 ESMTP Source Files List

esmtp.h	The internal header file for the ESMTP module.
esmtp_port.h	Provides compile-time configuration parameters, the API prototypes and defines needed by the user.
esmtp.c	Contains the main loop, state machine, and the base routines needed by the state machine. Most functions called by these routines are located in esmtp_utils.c or esmtpfull.c
esmtp_keycert.c	Code and data structures related to the SSL client for ESMTP. Includes X.509 certificates for various Certicat Authorities.
esmtp_mod.c	Code that defines, initializes and starts the ESMTP module.
esmtp_nt.c	Code for ESMTP netstat and config CLI commands
esmtp_utils.c	Supporting routines called by esmtpapi.c and esmtp.c.
esmtpapi.c	Contains code for the API calls.
esmtpfull.c	Code for handling email attachments and for starting an SSL client session
makefile.in	Makefile for esmtp directory

4.2 ESMTP Build Configuration Options

These defines in `ipport.h` determine how much of the ESMTP code will be included in the build

`USE_ESMTP`	Include EMSTP module
`ESMTP_MENUS`	Include ESMTP CLI commands (netstat and config)
`ESMTP_AUTH`	Include code to for SMTP authorization via username and password.
`ESMTP_FULL`	Include code to create email attachments.
`ESMTP_SECURITY`	Include code to handle SSL connection to the email server
`ESMTP_DEBUG`	Include code to print ESMTP debug messages to the console
`ESMTP_DEBUG_VERBOSE`	Include code to print text of SMTP messages sent and received by ESMTP
`DUMPEMAILBUF`	Include code to dump the contents of the session's memory buffer

Note: DNS_CLIENT and DNSC_GETADDRINFO must be defined with ESMTP.

5 Memory Pools

ESMTP uses memory pools that are preallocated at initialization time. This means that the amount of memory used by the ESMPT module is determined at initialization time by the configuration parameters. ESMTP, itself, does not make any other calls to allocate additional memory. However runtime memory limitations are still possible when SSL is used, because it allocates memory dynamically.

ESMTP uses three memory pools:

1. ESMTP memory buffer pool. Holds all parameters for an email session.
2. Transmit buffer pool. Used to store an ESMTP message, or part of a message, while it is being read from the application and processed (base64 or encryption) before it is transmitted to the email server.
3. Connection structure pool. Structure used internally by ESMTP to keep track of email session.

The size allocated for each pool is the per-session size of the element times the maximum number of sessions. The variables that control these sizes are shown in the section below.

6 Configurable Parameters

The configuration variables in the table below have compile time default values defined near the top of `esmtp_port.h`. They also can be changed at dynamically, normally at init time, but also at any time when no email sessions are active.

variable	name define name	description
maxconcursess	DFT_ESM_CONCURSESS	Maximum allowable number of concurrent sessions
esm_membufsize	DFT_ESM_MEMBUFSIZE	Size of per-session memory buffer used to store all parameters required for an email session
esm_txbufsize	DFT_ESM_TXBUFSIZE	Size in bytes of per-session transmit buffer.
esm_maxbufdata	ESM_MAXBUFDATA	Maximum size of any data field in per-session memory buffer.
esm_loc_domainname	ESM_LOC_DOMAINNAME	local domain name in the form “xxx.com”. This is required by the email server
esm_conntmo	ESM_CONNTMO	Time in seconds before ESMTP will give up trying to connect to the specified email server.
esmtp_idletmo	ESM_IDLETMO	Idle time out. The timeout period starts again at zero (`0`) whenever the application makes an API call, when ESMTP sends a request to the server reads a response.

The configuration parameters below can only be set at compile time by changing the default values in esmtp_port.h:

ESM_RXBUFSIZE	Maximum buffer size for messages from the email server. Most of these are small, or less than 200 bytes. Some may be 700 bytes or more, but the important data will be early in the buffer and the rest will be truncated by ESMTP.
ESMTP_MAXDOMAIN	Maximum length of a domain name used in esmtp_param API calls. ESMTP will send an error if the call contains a larger domain name. RFC 5231 allows up to 254 bytes.
ESMTP_MAXPORTLEN	Maximum bytes in the string containing a port number
DFT_VERSION	Default IP version that will be used to connect to the email server. If IP_V6 is defined and the servers DNS records only contain IPv6 addresses, then the connection will use IPv6. Note: This is only meaningful when both IPv4 and IPv6 are present in the build.
ESM_MINFORMATSPACE	Size of buffer used to format a single line of text in the email body. (See "Formatting of text in email body", below)
ESM_MAXWORDSEARCH	Maximum number of bytes to search before splitting a word that crosses the end-of-line boundary. (See "Formatting of text in email body", below)
ESMTP_SRVRDYTMO	Maximum time in seconds to wait for a “server ready” message after a connection has been made to the server
ESMTP_SIMPCMDTMO	Maximum time in seconds to wait for a response to an ESMTP request
ESMTP_DATADONETMO	Maximum time in seconds to wait for a “data done” response after passing a complete email to the server. If this timeout occurs, it must be assumed that the email was not sent.
ESMTP_QUITTMO	Maximum time in seconds to wait for the server to send a 221 response after ESMTP sends a QUIT command to the server. ESMTP will not pass the final status to the application and free up the session memory until it receives the 221 response or this timeout occurs. This timeout does not affect the success status, `ESM_TYPE_MAILDONE`, of the email.
ESMTP_SSLCLOSEDELAY	Time in ticks to allow for the graceful shutdown of the SSL connection before ESMTP closes the socket and deletes all of the related structures

7 Formatting of text in email body

ESMTP does some automatic reformatting of text for the body of an email so that it can be transmitted properly and displayed properly by the recipient's email display program. The SMTP specifications place a few requirements for the formatting of each line of text in an email body. Unix uses only a single linefeed character (ASCII LF) to indicate the end of a line. However, the SNMP specification requires two characters, carriage return and line feed (CRLF) , to indicate the end of any line. Any bare 'CR' or 'LF' characters found in the text must be changed to CRLF.

ESMTP requires that If any line starts with a period, another period must be added so that the line starts with two periods. The display program will strip of one of the periods.

While ESMTP automatically handles this reformatting, it should be understood that reformatting may require that the buffer used to hold the data during and after formatting be substantially larger than the original data. For example, if ESMTP were given a block of text that contained the series of eight bytes:

```
.LF.LF.LF.LF
```

then ESMTP would need to expand this to 16 bytes:

```
..CRLF..CRLF..CRLF..CRLF
```

The define `ESM_MINFORMATSPACE` limits the size of the buffer that will be used for formatting. To be safe, the default is twice as large as an 80 byte line of data. However, if it is known that data passed to ESMTP already uses CRLFs for line endings, then this buffer can be smaller.

A single line of text in the body should not be longer than about 80 bytes. If a line reaches 78 bytes in length without a CRLF, then ESMTP will look for a space character indicating the end of a word. When it finds the space, it will add a CRLF, assuming it has not already found one. `ESM_MAXWORDSEARCH` limits how far ESMTP will search for the end of a word following 78 bytes. If it reaches 78 bytes plus `ESM_MAXWORDSEARCH`, it will split the line at that point.

8 SSL

The traditional port for unencrypted email sessions is port 25. Today, very few email servers connected to the Internet will accept unencrypted email. Nearly all require all email sessions to use the Transport Layer Security (TLS) protocol, which is more commonly called SSL, the acronym of its predecessor. We use the term SSL throughout this document.

Email servers follow two different patterns in their use of SSL based on which port number is used for the connection.

When port 587 is used for the connection, the email session follows the pattern described in RFC 5321. The session begins in the clear. The client (ESMTP) will send an EHLO (Extended Hello) request, and the email server will respond with the services that it supports. “STARTTLS” will be listed as one of these services. ESMTP will begin SSL negotiations and once the connection is secure, it will resend the EHLO and get a new list of the services supported by the email server for secure connections. The remainder of the email session will be encrypted.

When port 465 is used, the server expects SSL negotiation to begin as soon as the TCP-level connection is made. The entire email session will be encrypted from the email server’s first “service ready” message to its final response to ESMTP’s QUIT request. This method is somewhat more efficient, because there is only one EHLO request and response.

Use of SSL is entirely transparent to the email application, except for the selection of which port number to use and the possible verification of X.509 certificates (described in the next section),.

8.1 SSL Configuration

The `esmtp/esmtp_keycert.c` contains code and data structures related to the SSL client component of the ESMTP module. The data structures include the top-level X.509 certificates that are currently being used to sign the lower-level X.509 certificates presented by the Google and Yahoo mail servers to ESMTP (during the SSL/TLS handshake) as a mechanism to identify the sender. Different email servers may use different Certificate Authorities to provide the top-level signing certificates.

If ESMTP is unable to validate the email server using the X.509 certificates currently in `esmtp_keycert.c`, it will report an error. The user will need to replace the certificates in `esmtp_keycert.c` with ones that are appropriate for their usage scenario. The certificates presented by the server can be seen in a Wireshark trace between "Server Hello" and the "Server Hello Done". Your system administrator should be able to provide you with the required certificates.

For more details see the sections on certificates in the SSL reference documentation included with your delivery. For more information on the SSL client used by ESMTP, see the section "Using the SSL 'Client Shim'" in the *CryptoEngine and Crypt API Technical Reference* manual.

9 Porting Engineer-provided functions

9.1 Email Application

The porting engineer must provide an email application to drive the email process. It starts by calling `esmtp_startsession()`, uses ESMTP's API calls to pass email parameters and data and then waits for a finial status as to whether or not the email server accepted responsibility for delivering the email.

9.2 Simple example of an ESMTP application

The APIs and callback functions used in this example will be described in the sections below.

```
extern void (*mystatus_cb)(int, int, int, int);
extern void (*mybody_cb)(int, char *, int);
int ssid = 0;              /* SSID for this email session  */

/* Main user application for sending email */
my_esmtp_app()
{
   int rc;

   ssid = esmtp_startsession("gmail.com", "587", ESMCF_USESSL | ESMCF_USEAUTH,
                             "login", "password", mystatus_cb);
   rc = esmtp_param(ssid, ESMTP_FROM, "me@example.com");
   rc = esmtp_param(ssid, ESMTP_TO, "you@example.com");
   rc = esmtp_param(ssid, ESMTP_SUBJECT, "This is the email subject");
   rc = esmtp_bodyfunctext(ssid, &mybody_cb);     /* Function to create text for body */
   rc = esmtp_exec(ssid);                         /* Done with parameters. Send email */
}


/* Callback function for obtaining the final status of the email session */
void
mystatus_cb(int ssid, int type, int code, void *data)
{
   switch (type)
   {
   case ESM_TYPE_FATAL:
     dprintf("CALLBACK: ssid=%d type=%d errcode=%d data=%s\n",
              ssid, type, code, data != NULL ? (const char *)data : "NULL");
      break;

   case ESM_TYPE_MAILDONE:
      dprintf("Session %d: Got maildone. Non-fatal errs=%s\n", ssid,
             (char *)data == NULL ? "NONE" : (char *)data);
      break;

   case ESM_TYPE_CLOSED:
      dprintf("Session %d closed non-fatal errs=%s\n", ssid,
            (char *)data == NULL ? "NONE" : (char *)data);
      break;
```

```
   }
}

/* Callback function to create text for the email body
 * Application will write data into "buf" provide by ESMTP
 * On input, "*len" is size of buf
 */
int
mybody_cb(int ssid, char *buf, int *len)
{
   int i;
   static int bytestosend = 600; /* Message body will be 600 'a' characters */
   int outlen = min(*len, 600);  /* Number of bytes to write this time */

   for (i = 0; i < outlen; i++)
      buf[i] = 'a';              /* Write 'a' characters into buf */

   *len = outlen;                /* Number of bytes written into buf */
   bytestosend -= outlen;
   if (bytestosend == 0)
      return (0);                /* We done. All bytes passed to esmtp */

   return(ESM_CALLAGAIN);      /* We have more to write */
}
```

9.3 esmtp_w32testapp.c

As provided with the source code distribution, the file "`extras/esmtp/esmtp_w32testapp.c`" directory provides a simplified example of an email application. It provides examples for how to call ESMTP's APIs, and it provides simple versions of the required callback functions.

Often the easiest way to develop your own email application would be to first build it with Visual Studio and link it in with the ESMTP library in the w32_nichetask_vc project. In this case, you could start with esmtp_w32testapp.c and follow the instructions in the extras/esmtp/readme.txt file for how to integrate it into the ESMTP module as a test application. Once it is integrated and running, you could gradually transform the code into your own application, testing it as you go. Once your application is developed and working, you would then port it to your target platform.

9.4 Callback Functions

The names of the callback function are unimportant, but these three are required for interraction with the ESMTP module.

The only required callback function is the cb_func pointer passed with the esmtp_start API. The callback is used to report the status of the email session. There are two optional callbacks for functions that will be used to produce data for the email body or to produce data for an email attachment.

cb_func

API Name

`cb_func() - Required application callback function used by ESMTP to report the final status on an email session`

Syntax

```
void (*cb_func)(int, int, int, void *));
```

Parameters

<table>
<tr><td>int ssid</td><td>SSID for this session</td></tr>
<tr><td>int status</td><td>Status for this email session. Defined value are (see esmtp_port.h):
<table>
<tr><th>define name</th><th>value</th><th>description</th></tr>
<tr><td>ESM_TYPE_FATAL</td><td>1</td><td>An error occurred which will cause the session to close. The email was not delivered</td></tr>
<tr><td>ESM_TYPE_MAILDONE</td><td>2</td><td>Email session closed after the email server accepted the email. Server responsible for delivery</td></tr>
<tr><td>ESM_TYPE_CLOSED</td><td>3</td><td>Email session closed without the email server accepting the email. The email will not be delivered.</td></tr>
</table></td></tr>
<tr><td>int error</td><td>Zero for success or one of the ESMERR codes defined in esmtp_port.h</td></tr>
<tr><td>void *data</td><td>Usually NULL, but may contain a pointer to a string describing one or more non-fatal errors.</td></tr>
</table>

Description

This callback is a required parameter for the esmtp_start API. It is used to report the results of the email session: error value, email delivered, or session closed without email delivery. Most ESMTP errors are fatal, causing the email session to close. However, errors related to individual email recipients (improperly formatted, rejected by email server, etc.) are not fatal as long as there is at least one valid recipient. When called, *data* may contain a pointer to a string of information about one or more errors related to recipients.

Returns

Nothing.

BODYDATACB

API Name

`BODYDATACB() - Callback function that produces ASCII data for the message body`

Syntax

```
int (*BODYDATACB)(int ssid, char *buf, int *len);
```

Parameters

ssid	SSID for this session
buf	ESMTP provided buffer where the callback function should write the data
len	At invocation, the maximum size of 'buf'. Upon return, the number of bytes put in 'buf' by application

Description

This application callback function is passed to ESMTP by the esmtp_bodyfunctext() API. After an email session has been opened to the email server and the email headers have been sent, ESMTP will call this application function to obtain the data for the email body. The function should write from 0 to “len” bytes of ASCII data into the provided buffer and set the `len` parameter to the number of bytes written.

Notes

- The callback **must** update (`* len`) in addition to providing the proper return value.

Returns

- `ESM_CALLAGAIN`: After it has sent the data from this call, ESMTP should call the function again to obtain more data. ESMTP will call the function again at the next opportunity, even if no data was provided in this call.
- `0`: All data has been passed. Do not call again.

ATTACHDATACB

API Name

`ATTACHDATACB() - Callback function that produces data for an attachment`

Syntax

```
int (*ATTACHDATACB)(int ssid, char *buf, int *len);
```

Parameters

ssid	SSID for this session
buf	ESMTP provided buffer where the callback function should write the data
len	At invocation, the maximum size of 'buf'. Upon return, the number of bytes put in 'buf' by application

Description

This application callback function is passed by the esmtp_attachfunctext() API if it will produce only ASCII data, or by the esmtp_attachotherfunc() API if it will produce non-ASCII data.. During email session, after the email body has been sent to the email server, ESMTP will call this application function to obtain the data for an email attachment. The function should write from 0 to “len” bytes of data into the provided buffer and set the “len” parameter to the number of bytes written.

Notes

- The callback **must** update (`* len`) in addition to providing the proper return value.

Returns

- `ESM_CALLAGAIN`: After it has sent the data from this call, ESMTP should call the function again to obtain more data. ESMTP will call the function again at the next opportunity, even if no data was provided in this call.
- `0`: All data has been passed. Do not call again.

10 API

10.1 esmtp_startsession

API Name

`esmtp_startsession() - Start an ESMTP session to send one email`

Syntax

```
int esmtp_startsession(char *server, char *port, uint32_t flags, char
*username, char *password, void (*cb_func)(int, int, int, void *));
```

Parameters

server	domain name of email server
port	port to use on email server
flags	bit field where the bits represent features for this email session

ESMCF_USESSL	0x01	Use SSL to connect to server
ESMCF_USEAUTH	0x02	Authenticate via usernames and passwords
ESMCF_IP4	0x04e	Prefer IPv4 for this connection to server
ESMCF_IP6	0x08	Prefer IPv6 for this connection to server

username	user name for users account on specified email server
password	password for users account on specified email server
cb_func	application callback function that ESMTP module will use to report final email status

Description

This routine is used to pass the basic parameters for an email session. ESMTP will:

- validate the parameters
- alloc the required ESMPT memory pools if they do not already exist
- Use DHCP to obtain an IP address for the specified server
- Save the session parameters in the session memory pool
- Return the SSID to be used with all APIs and callback functions.

Returns

Positive Session ID (SSID) or one of the negative error codes listed in esmtp_port.h

10.2 esmtp_param

API Name

`esmtp_param() - Store a single parameter (from, to, etc) for current mail session.`

Syntax

```
int esmtp_param(int ssid, uint16_t type, char *param);
```

Parameters

ssid	value returned by esmtp_startsession()
type	One of the first 6 non-data parameter types defined in `esmtp_port.h`: • ESMTP_FROM • ESMTP_REPLYTO • ESMTP_TO • ESMTP_CC • ESMTP_BCC • ESMTP_SUBJECT
param	String containing a single parameter.

Description

This API is used to pass parameters used in the header fields of the email message. Only one parameter may be passed with each call. All addresses must be in the form: mailbox@domain name (e. g. emailname@yahoo.com). A separate call must be made to pass each address. All of the calls that take an address may be called multiple times.

The call is repeated for each parameter.

Notes:

1. The address used for ESMTP_FROM must be a registered user on the email server specified in the esmtp_startsession() call.
2. The must be at least one call to ESMTP_FROM and there must be at least one recipient.

Returns

0 for success or one of the negative esmtp error codes.

10.3 esmtp_exec

API Name

`esmtp_exec() - Execute the email command`

Syntax

```
int esmtp_exec(int ssid);
```

Parameters

`ssid`	value returned by `esmtp_startsession()`

Description

This API tells ESMTP that the user has finished passing all parameters for this email session and ESMTP should send the email. ESMTP will:

1. open a session with the specified email server
2. use the SMTP protocol to pass all of the header and data information to the email server. Obtaining the data for the email body may require opening and reading a specified file or making one or more calls to a callback function that produces email body data.
3. Call the session cb_func to report the final status of the email.

Returns

0 for success or one of the negative esmtp error codes.

10.4 esmtp_quitbyssid

API Name

`esmtp_quitbyssid() - Close (abort) an active email session`

Syntax

```
int esmtp_quitbyssid(int ssid);
```

Parameters

`ssid`	value returned by esmtp_startsession()

Description

This API is used to abort an active email session. It may be used at any time between the call to esmtp_startsession() and ESMTP's call to the cb_func that indicates the final status of the email session. If ESMTP has already opened an SNTP connection to the email server, it will send an SMTP QUIT command. All session memory will be freed as a result of this command.

Calling this API after ESMTP's call to the application's callback function will result in a ESMTP_CONN_NOTFOUND error.

Returns

0 for success or one of the negative esmtp error codes.

10.5 APIs for providing data for the email body and attachments

There are 11 APIs that the application can use to provide data for email messages. Five are for providing data for the body, one for each possible source of the data. There are only three possible sources for attachment data, but there are separate APIs depending on whether the data is entirely ASCII text or it may contain non-textual data.

10.6 Body Data APIs

An application can provide ESMTP the data for the email body via one of the five sources listed in the table above. There is an application API for each source:

esmtp_body

API Name

`esmtp_body - Create the message's "body"`

Syntax

Data from buffer

```
int esmtp_bodybuftext(int ssid, char *bufptr);
```

Data from file

```
int esmtp_bodyfiletext(int ssid, char *filename);
```

Data from Function

```
int esmtp_bodyfunctext(int ssid, BODYDATACB funcptr);
```

Data from Command

```
int esmtp_bodyclicmdtext(int ssid, char *cmdstr);
```

Data from script

```
int esmtp_bodyscripttext(int ssid, char *scriptname);
```

Parameters

ssid	Value returned by `esmtp_startsession()`.
bufptr	Pointer to an application provided buffer containing ASCII data. The end of the data is indicated by a NUL. There cannot be any other NULs or non-ASCII data preceding this NUL. The buffer and the data must remain constant until the email has been accepted by the email server as indicated by a call to the application's main callback function.
filename	Pointer to the name of a file that ESMTP can open and read. In the NicheStack demo for Windows, the file must be in the same directory as `iniche.exe` (e.g.: "`src41 /ReferencePorts/w32_nichetask_vs`").
funcptr	Pointer to an application provided callback function. While the email session between ESMTP and the email server is in progress, ESMTP will call this one or more times to obtain the data for the body (See description of `BODYDATACB()`)
cmdstr	Pointer to a NUL terminated CLI command string where the string is a command name plus any parameters exactly as it would be typed at a console. This string must remain constant until the email has been accepted by the email server as indicated by a call the application's main callback function. Note: This is only available for the message body
scriptname	Pointer to the name of a script file that ESMTP can open and read. The script file will contain one or more CLI command strings. Note: This is only available for the message body

Description

These APIs tell ESMTP how to obtain the data for the body of an email. There is a separate API for each of the five possible sources of data. The second parameter of each API is a pointer a specific type of source. A single email can only have one body, so only one of these calls can be used with each email. In all cases, the body can only contain ASCII data.

Returns

0 for success or one of the negative ESMTP error codes.

10.7 APIs for providing non-ASCII data for an attachment

CLI commands and scripts cannot be used to provide attachment data. An application can provide attachment data to ESMTP via one of three sources. There is an application API for each source.

esmtp_attachtext

API Name

`esmtp_attachtext - Create an ASCII Attachment`

Syntax

```
int esmtp_attachbuftext(int ssid, char *buf, char *displayname);

int esmtp_attachfiletext(int ssid, char *filename, char *displayname);

int esmtp_attachfunctext(int ssid, ATTACHDATACB funcptr, char
*displayname);
```

Parameters

ssid	Value returned by `esmtp_startsession()`.
buf	Pointer to an application provided buffer containing ASCII data. The end of the data is indicated by a NUL. There cannot be any other NULs or non-ASCII data preceding this NUL. The buffer and the data must remain constant until the email has been accepted by the email server as indicated by a call to the application's main callback.
filename	Pointer to the name of a file that ESMTP can open and read. In the NicheStack demo for Windows, the file must be in the same directory as `iniche.exe` (example: "`src41/ReferencePorts/w32_nichetask_vs`").
funcptr	Pointer to an application provided callback. While the email session between ESMTP and the email server is in progress, ESMTP will call this one or more times to obtain the data for the body or an attachment (See BODYDATACB and ATTACHDATACB).
displayname	Suggested display name. It should not contain path information. If the attachment data comes from a file, the display name need not be the same as the source file name. The sending application can only suggest a name to be used. It cannot suggest path information. The email display program on the final delivery system makes the final determination of the name displayed for an attachment.

Description

These APIs tell ESMTP how to obtain the data for one email attachment. There's a separate API for each of the 3 possible sources of data. The second parameter is a pointer to a specific type of source. An email may have multiple attachments, so any of these APIs may be called multiple times.

Returns

0 for success or one of the negative ESMTP error codes.

esmtp_attachother

API Name

esmtp_attachother - Create a Non-ASCII Attachment

Syntax

```
int esmtp_attachotherbuf(int ssid, uint8_t *buf, char *displayname, int
numbytes,char *mimetype, char *mimesubtype, char *mimeparam);

int esmtp_attachotherfile(int ssid, char *filename, char *displayname, char
*mimetype, char *mimesubtype, char *mimeparam);

int esmtp_attachotherfunc(int ssid, ATTACHDATACB funcptr, char
*displayname, char *mimetype, char *mimesubtype, char *mimeparam);
```

Parameters

ssid	value returned by `esmtp_startsession()`
buf	Pointer to an application provided buffer. The buffer and the data must remain constant until the email has been accepted by the email server as indicated by a call the application's main callback function.
filename	Pointer to the name of a file that ESMTP can open and read. In the NicheStack demo for Windows, the file must be in the same directory as iniche.exe (`src41/ReferencePorts/w32_nichetask_vs`).
funcptr	Pointer to an application provided callback function. While the email session between ESMTP and the email server is in progress, ESMTP will call this one or more times to obtain the data for the body or an attachment (See description of `BODYDATACB()` and `ATTACHDATACB()`)
displayname	Suggested display name. It should not contain path information. If the attachment data comes from a file, the display name need not be the same as the source file name. The sending application can only suggest a name to be used. It cannot suggest path information. The email display program on the final delivery system makes the determination of the name displayed for an attachment.
mimetype	Pointer to a NUL terminated string name for one of the top-level Media Types as described in RFC 2046. Other types are possible as long they are understood by the specified email server. Examples: "`text`", "`image`", "`Application`"
mimesubtype	Pointer to a NUL terminated string name for a Media subtype. Many subtypes are possible as long they are understood by the specified email server. Examples: `gif`, `octet-stream`.
mimeparm	NULL or a pointer to a NUL terminated string that adds a MIME-type qualifier to the mimetype and subtype. For example, it may specify a language or a character set.

Description

These APIs tell ESMTP how to obtain the data for one email attachment. An email may have multiple attachments, so any of these APIs may be called multiple times.

Examples:

```
esmtp_attachotherfile(3, "pic123", "yourpicture.jpg", image, jpeg, NULL);
esmtp_attachotherfunc(4, sensorfunc, "currdata", "application", "octet-stream",
```

Returns

0 for success or one of the negative ESMTP error codes.