

sevenstax Simple Mail Transfer Protocol

SMTP User Manual

Revision No.: 1.06
State: Released
Author: Stefan Suffenplan, sevenstax GmbH

Initial version: 28/11/02
Last change: 08/08/03
Changed by: Stefan Suffenplan
Last Review: 08/08/2003
Publication: 08/08/2003
Filename: sevenstaxSMTP_UserManual_106.pdf

Table of Contents

1 Introduction to sevenstaxSMTP.....3

2 Mode of Operation.....4

3 Public Functions.....7

4 Application Defined Handler Functions.....10

 4.1 Notify Handler Function.....10

 4.2 Send Data Handler Function.....10

 4.3 Get Second Counter Handler Function.....10

 4.4 Get Mail Handler Function.....10

5 Notify-Codes.....11

6 Configuring sevenstaxSMTP.....12

7 Example.....13

 7.1 Application fragments.....13

 7.2 Application flow.....22

8 Description of Files.....24

9 Global definitions.....25

10 Change History.....26

1 Introduction to sevenstaxSMTP

sevenstaxSMTP is an implementation of the SMTP protocol. SMTP stands for "Simple Mail Transfer Protocol" and is the world wide standard protocol for sending mails. The implementation is made for use in micro controllers but can also be used on every other platform.

The main tasks of sevenstaxSMTP are on the one hand to build and send SMTP commands and mail data to the SMTP server and on the other hand to interpret the responses from the SMTP server.

sevenstaxSMTP ensures sending mails over those SMTP servers that accepts the authentication method "plain" or that don't need any authentication.

Not part of sevenstaxSMTP is to establish a TCP connection to the SMTP server. This has to be done from the application itself (e.g. by using sevenstaxTCP) before starting the sending process. To send data over TCP to the server the application must provide a corresponding handler function. This handler function will be called automatically by sevenstaxSMTP. On TCP data receipt (SMTP server responses) the application has to call the SMTP parser function.

Also the MIME encoding of the mail is not part of sevenstaxSMTP because it is not a task of SMTP - SMTP is "only" capable to deliver a given mail to the SMTP server. But you can use sevenstaxMIME to realise the mail encoding (with respect to the Internet message format and the MIME standard) and delivering the mail in packets. In this case a corresponding handler function must be registered. Please note that MIME does not only support the usage of attachments but also of special characters. SMTP supports only 7 bit ASCII characters. There is an enhanced version of SMTP which is capable to process 8 bit ASCII characters too. Although this enhanced SMTP runs on many current SMTP servers there is no guarantee of an 8 bit ASCII support.

If you do not want to send an encoded mail, you can use sevenstaxSMTP to send not encoded strings or a sequence of characters directly. Most SMTP servers will add corresponding header fields and send the mail correctly although the mail could have 8 bit coded characters. In this case you do not need to give sevenstaxSMTP the handler function described above.

At last a notify handler function should be given by the application which is called from sevenstaxSMTP on successful termination or on error.

2 Mode of Operation

The following figure shows the integration of the sevenstaxSMTP module into the system.

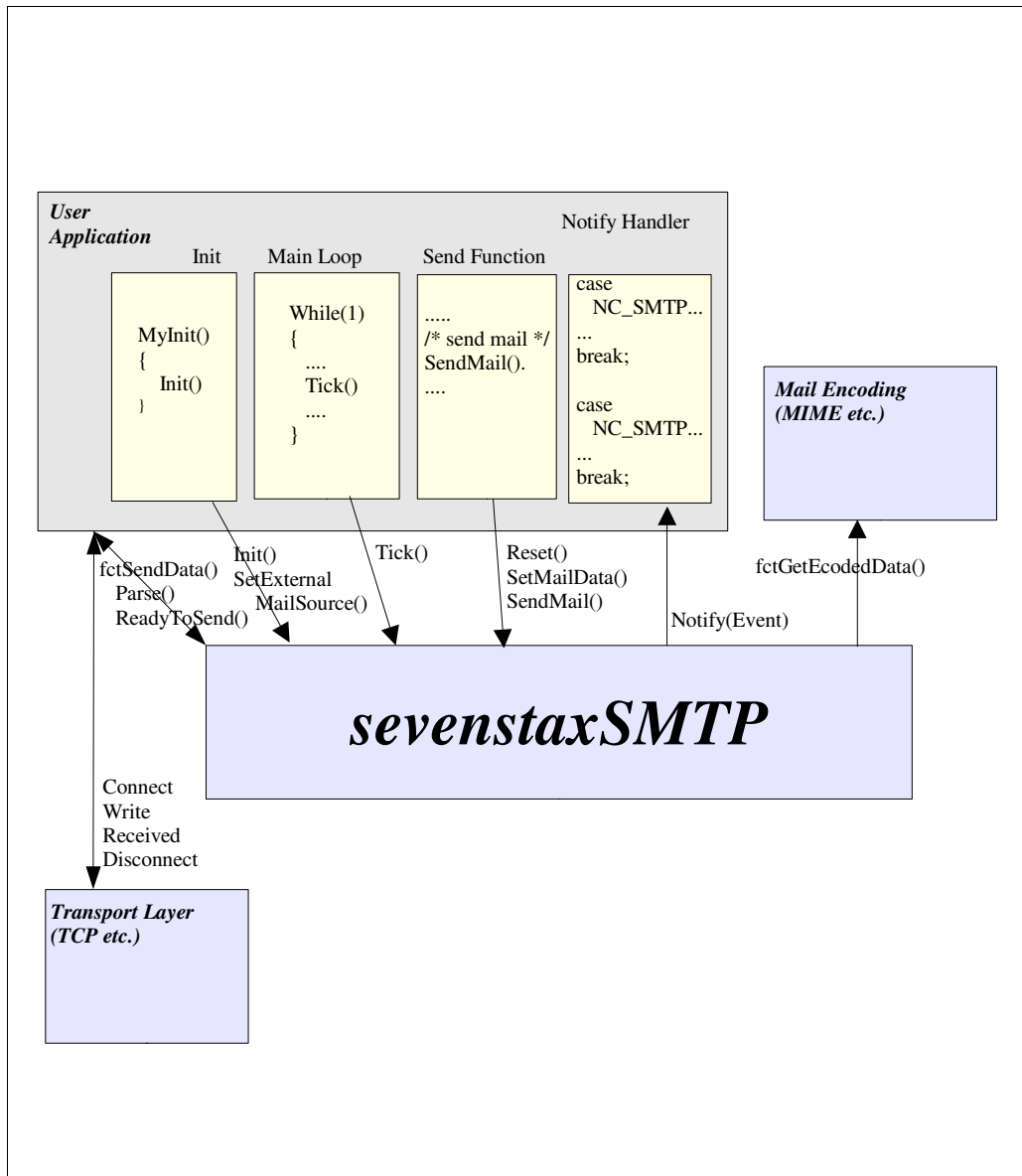


Fig 1. Integration into system

As described above, sevenstaxSMTP is designed to work in conjunction with different TCP implementations (transport layer) and if needed any mail encoder (MIME etc.).

The interface to send data over the transport layer consists of a handler function that must be registered when calling the SMTP init function *stxSMTP_Init* (s.b.). If sevenstaxSMTP wants to send data (SMTP commands or mail data) to the server it will call this registered function which might be a direct call to the TCP's write function or a function that calls a TCP's write function on her part. (The sevenstaxTCP write function can not be used directly because it expects a connection id. This id is the result of establishing a TCP connection using sevenstaxTCP – as described in the introduction the establishing of TCP connections is not part of SMTP in general.)

sevenstaxSMTP is a stateful protocol that has to be informed by calling the public SMTP function *stxSMTP_ReadyToSend* when the sending process has been completed. That must be done by the application as all results are directly reported to its notify handler. The application has to call this function when data was successfully delivered over TCP. The detection of this event depends on the used TCP/IP stack. If you use sevenstaxTCP the TCP notify handler function will be called with an appropriate notify code.

The interface to the mail encoder (if used) consists of a handler function that delivers encoded mail packets. These mail packets are requested from sevenstaxSMTP and the size is given by the size of the send block (SIZE SMTP SEND BLOCK). With this method streaming mail encoders are supported. The main advantage of this method is that the encoded mail must not exist in an entire block to save memory. This function is also registered within the SMTP init function *stxSMTP_Init* (s.b.). A good choice is to use a function from sevenstaxMIME that represents a streaming mail encoder.

If you do not want to use a mail encoder but send only a constant string directly via SMTP you can call the init function with NULL as mail encoder function pointer. In this case the string is delivered via the function *stxSMTP_SetMailData* (s.b.).

sevenstaxSMTP must be informed which method it has to use. This is done with the function *stxSMTP_SetExternalMailSource* (s.b.). With parameter TRUE it uses the first alternative, that means the external mail encoder function, whereas FALSE chooses the second alternative (constant string). In most cases this function is called in the init routine but it is also possible to call it whenever you want to dynamically switch between these methods.

Like the other handler routines the notify function is also set within the init function *stxSMTP_Init* (s.b.). This notify function is called on several SMTP events, e.g. if the process of sending a mail has failed or successfully finished.

sevenstaxSMTP checks several timing conditions. If for instance the SMTP server does not response within a certain time, sevenstaxSMTP will cancel the mail sending process and will call the notify function with an appropriate notify code. Because the determination of the actual time depends on the software/hardware environment the application must provide a function that returns a two byte second counter. This function also has to be registered in the init routine *stxSMTP_Init* (s.b.) which thus has altogether four function pointers.

Depending on the implementation of the application and transport layer the receipt of data is signalled in a different form (message/callback routine/polling/...). So it is a task of the application to pick up the received data and deliver them to the SMTP module. To do this the SMTP parser function *stxSMTP_Parse* (s.b.) has to be called. The parser extracts the SMTP return code and recognises the end of the server response. Depending of the SMTP return

code it forces the sending of the next SMTP command after the end of the response was detected. The SMTP parser classifies the return codes into two parts: less than 400 (no error) and equal to or greater than 400 (error). On error SMTP terminates the sequence and calls the notify function with an error notify code.

Every time when the application has established a (TCP) connection to the SMTP server it must call the reset function *stxSMTP_Reset* (s.b.). In this case the SMTP module resets its internal states and waits for a greeting message from the SMTP server which will be replied by an SMTP "HELO" command in order to send an email. If the reset function is not called the SMTP module will consider the (TCP) connection as being active for a longer time. (In this case it would not wait for a greeting message and, when the sending mail request is set (s.b.), begin with the "MAIL FROM" command)

Every time when a TCP connection to the SMTP server is active the application can request sevenstaxSMTP to send a mail. To do this the function *stxSMTP_SendMail* (s.b.) must be used.

If this is the first call after activating the (TCP) connection (*stxSMTP_Reset* called, s.a.) sevenstaxSMTP will not start immediately its request sequence. This is because first the SMTP server sends a greeting message and sevenstaxSMTP waits for that. After the application has delivered this greeting message to SMTP (*stxSMTP_Parse*, s.a.), the request sequence is started by sevenstaxSMTP.

There is a tick function that acts as the heart beat of the SMTP module. Its name is *stxSMTP_Tick*, s.b., and must be called cyclically as often as possible. This function is among other things necessary to check the time-outs with regard to the SMTP server.

3 Public Functions

sevenstaxSMTP has a small public interface. Please see the following descriptions for details of the API (Application Programming Interface).

All used macros and type definitions are listed in the chapter "Global definitions".

- void **stxSMTP_Init**
(PROTOCOL_NOTIFY_HANDLER fctNotifyPara,
SMTP_SENDDATA_FCTTYPE fctSendDataPara,
SMTP_GETSECCOUNTER_FCTTYPE fctGetSecCounterPara,
SMTP_GETMAIL_FCTTYPE fctGetMailPara)

This function initialises the SMTP control sub-module. It must be called one time during the init process. The parameters specifies the application defined handler routines. They should be not equal to null. In the other case sevenstaxSMTP will not work correctly.

fctNotifyPara points to the notify function which is called from sevenstaxSMTP on several SMTP events described later.

fctSendDataPara points to the function that will send data over the transport layer. sevenstaxSMTP calls this handler function when it wants to send data to the SMTP server. Normally you will use TCP as transport layer. As many TCP write functions (like that of sevenstaxTCP) requires as additional parameter a connection id they can not be registered directly to SMTP. Instead an application-specific write function should be used that knows the connection id and that satisfies the SMTP write function requirements.

fctGetSecCounterPara points to the function that returns the actual second counter. This handler function is called from sevenstaxSMTP to control timeouts.

fctGetMailPara points to the function that returns packets of the mail data to send. This handler function is called from sevenstaxSMTP when it wants to send the next (where applicable encoded) mail packet. By this method the encoding can be processed in a streaming manner. As the encoding of a mail is not SMTP specific but is based on other standards like MIME the mail generation is not part of sevenstaxSMTP module. Thus the sevenstaxSMTP does not have to know anything about mail data like the mail subject. A good solution is to use the sevenstaxMIME (Multipurpose Internet Mail Extensions) module that ensures a streaming encoding and decoding of mails. If you only want to send a constant string over SMTP this parameter can be a null pointer. The constant string has to be defined later.

The formats of the four handler functions are described in the next chapter.

- void **stxSMTP_Reset**
(void)

This function must be called every time when a (TCP) connection to the SMTP server has been established. This is necessary because the SMTP module does not handle the server connection on the transport layer (TCP). If a new (TCP) connection has been established (and thus this reset routine was called) the SMTP module knows that it has to connect to the SMTP server on SMTP level (waiting for a greeting message, sending "HELO" command etc.).

- void **stxSMTP_Tick**
(void)

This function handles as the heart beat of the sevenstaxSMTP. It must be called cyclically as often as possible to assure a fast sending of emails.

- void **stxSMTP_SetMailData**
(STRING szMailRawDataPara)

If you don't want to use a mail encoder in a form described in the init routine but only want to send a raw constant string directly via SMTP, you can specify this string in this function.

This strings **must** remain unchanged until the registered notify handler function was called with a notify code which states that the SMTP process is ready or has failed. This is because sevenstaxSMTP doesn't copy the delivered string but accesses them directly to save memory.

- void **stxSMTP_SetExternalMailSource**
(BOOL bExternal)

Specifies the mail source. If `bExternal` is TRUE sevenstaxSMTP will use an external mail encoder specified in the SMTP init routine. If the value is FALSE the string delivered in the "set mail data" function above will be used. The default value is FALSE.

- void **stxSMTP_SendMail**
(STRING szSMTPServerUsernamePara,
STRING szSMTPServerPasswordPara,
STRING szSenderMailAddressPara,
STRING szRcptPara)

This main function is used to send a mail via SMTP.

The first two parameters indicate the user name and password of the SMTP server account and are used to perform an authentication process where applicable. In many cases this is the same data used when fetching mails POP3. If you are sure that your SMTP server does not need authentication you do not have to specify these parameters. The maximum length for the first two parameters is 65535 chars.

The third parameter specifies the sender of the mail. This can be only the raw mail address or the extended format that consists of a name followed by a '<', followed by the mail address and terminated by a '>' (f.e. "Nick <nick@name.xy>"). The last parameter specifies a list of recipients, separated by commas. Every recipient can be specified by the mail address or by the extended format, f.e. "Näck <naeck@name.xy>, nock@name.xy". The length of the sender or of every recipient can respectively have a maximum length of 255.

These strings **must** remain unchanged until the registered notify handler function was called with a notify code which states that the SMTP process is ready or has failed. This is because sevenstaxSMTP doesn't copy the delivered strings but accesses them directly to save memory.

- void **stxSMTP_ReadyToSend**
(void)

This function is called from the application if it has noticed that the current data packet was successfully sent to the server via transport layer (TCP). For the SMTP module that means that it can f.e. build the next packet and send it.

- BOOL **stxSMTP_Parse**
(STRING pBuffer,
UINT16 uCount)

This function must be called every time when a data packet from the SMTP server was received. The data packet is delivered in the first parameter and the count of bytes in the second. The return value indicates if an error is encountered (TRUE on error, FALSE else).

4 Application Defined Handler Functions

In this chapter all necessary handler functions are described in detail. All handler functions are registered within the SMTP init function.

All used macros and type definitions are listed in the chapter "Global definitions".

4.1 Notify Handler Function

sevenstaxSMTP requires a notify handler procedure provided by the application. sevenstaxSMTP calls this handler procedure which must be of the type `PROTOCOL_NOTIFY_HANDLER` on every event.

The first parameter `code` of the handler function is called the notify code. This is actually an identifier of the message which sevenstaxSMTP has sent. With one of the `NC_SMTP_XXX` codes (s.b.), sevenstaxSMTP will call the handler procedure to inform about occurred events.

The two parameters `lparam` and `wparam` may carry additional informations depending on the message.

4.2 Send Data Handler Function

To send data to the SMTP server sevenstaxSMTP needs a handler function provided by the application. Whenever sevenstaxSMTP wants to send data it calls this handler procedure which must be of the type `SMTP_SENDDATA_FCTTYPE`.

The first parameter points to the packet that has to be sent and the other parameter determines the size of the packet. In most cases a TCP write function will be called from the handler function.

4.3 Get Second Counter Handler Function

To demand the current second counter (f.e. to handle timeouts) sevenstaxSMTP needs a handler function provided by the application. Whenever sevenstaxSMTP needs the current value of the second counter it calls this handler procedure which must be of the type `SMTP_GETSECCOUNTER_FCTTYPE`.

The function has no parameter. The return value is a two byte value (UINT16).

4.4 Get Mail Handler Function

To get a packet of the (encoded) mail to be sent, sevenstaxSMTP calls a handler function provided by the application. Whenever sevenstaxSMTP requests the next mail packet it calls this handler procedure which must be of the type `SMTP_GETMAIL_FCTTYPE`.

The first parameter indicates if the data request is called first or repeatedly (`FIRST/NEXT`). The second parameter points to a target buffer where the encoded mail data has to be placed and the last parameter determines the size of the target buffer. The handler function there places the count of chars which it has been copied effectively. If the number is smaller than the delivered size of the target buffer, the SMTP module assumes that it was the last mail data packet.

Note: This handler function does not have to be defined if only a raw string is to be sent.

5 Notify-Codes

The notify function is called with one of the following notify codes.

- `NC_SMTP_READY`

This event is issued after a successful mail delivery.

- `NC_SMTP_ERRCODE`

This event occurs if the SMTP server has replied with an error code. The SMTP error code is delivered in the parameter `lparam`.

- `NC_SMTP_NO_ANSWER`

This event occurs if the SMTP server has not given a response within one minute.

- `NC_SMTP_WRONG_ANSWER`

This event occurs if the SMTP server response could not be identified (f.e. wrong format).

6 Configuring sevenstaxSMTP

There are some parameters defined in 'smtpx.h' which can be customized. These parameters should properly be adjusted depending on the used notification codes and free memory/time resources. The following list describes them in detail:

- `SIZE SMTP SEND BLOCK` *default: 128*
Defines the size of the send block. If the size is small there are sent many small packet to the server and so the time to send a mail is elongated. If the size is large there are sent less and larger packets but mail sending is processed faster.
- `NC SMTP . . .` *default: 1024...*
Defines the SMTP notify code numbers (see above).

7 Example

7.1 Application fragments

```
/** smtp settings */
static unsigned char PTR  szSMTPServer          = "auth.mail.onlinehome.de";
// smtp server
static unsigned short     uSMTPServerPort      = 25;
// smtp server port
static unsigned char PTR  szSMTPServerLogin     = "cc33806830-1004";
// sevenstax smtp server test account name (for smtp auth)
static unsigned char PTR  szSMTPServerPassword = "geheim";
// smtp server password (for smtp auth)
static unsigned char PTR  szSenderMailAddress  = "SevenstaxSMTP <smtp@sevenstax.de>";
// sender mail address
static unsigned char PTR  szRcptMailAddresses  = "SevenstaxSMTP <smtp@sevenstax.de>";
// mail recipient
```

```
/* SIMULATION OF USER APPLICATION */
typedef enum
{
    MAILAPP_IDLE,
    MAILAPP_GO_ONLINE,
    MAILAPP_DO_CONNECT_SERVER,
    MAILAPP_CONNECTING_SERVER,
    MAILAPP_DO_SEND_MAIL,
    MAILAPP_SENDING_MAIL,
    MAILAPP_STAY_ONLINE,
    MAILAPP_OFFLINE
} APPLSTATE;
```

```
static APPLSTATE eMailAppState = MAILAPP_IDLE;

void MailApp_SendData          (STRING pRequest, UINT16 uSize);
UINT16 MailApp_GetSecCounter   (void);
void MailApp_ConnectMailServer (void);
```

```
UINT32 MailApp_NotifyHandlerSMTP
(NOTIFY_CODE uNotifyCode, UINT32 lParam, UINT16 wParam);
UINT32 MailApp_NotifyHandlerISP
(NOTIFY_CODE uNotifyCode, UINT32 lParam, UINT16 wParam);
UINT32 MailApp_NotifyHandlerTCP
(NOTIFY_CODE uNotifyCode, UINT32 lParam, UINT16 wParam);
```

```
void MailApp_Init(void)
{
    DC_IspSetCallback(MailApp_NotifyHandler);
    /* init SMTP control & parser module */

    stxSMTP_Init(MailApp_NotifyHandlerSMTP,
                MailApp_SendData,
                MailApp_GetSecCounter,
                stxMIME_GetEncodedMail);
}
```

```
/* SIMULATION OF USER APPLICATION */
void MailApp_Tick(void)
{
    UINT16 uSec;
    stxSMTP_Tick();
    switch (eMailAppState)
    {
        case MAILAPP_IDLE:           // give three more seconds before dialing in ISP
            TimerGetSys_sec(uSec);
            if ((uSec - wStateStartTime) > 3)
            {
                if (DC_IspConnect(szISPTelNr, szISPLogin, szISPPassw) == DC_OK)
                    eMailAppState = MAILAPP_GO_ONLINE;
                else
                    eMailAppState = MAILAPP_IDLE;
                wStateStartTime = uSec;
            }
            break;

        case MAILAPP_GO_ONLINE:
            break;

        case MAILAPP_DO_CONNECT_SERVER:
            /* connect to mail server ... */
            MailApp_ConnectMailServer();
            eMailAppState = MAILAPP_CONNECTING_SERVER;
            break;

        case MAILAPP_CONNECTING_SERVER:
            break;
    }
}
```

```
case MAILAPP_DO_SEND_MAIL:
    /* reset SMTP module */
    stxSMTP_Reset();
    /* set mail source to external */
    stxSMTP_SetExternalMailSource(TRUE);
    /* and continue to send the mail */
    stxSMTP_SendMail
        (szSMTPServerLogin, szSMTPServerPassword,
         szSenderMailAddress, szRcptMailAddresses);
    eMailAppState = MAILAPP_SENDING_MAIL;
    break;

case MAILAPP_SENDING_MAIL:    // wait for success/error from SMTP notify
                               // handler
    break;                    // this state is handled by
                               // MailApp_NotifyFct()

case MAILAPP_STAY_ONLINE:
    TimerGetSys_sec(uSec);
    // disconnect 10 Seconds after sending mail
    if ( ((uSec - wStateStartTime) > 10) &&
         (DC_IspGetState() == CONNECTED)
        )
    {
        DC_IspDisconnect();
        eMailAppState = MAILAPP_OFFLINE;
    }
    break;

case MAILAPP_OFFLINE:        // stay offline
    break;

default:
    break;
}
}
```

```
/******  
*   FUNCTION:      MailApp_SendData  
*   DESCRIPTION:   function that performs the sending of data  
*   PARAMETERS:   pRequest:  pointer to data to send  
*                 uSize:     count of bytes to send  
*   RETURN:       -  
*   COMMENT:      -  
*****/  
void MailApp_SendData (STRING pRequest, UINT16 uSize)  
{  
    DCERR err;  
  
    ODS (DBG_PRT,DBG_INFO,"MailApp_SendData...");  
    ODS3 (DBG_PRT,DBG_INFO,">>>> %s (%u). uConnID=%u",pRequest,uSize,uConnID);  
  
    /* call tcp write function */  
    err = DC_TCPWrite (uConnID, pRequest, uSize);  
  
    ODS1 (DBG_PRT,DBG_INFO,"MailApp_SendData hat %u zurueckgegeben",err);  
}
```

```
/******  
*   FUNCTION:      MailApp_GetSecCounter  
*   DESCRIPTION:   function that returns the actual second counter  
*   PARAMETERS:   -  
*   RETURN:       actual second counter  
*   COMMENT:      -  
*****/  
UINT16 MailApp_GetSecCounter (void)  
{  
    UINT16 uSec;  
  
    TimerGetSys_sec (uSec);  
  
    return (uSec);  
}
```

```
/* *****  
* FUNCTION:      MailApp_ConnectMailServer  
* DESCRIPTION:   connects to smtp server (isp is connected)  
* PARAMETERS:    -  
* RETURN:        -  
* COMMENT:       -  
***** */  
void MailApp_ConnectMailServer(void)  
{  
    if (uConnID == UINT8_MAX)  
    { /* we are not connected to SMTP server */  
        IPV4 ipSMTPServerIP;  
        /* create tcp connection by dns (server name) */  
        uConnID = DC_TCPCreateConnectionDns(szSMTPServer, uSMTPServerPort,  
                                            MailApp_NotifyHandlerTCP);  
    }  
    if (uConnID < UINT8_MAX)  
    { /* tcp connection is created */  
        /* get the connection state */  
        DC_STATE uConnState = DC_TCPGetState(uConnID);  
        DCERR eRet;  
        switch (uConnState)  
        {  
            case NOT_CONNECTED:  
                /* not connected */  
                /* connect now */  
                eRet = DC_TCPConnect(uConnID);  
                /* inform application if not successful */  
                if (eRet != DC_OK)  
                {  
                    DC_IspDisconnect(); /* disconnect from ISP */  
                    eMailAppState = MAILAPP_IDLE;  
                    TimerGetSys_sec(wStateStartTime);  
                }  
                break;  
            case CONNECTED:  
                /* connected */  
                DC_IspDisconnect(); /* disconnect from ISP */  
                eMailAppState = MAILAPP_IDLE;  
                TimerGetSys_sec(wStateStartTime);  
                break;  
            default:  
                break;  
        }  
    }  
}
```

```

/*****
*   FUNCTION:      MailApp_NotifyHandlerSMTP
*   DESCRIPTION:   notify functions for SMTP events
*   PARAMETERS:   uNotifyCode:  notify code (s. notifycodes.h)
*                 lParam:       long parameter of event
*                 wParam:       short parameter of event
*   RETURN:       0
*   COMMENT:      -
*****/
UINT32 MailApp_NotifyHandlerSMTP(NOTIFY_CODE uNotifyCode, UINT32 lParam, UINT16 wParam)
{
    switch (uNotifyCode)
    {
        case NC_SMTP_READY:
            /* SMTP is ready with an action */

            ODS(DBG_PRT,DBG_INFO,"Notify: NC_SMTP_READY");
            eMailAppState = MAILAPP_STAY_ONLINE;
            TimerGetSys_sec(wStateStartTime);
            break;

        case NC_SMTP_ERRCODE:
            /* SMTP has failed to perform an action */

            ODS(DBG_PRT,DBG_INFO,"Notify: NC_SMTP_ERROR");
            eMailAppState = MAILAPP_STAY_ONLINE;
            TimerGetSys_sec(wStateStartTime);
            break;

        case NC_SMTP_NO_ANSWER:
            /* SMTP has failed to perform an action */
            ODS(DBG_PRT,DBG_INFO,"Notify: NC_SMTP_NO_ANSWER");
            eMailAppState = MAILAPP_IDLE;
            TimerGetSys_sec(wStateStartTime);
            break;

        case NC_SMTP_WRONG_ANSWER:
            /* SMTP has failed to perform an action */

            ODS(DBG_PRT,DBG_INFO,"Notify: NC_SMTP_WRONG_ANSWER");
            eMailAppState = MAILAPP_STAY_ONLINE;
            TimerGetSys_sec(wStateStartTime);
            break;
    }
    return 0;
}

```

```

/*****
*   FUNCTION:      MailApp_NotifyHandlerISP
*   DESCRIPTION:   notify function for ISP events
*   PARAMETERS:   uNotifyCode:  notify code (s. notifycodes.h)
*                 lParam:       long parameter of event
*                 wParam:       short parameter of event
*   RETURN:       0
*   COMMENT:      -
*****/
UINT32 MailApp_NotifyHandlerISP (NOTIFY_CODE uNotifyCode, UINT32 lParam, UINT16 wParam)
{
    UINT32 RetVal = 0;

    switch (uNotifyCode)
    {
        case NC_PPP_CONNECTED:
            /* ppp connected */

            ODS(DBG_PRT,DBG_INFO,"Notify code: NC_PPP_CONNECTED");

            eMailAppState = MAILAPP_DO_CONNECT_SERVER;

            break;

        case NC_MDM_NO_CARRIER:
        case NC_MDM_CONNECTION_ERROR:
        case NC_MDM_BUSY:
        case NC_MDM_NO_LINE:
            /* isp failed to connect */

            ODS(DBG_PRT,DBG_INFO,"Notify code: NC_MDM_NO_CARRIER / NC_MDM_CONNECTION_ERROR
            / NC_MDM_BUSY / NC_MDM_NO_LINE");

            DC_IspDisconnect();
            eMailAppState = MAILAPP_IDLE;
            TimerGetSys_sec(wStateStartTime);

            uConnID = UINT8_MAX;

            break;

        default:
            break;
    }

    return (RetVal);
}

```

```

/*****
*   FUNCTION:      MailApp_NotifyHandlerTCP
*   DESCRIPTION:   notify function for TCP events
*   PARAMETERS:   uNotifyCode:  notify code (s. notifycodes.h)
*                 lParam:       long parameter of event
*                 wParam:       short parameter of event
*   RETURN:       0
*   COMMENT:      -
*****/
UINT32 MailApp_NotifyHandlerTCP(NOTIFY_CODE uNotifyCode, UINT32 lParam, UINT16 wParam)
{
    ABORT_REASON reason;
    UINT8 uTcpState = UINT8_MAX;
    UINT8 PTR pData = NULL;
    UINT16 uDataSize = 0;

    switch (uNotifyCode)
    {
        case NC_TCP_CONNECTED:
            /* tcp connected */
            uTcpState = TCP_CONNECTED;
            break;

        case NC_TCP_CONNTIMEOUT:
            /* connect timeout */
            uTcpState = TCP_CONNECTION_ERROR;
            break;

        case NC_TCP_FIN:
            /* tcp disconnected */
            reason = (ABORT_REASON) lParam;
            switch (reason)
            {
                case AR_CLOSED:           /* Closure because we send a FINACK (default) */
                    uTcpState = TCP_DISCONNECTED;
                    break;
                case AR_NO_MACADDRESS:
                case AR_CLOSE_FORCED:     /* Close by any other means */
                case AR_RTM_EXCEEDED:    /* Too many retransmissions */
                    uTcpState = TCP_CONNECTION_ERROR;
                    break;
                default:
                    break;
            }
            break;
    }
}

```

```
    case NC_TCP_RECV:
        /* data received */
        uTcpState = TCP_DATA_RECEIVED;
        pData = ((TCP_INFORMATION PTR) lParam)->data;
        uDataSize = ((TCP_INFORMATION PTR) lParam)->datasize;
        break;

    case NC_TCP_TXREADY:
        /* data sent, ready to send new data */
        uTcpState = TCP_DATA_SENT;
        break;

    default:
        break;
}

switch (uTcpState)
{
    case TCP_CONNECTION_ERROR:
        /* tcp connection error */
        DC_IspDisconnect();
        eMailAppState = MAILAPP_IDLE;
        TimerGetSys_sec(wStateStartTime);
        break;

    case TCP_CONNECTED:
        /* tcp connected */
        eMailAppState = MAILAPP_DO_SEND_MAIL;
        break;

    case TCP_DATA_RECEIVED:
        /* we received data */
        ODS1(DBG_PRT, DBG_INFO, "<<<< %s", (STRING)pData);
        if (eMailAppState == MAILAPP_SENDING_MAIL)
            /* deliver data packet to SMTP parser */
            bParseError = stxSMTP_Parse(pData, uDataSize);
        break;

    case TCP_DATA_SENT:
        /* data was successfully sent */
        /* inform SMTP control module */
        stxSMTP_ReadyToSend();
        break;

    default:
        break;
}
return 0;
}
```

7.2 Application flow

Below an example of a typical sequence of function calls from an application that uses sevenstaxSMTP:

- application initialises sevenstaxSMTP by registering four handler functions:
`stxSMTP_Init (MySMTPNotifyHandler,
 MyWriteTCP,
 MyGetSecCounter,
 MyGetMail);`
- application decides to use take the mail data from the external get mail function:
`stxSMTP_SetExternalMailSource (TRUE);`
- application starts TCP connection to SMTP server and waits, until TCP connection has been established.....
- application resets sevenstaxSMTP (every time after a new TCP connect):
`stxSMTP_Reset ();`
Please note that most SMTP server will close the TCP connection after a certain time of inactivity.
- application has received TCP data with greeting text from SMTP server and has stored them in buffer `szRcvBuf`, count of chars received stored in `uCntRcv`
- application delivers received data to sevenstaxSMTP parser:
`stxSMTP_Parse (szRcvBuf, uCntRcv);`
- application gives sevenstaxSMTP the order to send the mail with following parameters:
`szSMTPUser = "cc34567890-0123";
szSMTPPass = "geheim";
szMailSnd = "Tim Türling <tim@tuerling.de>";
szMailRcpt = "Tom Türling <tom@tuerling.de>, info@sevenstax.de";
stxSMTP_SendMail (szSMTPUser, szSMTPPass, szMailSnd, szMailRcpt);`
(parameters must not be changed until notify handler is called with ready or error code)
- sevenstaxSMTP requests to send the first command:
`MyWriteTCP ("HELO ...", size);`
- sending in progress
- sending ready
- application informs sevenstaxSMTP that data was sent successfully:
`stxSMTP_ReadyToSend ();`
- sevenstaxSMTP waits for the answer from the SMTP server because the command was sent completely.....
- application has received TCP data with a response text from SMTP server.
- application delivers received data to sevenstaxSMTP parser:
`stxSMTP_Parse (szRcvBuf, uCntRcv);`
-
- sevenstaxSMTP requests the first mail data packet:
`MyGetMail (FIRST, szBuf, &uLen);`

- sevenstaxSMTP requests to send this mail data packet:
MyWriteTCP(szBuf, uLen);
- sending in progress
- sending ready
- application informs sevenstaxSMTP that data was sent:
stxSMTP_ReadyToSend ();
- sevenstaxSMTP requests the next mail data packet:
MyGetMail(NEXT, szBuf, &uLen);
- sevenstaxSMTP requests to send this mail data packet:
MyWriteTCP(szBuf, uLen);
- sending in progress
- sending ready
- application informs sevenstaxSMTP that data was sent:
stxSMTP_ReadyToSend ();
-
- sevenstaxSMTP requests to send the last command:
MyWriteTCP("QUIT\r\n", 6);
- sending in progress
- sending ready
- application informs sevenstaxSMTP that data was sent:
stxSMTP_ReadyToSend ();
- application has received TCP data with a response text from SMTP server.
- application delivers received data to sevenstaxSMTP parser:
stxSMTP_Parse(szRcvBuf, uCntRcv);
- sevenstaxSMTP informs application that sending mail was successful:
MySMTPNotify(NC_SMTP_READY, 0xFFFF, 0xFF);

8 Description of Files

<i>File</i>	<i>Description</i>
smtp.c	sevenstaxSMTP main module - source code
smtp.h	sevenstaxSMTP main module - private header
smtpx.h	sevenstaxSMTP main module - export header
base64en.c	base64 encoder - source code (for SMTP authentication)
base64enx.h	base64 encoder - export header
protdefs.h	common definitions for all sevenstax protocols - header
standard.h	standard definitions - header
xtoa.c	number to string converter - source code
xtoax.h	number to string converter - export header

9 Global definitions

The following global type definitions and defines are used in this document.

standard.h:

```
typedef char far * STRING;
#define BOOL UINT8
typedef unsigned char UINT8;
typedef unsigned short UINT16;
```

protdefs.h:

```
typedef void (*PROTOCOL_NOTIFY_HANDLER) (NOTIFY_CODE code, UINT32 lparam, UINT16 wparam);
typedef UINT16 NOTIFY_CODE;
```

smtpx.h:

```
typedef void far (*SMTP_SENDDATA_FCTTYPE) (STRING, UINT16);
typedef void far (*SMTP_GETMAIL_FCTTYPE) (UINT8, STRING, UINT16 PTR);
```

10 Change History

<i>Vers.</i>	<i>Date</i>	<i>by</i>	<i>Change description</i>
1.00	28.11.02	Suf	Base version
1.01	04.07.03	Suf	Whole document described in more detail
1.02	07.07.03	Suf	max. length of parameters of SendData function included
1.03	08.07.03	Suf	Whole document described in more detail
1.04	18.07.03	ck	Colors of Figure 1 changed for better reading
1.05	29.07.03	Suf	- Adaption to changed API - Code fragments included - Chapter Global definitions included
1.06	30.07.03	Suf	Little improvements