

---

**NetCPlus Internet Solutions Inc.**  
**<http://www.netcplus.com/winsockapi.html>**  
**Sales : +1 727 391 8966**  
**Support : +1 727 391 5872**

*Code and library design Copyright : NetCPlus Internet Solutions, Inc 2004.*

---

## **The SMTPlib\_API library for including SMTP mail sending in your applications - without the tears !**

**INDEX GOES HERE...**

### **Developers Manual**

Thank you for trying the **SMTPlib\_API** library from NetCPlus. We are sure you will find it will allow you to quickly and easily implement the sending of email using the Internet standards SMTP protocol.

The SMTPlib\_API library provides support for both standard and Authenticated SMTP delivery, and is capable of sending one single message, or the contents of an entire folder, depending only on your application's needs !

NetCPlus are a professional software development organization of long standing that specialize in the development of TCP based applications, mostly for Intranet and Internet use, and have quite naturally developed these libraries over a period of years to avoid repetitive recoding of both the basic and advanced functionality required for TCP/IP functionality and the effective sending of mail using the SMTP protocol.

Although the library has been developed for developers and programmers who are NOT skilled or experienced TCP/IP programmers, it is still an equally useful tool in the armory of any developer who does not have any, some, or possibly even all of these TCP skills, but who still want a fast, easy way to enable the sending of mail via an SMTP interface for their own applications by using a set of well proven, bug-free, and full documented / supported, highly efficient SMTP and Winsock (TCP/IP) function calls, which is exactly what **SMTPlib\_API** from NetCPlus provides you with.

The **SMTPlib\_API** doesn't try to re-invent any wheels, so it uses our own well proven **WinSock\_API** library (the WinSock\_API.DLL is provided with this library) to handle all of the required Winsock handling. This is totally transparent to you, as all you need to do to have this powerful support is to ensure that your application can find WINSOCK\_API.DLL as well as SMTPLIB\_API.DLL and away you go.

All of the NetCPlus Developer libraries are written entirely in standard "C" for maximum platform portability and use ONLY the Microsoft Windows API. They are compiled and built using the Industry standard Microsoft Visual C 6.x. Due to this, and although unlikely, it is possible that the LIB files that are generated by the Microsoft compiler/linker may not be compatible with the Borland Compiler or some other "non Microsoft compatible" compilers. No low level assembler or other "non Window's API compatible" code is used in the libraries, making them usable across almost all versions of Microsoft Windows (NT 3.5 being the only exception).

Your applications can be written and compiled using virtually any Microsoft "C" compiler, and also from Visual Basic, Visual FoxPro and Delphi, providing only that you can create the correct **SMTPlib\_API** function declarations to suit those development environments. We have not included these other development environment declarations in this documentation due to possible variations in declaration standards required by those (diverse) environments. However, if you check our web site, you may find declaration files for Visual Basic are now available.

=====

**A SHORT (but hopefully useful) OVERVIEW OF EXACTLY WHAT THE SENDING OF SMTP MAIL is really all about.**

**We recommend that all developers using his library browse through this information even though the information it contains may already be known to you !**

The SMTP protocol is the (only) one that is used for sending email across the Internet and virtually all other forms of networks. SMTP is a TCP/IP protocol defined in the RFC821 and RFC822 documents.

However, everyone knows that the learning curve to be able to program SMTP (and therefore also handle TCP/IP) applications successfully is very steep ( in fact we can guarantee that !!! ), and many developers do not have these skills, or they do not have either the time or inclination to learn them to the level required.

To avoid you needing to do so is exactly why we have developed and released **SMTPlib\_API**.

**THE SMTP PROTOCOL**

SMTP is (virtually) the ONLY protocol used to transmit email across a network. It has been around for quite some time now, and has been proved to be a pretty reliable protocol over all. It allows any two machines to make a connection between each other irrespective of what operating system they may be running to send (RFC 822 compliant) mail messages between themselves. Once a connection is made, (through the use of what is called a SOCKET provided by the TCP/IP protocol stack (Winsock for Windows), those machines can start the protocol exchange needed to transmit all of the required message data and header information.

If you like, you can think of these SMTP protocol commands as a private code system between the machine that is trying to send an email, and the destination system, usually an SMTP server.

SMTP servers are by design totally passive, so it is (virtually) always up the Client application to initiate any SMTP connection. It is also normally left to the client to terminate that session whenever they are ready to do so, although the host may do so if it so chooses, or if it identifies an error in the protocol.

**THE SMTP PROTOCOL in a little detail**

Once the Winsock layer (handled entirely by our own **WinSock\_API**) has established a connection between your application and the receiving SMTP server, you can start sending mail immediately.

In the brief, but nevertheless complete session outline below,

--> is SMTPlib\_API's requests  
<-- represents the servers responses

```

-->[open TCP connection by connecting through an open socket]
<-- 220 hostname: BusinessMail SMTP server - Ready
-->HELO client.lan.net
<-- 250 hostname: hello client.lan.net
--> MAIL FROM: <anyuser@client.lan.net>
<-- 250 Sender OK.
--> RCPT TO: <recipient.email@xyz.domain.com>
<-- 250 Recipient OK
--> RCPT TO: <recipient.two.email@xyz.domain.com>
<-- 250 Recipient OK
.....May be multiple RCPT lines and 250 Recipient OK acknowledgments
--> DATA
<-- 354 Enter e-mail, end with "." on line.
--> Received: (from anyuser@localhost) by client.lan.net
--> (8.8.5/8.8.5) id VAA15646 Sun, 1 Mar 1998 21:20:37 -0800
--> Date: Sun, 1 Mar 1998 21:20:37 -0800
--> From: Any User <anyuser@client.lan.net>
--> Message-Id: <199803020520.VAA15646@netcplus.com>
--> To: end.recipient@emailaddress.com
-->
--> Hello this is a test
--> more testing

--> .
<-- 250 Message excepted
--> QUIT
<-- 221 Closing connection.

```

NB – As shown, there may often be multiple RCPT TO: lines (and therefore matching 250 responses) if you are sending this message to more than one email address.

The only other point worth noting here and now in this conversation is the line that contains a period on it's own. This is the way the SMTP protocol identifies that it has received ALL of the message data. This is in fact a CR/LF period CR/LF combination. All messages sent via SMTP must send this as the very last line of the message, but you do not have to worry about this as the SMTPlib\_API library handles it for you automatically.

That's really about all there is to sending an email message - in a nutshell.....

=====

## **LOGGING by the LIBRARY**

**SMTlib\_API** also logs all activity to a file named **smtpliblog.txt** that is saved in the default Windows \SYSTEM or \SYSTEM32 folder, depending on the version of Windows.

This log file shows all function calls made, with results and errors, and (by default) is automatically trimmed to 2MB whenever it exceeds 10MB in size.

This log file should always be your first point of reference to identify any errors caused by improper use of the library.

To allow you to customize this logging system, you can specify the path and filename of this log file by creating the following entry in the Windows registry :-

**HKLM\Software\netcplus\smtplib\Options**  
**REG\_SZ LogFile**

and set it's value to the fully qualified path and file name you want for the log file. If you only put

in a file name, it will automatically default to the standard Window's TEMP folder. If you do not use this entry, the log file will be name smtpliblog.txt and can be found in the Windows \SYSTEM32 folder of your machine.

You can also control the maximum and minimum size of the log by creating the following two keys

**HKLM\Software\netcpplus\smtplib\Options**  
REG\_DWORD **LogFileMax**  
REG\_DWORD **LogFileMin**

These must be set to a maximum value of at least 5 and a minimum value of 1 or the library will use the default settings of 10 and 2, which are the sizes used if you do not have these registry entries.. The values in these entries MUST BE in MEGABYTES.

Finally on the question of library logging, you can disable log file generation completely by creating the following key :-

**HKLM\Software\netcpplus\smtplib\Options**  
REG\_DWORD **UseLogFile**

And setting it's value to ZERO  
Setting it back to ONE will re-enable logging.

This can of course be used to allow you to debug a customer installation if the need arises !

A sample session log is shown below. In this example a mail message is sent to a single recipient via a specified relay server

A final and very useful debugging aid is available by creating the following registry entry :-

**HKLM\Software\netcpplus\smtplib\Options**  
REG\_DWORD **ShowWarningBoxes**

When set to ONE, the library will display a standard Window's MessageBox() whenever a failure in the library occurs, allowing you to quickly and easily identify and debug your application code when running in the debugger environment.

Setting this value back to ZERO will stop these MessageBoxes() from being displayed.  
The Default is ZERO (OFF)

**HKLM\Software\netcpplus\smtplib\Options**  
REG\_DWORD **CallBack**

When this entry is set to ONE, the library will send messages back to your application in real time using the defined callback function you declare in your application. (see later on in this manual for more information on this)

Setting this value back to ZERO will stop the library trying to send this information to your application.

2004/09/25 15:57:34 SMTPLibAPI [SMTPLIB\_API\_DEMO\_APP] - SESSION CLOSED.

A full log entry looks similar to above, with date/time (or ticks) SMTPLibAPI followed by the calling

application name, and then the message itself.

You can control what is included in each line as follows.

Setting this value to TWO will stop the library adding the "SMTPlibAPI" to the line  
Eg: 2004/09/25 15:57:34 [SMTPLIB\_API\_DEMO\_APP] - SESSION CLOSED.

Setting this value to THREE will stop the library adding both the "SMTPlibAPI" and the application name to the line.  
Eg: 2004/09/25 15:57:34 - SESSION CLOSED.

**HKLM\Software\netcpplus\smtpplib\Options**  
REG\_DWORD **MaxSendMsgs**

This setting tells the library the maximum number of messages it should ever send in a single connection or session. A value of 250 is set in the smtpplib\_api.h file using #define MAX\_MSGS 250, but this registry entry will override this if found.

**HKLM\Software\netcpplus\smtpplib\Options**  
REG\_DWORD **SendPercentSent**

If enabled, the callback function will return you the percentage of each message that has been sent in real time. This is very useful if you want to implement a progress bar to show your users how the send operation is progressing. This value is ALWAYS on the send progress of each individual message. If you want to show overall progress when sending multiple messages, you need to handle this in your own code.

The default for this is OFF.

=====

**Documentation standards:**

All **SMTPLIB** Function calls are in **BOLD**.

ALL **SMTPLIB** structure variables mentioned in descriptive text are in **BOLD**.

ALL comments in code examples are *italicized*.

All function parameters show the direction it is used as IN - IN/OUT - or OUT -

=====

**GETTING STARTED**

**HEADER FILES**

Before you are able to take advantage of the power in the NetCPlus **SMTPLIB** library, you need to have the following #include statement in all source files that make a call to the library, (or in your main header file).

**#include "smtpplib\_api.h"**

**LIB FILES**

Although **SMTPIib\_API** does of course have to use the Windows Winsock when sending mail, this is handled completely by the **WINSOCK\_API.DLL** that is provided with the **SMTPIib\_API** system, but you will also need to specify that WSOCK32.LIB is included in your linker command line.

(This LIB may vary - check your windows documentation.) However, this is typically the correct name for the Winsock library for both Winsock 1.1 and Winsock 2. (NB The Winsock library currently only requires Winsock V1.1, and does not use any functionality provided by the newer Winsock 2, but Winsock V2 is also guaranteed by Microsoft to be completely backwards compatible to v1.1.

You also need to add the **smtplib\_api.lib** to your linker line, as this exports all of the **SMTPIib\_API** and a required subset of the **WinSock\_API** functions and makes them available to your application.

## **THE SMTPIib\_API LIBRARY**

**IMPORTANT** - You do not have to do all the following just to test the **SMTPIib\_API** library, but it is the recommended way to install and access it once you ship your SMTP enabled application publicly. For testing, you can put **smtplib\_api.dll** and **winsock\_api.lib** in your own applications folder and they will work just fine....

However, we strongly recommend that you install this (as recommended by Microsoft for all DLL's) in a folder similar to that shown below.

In fact the installer will have placed copies of the DLL in a folder named NETCPLUS for you in addition to the files in the installation folders \DLL folder....(this can be changed later by simply renaming it to whatever "Your\_app\_name" you choose)

x:\program files\common files\Your\_app\_name

and then all you need to do is have your application install program set up a registry entry for your application that points to this folder.

The Installer will have also created matching Registry keys (for NETCPLUS) for the items shown below.

The registry key we use for all our own applications is as shown below :-  
( HKLM stands for HKEY\_LOCAL\_MACHINE ! )

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\application\_name.exe  
and the key value entry is a REG\_SZ named

Path

which should contain as its value :-

x:\PROGRAM FILES\COMMON FILES\NetCPlus

or whatever folder name you choose if you would rather replace NetCPlus with your own corporate name !!

## **SMTPLIB HEADER FILE INFORMATION**

The **SMTPIib\_API** library header file we provide includes the following declarations that you

need to be familiar with as you start using **SMTPlib\_API** :-

export type declaration ( just for completeness !)

**#define SMTPLIB\_API \_\_declspec(dllexport)**

//This is the **SMTPLIB\_SEND STRUCTURE** that is used to pass all information to the **SMTPlib\_API** library.

NB – This structure is very highly documented in the `smtlib_api.h` file

typedef struct

```
{  
  
    //SMTP host server (optional) - ONLY COMPLETE if you want mail sent to it directly!!!!  
    // IN - may be empty if sending MX, else it is the **name** of the Host SMTP server  
    char    szSMTPServerName[256];  
  
    //SMTP host server (optional) - ONLY COMPLETE if you have a valid IP ADDRESS and  
    //want mail sent to it directly!!!!  
  
    // if not, the library will resolve the hostname entered above, and fill this field with the  
    correct IP Address  
  
    // IN - may be empty if sending MX, else it MAY be the **IP ADDRESS** of the Host  
    SMTP server you want to connect to?  
    char    szSMTPServerIPAddress[16];  
  
    // OUT - filled by winsock getHostByName() if sending MX, else it is empty  
    char    szDNSAddresses[8][16];  
  
    // OUT - filled by winsock if more than one IP address is returned by DNS lookup  
    int      iDNSEntries;  
  
    // IN - only used for MX sending normally, but it should be available for the HELO or  
    EHLO SMTP protocol as well as some SMTP servers demand it  
    char    szSenderDomain[256];  
  
    // IN - The port to use for connections (default is 25) if left at ZERO, library defaults to 25  
    int      Port;  
  
    //Fully qualified path to message(s) to be sent  
    // IN - FQ path and filename of the message to be sent  
    char    szMessage[MAX_PATH];  
  
    // IN - Specify the message file suffixes to be checked for if searching a folder - default is  
    TXT  
    char    szFileType[12];  
  
    // IN - Set this to TRUE if you want the library to delete the message(s) files after they  
    have been sent (if success ONLY)  
    BOOL    bDeleteAfterSend;  
  
    //Senders email address
```

// IN - MUST be COMPLETED, UNLESS you set the pSend->bUseEnvelope flag below  
as the sender address is then retrieved from that messages envelope file

**char szSenderAddress[256];**

//Authentication details (ONLY if required by receiving server)

//It is strongly recommended you only complete these

//if you are sure the SMTP server you are sending to

//actually requires SMTP authentication.

//IMPORTANT - This version of the SMTPlib\_API library ONLY

//provides support for the most commonly used authentication,

//which is known as AUTH LOGIN PLAIN, and which sends the

//authentication data in mime encoded format to the

//receiving server

**char szUserName[128];** // IN -> Authentication login name

**char szUserPassword[128];** // IN -> Authentication password

//Specify how you want the library to identify recipient(s) addresses

// IN - if you want library to read ENV file to obtain both MAIL FROM and RCPT TO  
address(s)

**BOOL bUseEnvelope;**

// IN - if you want library to use SINGLE email address in pSend->szRecipientAddress

**BOOL bUseAddress;**

//If you want override a recipient's address, fill this out

// IN - This is the single recipient address that is used if

**char szRecipientAddress[256];**

// IN - if you want library to use email address(s) in pSend->hRecipients

**BOOL bUseList;**

//sending to multiple recipients

// IN - HANDLE - if the above is checked,you pass a global memory handle to the list of  
recipients in here

**HGLOBAL hRecipients;**

// INTERNAL USE ONLY - used for the above handle, DO NOT USE

**LPSTR pRecipients;**

// OUT - library fills this out - after parsing list of addresses

**Int iRecipientscount;**

//if you wish, and you have an email message already in memory

//you can pass the handle to this message to the library so it does not have to read it in  
again.

//CAUTION - You must always allocate memory using GLOBALALLOC and use GHND or  
GMEM\_ZEROINIT | GMEM\_MOVEABLE so that the library can append the termination  
protocol data to the end of your data in the buffer.

// IN - \*\*UNLOCKED\*\* GLOBAL Memory handle to email message in memory

**HGLOBAL hMessageHandle;**

```

// IN/OUT Pointer to be used with the memory you have allocated in the above field
LPSTR pMessageHandle;
//General use
// IN - if sending multiple messages, the total messages you want sent - ZERO means
ALL msgs in the folder
LONG IMessagesTotal;
// OUT - return value of how many got sent
LONG IMessagesSent;
// OUT - # of messages that could not be sent
LONG IMessagesFailed;
// OUT - Error code if a failure occurred
int iErrorCode;
// OUT – Any Winsock Error code returned by a winsock call
int iWinsockError;
// OUT – The active socket in use
SOCKET iSfd;
}

```

#### **SMTPLIB\_SEND;**

Library standard return values

```

#define SMTPLIB_OK 0 //Success
#define SMTPLIB_FAIL -1 //Failure

```

We strongly recommend your implementation code tests for these values as returned by the library eg:

```

result = SMTPLIB_function_call()
if(result == SMTPLIB_FAIL)
{
    //handle error by checking the value in iErrorCode in the SMTPLIB_SEND STRUCTURE
    //shown above. (There may also be a winsock error code in iWinsockError)
}

```

There are many other detailed error codes #defined by smtpplib\_api.h that the library will fill the pSend->iErrorCode variable with on return so that you can identify the cause of the problem. We recommend you familiarize yourself with these and check for them in you own code.

The following important structure is also declared for you in smtpplib\_api.h.

You need to declare an instance of this as a global variable in your application, plus we suggest you also declare a pointer to it, and then use this pointer (the address of the structure) as all **SMTPIib\_API** functions expect this pointer as their first parameter.

```

typedef struct {
    char          szWindowClass[128];
    char          szSMTPLibVersion[24];
    char          szTCPLibVersion[24];
}

```

```

char        szSMTPLibDate[8];
char        szTCPLibDate[8];
HWND        hWndCallBack;
DWORD       dwStructureVersion;
DWORD       dwSMTPStructVersion;
DWORD       dwTCPStructVersion;
short       iInitialized;
}SMTPLIB_SESSION;

```

**YOU DEFINITELY MUST NOT CHANGE THIS STRUCTURE IN ANY WAY.**

Its contents are updated by **SMTPlib\_API** to keep a record of the calling application. YOU SHOULD NOT CHANGE the values of ANY these structure members or calls to the library may well fail.....

WE SUGGEST YOU DECLARE THESE IN YOUR GLOBALS header file as shown below :-

```

SMTPLIB_SESSION SMTPSession,
                * pSession;

```

**Structures used by MX record DNS lookups**

**IMPORTANT** – these structures are declared in the **Winsock\_API.h** file, not in the **SMTPLIB\_API.h** file

```

typedef struct
{
    char szHost[128];
    char IPAddresses[12][24];
    int iPreference;
    int iTotalIP;
    int iTotalValid;
    BOOL bValid[12];
}MXHOSTS;

```

```

typedef struct
{
    int iTotalResponses;
    MXHOSTS szMXHosts[20];
}MXADDRESSES;

```

These are only used when MX record resolution is required.

## **Description of available Library functions**

```

SMTPLIB_Initialize_Library(
    IN – SMTPLIB_SEND *pSession,
    IN – HWND hWnd,);
    IN – char * szWindowClass);

```

## IMPLEMENTATION NOTES

### STARTING UP

All you need to do is to make the following call in your WinMain function. This initializes the **SMTPlib\_API** library correctly for further use by your application.

```
VOID SMTPLIB_Initialize_Library(  
    IN *pSession,  
    IN hWnd);  
    IN - "NETCFAXCLIENT");
```

SMTPLIB\_SEND \*pSession is a pointer to your globally declared SMTPlib\_API session.

HWND hWnd is your calling window's handle, or a NULL pointer may be passed down by casting it as - (HWND)NULL.

char \* szWindowClass is the (unique) name of your application (we recommend you do not use spaces, and you must restrict it to less than 128 characters). This allows the library logging to report back which of your applications is using or doing what....

The "NCFAXCLIENT" in our example should of course be your application name. You should try to ensure that this name is **likely to be totally unique** on any machine using the library.

**This MUST always be the first library call made, before any other calls, or they will certainly fail !**

THAT'S ALL YOU NEED TO BE ABLE TO START TO USE THE **SMTPlib\_API**.....

---

### USING THE LIBRARY

.....

### CLOSING DOWN YOUR APPLICATION

When your application is terminating, we recommend you should add something similar to the code shown below.

*//You MUST Close down your session with the **SMTPlib\_API** library before terminating your program, so we recommend this call is placed after the main windows message loop, or in your WM\_DESTROY code*

```
VOID SMTPLIB_Closedown_Library(SMTPLIB_SESSION * pSession, "NCFAXCLIENT");
```

---

```
int SMTPLIB_Initialize_Structure(SMTPLIB_SESSION * pSession,  
    SMTPLIB_SEND * smtp_structure);
```

Before your application can use the SMTPLIB\_SEND structure, it should always call this function to ensure it has been correctly initialized.

---

```
int SMTPLIB_Send_Message(SMTPLIB_SESSION * pSession,
                        SMTPLIB_SEND * pSend,
                        FPN fCallback);
```

This is probably the most important, and powerful function provided in the **SMTPlib\_API** library. It handles ALL of the connecting to a server, sending messages and cleaning up after itself, and reports back to your application via the callback function as it does so.

All you need to do is to fill out the SMTPLIB\_SEND structure with the relevant information and then call this function and presto – your mail will be sent by **SMTPlib\_API**.

---

```
int __stdcall SMTPLIB_Perform_DNS_lookup(SMTPLIB_SESSION * pSession,
                                        SMTPLIB_SEND * pSend);
```

This is a utility function that lets you check to see if a server host name (or even an IP address) is valid before trying to send email to it. It automatically performs both forward and reverse DNS lookups, depending on which field in the SMTPLIB\_SEND structure you complete before making this call.

---

```
//Parses an ENV file and fills out the relevant pSend structure fields
int __stdcall SMTPLIB_Get_Message_Details(SMTPLIB_SEND * pSend);
```

---

```
//get error information string from error code
VOID __stdcall SMTPLIB_Get_Error_Details(SMTPLIB_SEND * pSend);
```

---

```
//make DNS lookup for MX record to get SMTP host
int __stdcall SMTPLIB_Get_Mx_Servers(char * domain,
                                    VOID * IPAddresses,
                                    char * IPAddress,
                                    BOOL bForce);
```

---

```
//return the currently specified DNS server(s) on the Network Interface card
//of this machine. The buffer needs to be 5 * 20 bytes (100) to handle up
//to 5 possible DNS entries
```

```
VOID __stdcall SMTPLIB_Get_Default_Dns_Server(char * DNS);
```

---

## **SMTPLIB EXAMPLE CODE**

Using the SMTPlib\_API LIBRARY in real world applications

## **SUMMARY EXAMPLES**

(See further down for a fully documented and working source code example)

These show only a single comment line above each line of code to give you an idea of what they each mean. They also illustrate clearly how easy it is to use the **SMTPlib\_API** library to send emails of any type and number to a Relay Server or direct to a recipient using MX records.

### **OVERVIEW EXAMPLE 1**

This code section illustrates sending all files in a specified folder to one or more recipients.

In this example we are sending these files to a specified relay server, and as we have provided matching ENV files in the same folder, the **SMTPlib\_API** library can obtain all of the sender and recipient(s) addresses automatically for each message before it sends them.

```
//Set Relay server name
strcpy(pSend->szSMTPServerName, "EMACHINE");
//Local Relay server IP address (optional but used if specified instead of name)
strcpy(pSend->szSMTPServerIPAddress, "192,168,4,55");

//tell library we have provided ENV files
pSend->bUseEnvelope = TRUE;

//Set folder for messages
strcpy(pSend->szMessage, "C:\\ss4 test system\\outgoing");

//Set SMTP port (default is always 25)
pSend->Port = 25;
//set senders domain name
strcpy(pSend->szSenderDomain, "NETCPLUS.COM");
//specify message file suffix (only if necessary)
strcpy(pSend->szFileType, "TXT");
```

### **OVERVIEW EXAMPLE 2**

This code section illustrates sending a single file (named myresumee\_email.txt) by specifying the full path and filename to a single specified recipient in the pSend-> szRecipientAddress field.

We are also sending this via a specified relay server, but we have not provided a matching ENV file, so have to fill out the relevant fields in the pSend structure;

```
//Relay server name
strcpy(pSend->szSMTPServerName, "EMACHINE");

//Local Relay server IP address (optional but used if specified instead of name)
strcpy(pSend->szSMTPServerIPAddress, "192.168.4.55");

//set senders email address
strcpy(pSend->szSenderAddress, "iant@netcplus.com");

//tell library we are using the single recipient address field
pSend->bUseAddress = TRUE;
//set single recipients address
strcpy(pSend-> szRecipientAddress, "myparents@home.com");

//set FQ path and filename for message
```

```
strcpy(pSend->szMessage, "C:\\ss4 test system\\outgoing\\myresumee_email.txt");
```

```
//Set SMTP port (default is always 25)
```

```
pSend->Port = 25;
```

```
//set senders domain name
```

```
strcpy(pSend->szSenderDomain, "NETCPLUS.COM");
```

```
//specify message file suffix (only if necessary)
```

```
strcpy(pSend->szFileType, "TXT");
```

### **OVERVIEW EXAMPLE 3**

This code section illustrates sending a single file (named myresumee\_email.txt) by specifying the full path and filename, but this time to multiple recipients, and as we have not provided an ENV file we need to use the global memory pointer handle pSend->hRecipients to hold these and of course allocate some memory to hold them.

PLEASE DON'T FORGET TO UNLOCK THE MEMORY \*\*BEFORE\*\* CALLING THE LIBRARY

Again we are sending this via a specified relay server, but if we wanted to make this an example of sending via MX, all we would need to do is NOT SPECIFY pSend->szSMTPServerName. All other settings would be EXACTLY the same.

```
//Relay server name
```

```
strcpy(pSend->szSMTPServerName, "EMACHINE");
```

```
//Local Relay server IP address (optional but used if specified instead of name)
```

```
strcpy(pSend->szSMTPServerIPAddress, "192.168.4.55");
```

```
//set senders email address
```

```
strcpy(pSend->szSenderAddress, "iant@netcplus.com");
```

```
//tell library we are using the multiple recipients global memory address data
```

```
pSend->bUseList = TRUE;
```

```
//set up the global memory list of recipients
```

```
pSend->hRecipients = GlobalAlloc(GHND, 1024);
```

```
pSend->pRecipients = (LPSTR)GlobalLock(pSend->hRecipients);
```

```
strcpy(pSend->pRecipients, "<olwent@netcplus.com> <garyw@netcplus.com>");
```

```
GlobalUnlock(pSend->hRecipients);
```

```
//set FQ path and filename for message
```

```
strcpy(pSend->szMessage, "C:\\ss4 test system\\outgoing\\myresumee_email.txt");
```

```
//Set SMTP port (default is always 25)
```

```
pSend->Port = 25;
```

```
//set senders domain name
```

```
strcpy(pSend->szSenderDomain, "NETCPLUS.COM");
```

```
//specify message file suffix (only if necessary)
```

```
strcpy(pSend->szFileType, "TXT");
```

### **Ok, so now, here is A REAL WORLD (fully documented) EXAMPLE**

To show you just how easy it really is to use SMTPlib\_API to send emails, the following is a (VERY HIGHLY

COMMENTED) example of the code in our own networked NetCFax Fax client application that has an option to send emails, and of course, this uses the **SMTPlib\_API** library.

This entire code is repeated under this without any commenting to show you how little code is really required to send an email message across your local network., or even across the world.

As you can see, all SMTP code is handled by SMTPlib\_API (naturally), and the total source code needed to perform what is a very good and working example of classic SMTP functionality amounts to less than 70 lines of real code, plus of course our (fairly copious) commenting. This particular code fragment is **ACTUALLY EXACTLY AS USED in our Fax client**, and it works very well to send mail using a relay server or via MX to the end users address directly.

The code required to do this same functionality manually would probably amount to 1000+ lines of detailed, and very bug prone code, and that would of course be very much more complex, requiring a large number of low level SMTP calls and data manipulation, plus a lot of global memory allocation and ReAllocation to allocate the memory required to store the message data that is to be sent by the library

#### INTERNAL FUNCTION DECLARATIONS

```
=====
VOID smtpplib_send_message_thread(VOID * args);
=====
```

```
FPN __stdcall sendmail_callback(VOID * pSendStructure, int iCallback, char * Msg, int iErrorCode)
/*
The above is YOUR APPLICATIONS declaration of the library callback function that is defined in
SMTPLIB.H
```

It is declared in SMTPLIB.H as :-

```
typedef (__stdcall *FPN) (VOID * pSend, int iCallback, char * Msg, int iErrorCode);
```

Progress and error information can be returned to your application in real time using this callback functionality, although you do not have to use it.

```
As you can see this callback function returns a VOID * to a complete SMTPLIB_SEND STRUCTURE with
relevant information you might need in it, plus the message type identifier (iCallback), a string containing
information, and an error code value that is ZERO if it is simply an informational progress message
*/
```

```
/*
We have chosen to implement the sending of mail as separate thread
This is only done to avoid the main application being blocked when
the SMTP library uses a blocking call to try to connect() to the
remote server
```

```
It is the way we recommend you should handle this !!
*/
```

START OF ACTUAL CODE EXAMPLE :-  
(This source code is provided for you in the file smtpcall.cpp in the \SOURCE folder of the **SMTPlib\_API** installation)

**For illustration, we have made all lines of actual code in BOLD**

```
/*
=====
Function called by your main application code
This function just spawns the new thread shown below
```

```
*/
int faxsmtp_send_email(HWND hWnd, int iSendmode)
```

```

{
    THREADARGS ThreadArgs;
    memset(&ThreadArgs, 0, sizeof(THREADARGS));
    //debug only
    iSendmode = 0;

    sprintf(ThreadArgs.arglist, "%ld,%d", hWnd, iSendmode);
    _beginthread(smtpplib_send_message_thread, 0, (VOID*)&ThreadArgs);
    do {
        Sleep(10);
    }while (!ThreadArgs.bThreadLoaded);
    return WINSOCK_OK;
}

```

```
/*
```

```

=====
THIS IS THE THREAD ITSELF
THAT SETS UP THE SEND
STRUCTURE AND CALLS THE
SEND FUNCTION IN THE LIBRARY

```

This is where all the real work in terms of setting up the structures to tell the **SMTPlib\_API** library what you want it to do

```
*/
```

```

//=====
VOID smtpplib_send_message_thread(VOID * args)
//=====

```

```
{
```

```

    char msg[256], *p1, *p2;
    //Local function declarations

```

```

    THREADARGS ThreadArgs, *pThreadArgs; // required for thread alone
    HWND hWnd; //probably your parent window
    SMTPLIB_SEND smtpsend, *pSend; //The main SMTPlib_API structure
    SMTPLIB_SESSION Session; // The main SMTPlib_API session structure
    int result;
    int iSendmode;

```

```

//=====
//handle safe thread startup
//=====
pThreadArgs = (THREADARGS *)args;
ThreadArgs = *pThreadArgs;
pThreadArgs->bThreadLoaded = TRUE;
//=====

```

```

//=====
//Get the arguments passed to thread.
//=====
p1 = pThreadArgs->arglist;
p2 = strchr(p1, ',');
lstrcpyn(msg, p1, (p2-p1)+1);
hWnd = (HWND)atoi(msg);
p1 = ++p2;
strcpy(msg, p1);
iSendmode = (int)atoi(msg);
//=====

```

```
/*
```

```

=====
Ok, now were all done with safe thread handling

```

we can get on with the real SMTP work...

```
=====
AS WE HAVE JUST DECLARED THIS STRUCTURE ABOVE
and as YOU CANNOT ASSUME THAT MICROSOFT will provide clean structures we need to
ensure it is clean ourselves before sending it to the SMTPLIB library for initialization
*/
```

```
//=====
memset((VOID*)&Session, 0, sizeof(SMTPLIB_SESSION));
```

```
/*
```

```
=====
initialize SMTP Library
THIS MUST BE DONE BEFORE ANY OTHER SMTPLIB calls are made
```

---

```
*/
```

```
SMTPLIB_Initialize_Library(&Session, hWnd, "NCFAXCLIENT");
```

```
//initialize the new SMTP structure for the same reasons as above
```

```
memset((VOID*)&smtpsend, 0, sizeof(SMTPLIB_SEND));
SMTPLIB_Initialize_structure(&Session, &smtpsend);
```

```
//assign our pointer to SMTPLIB_SEND structure
pSend = &smtpsend;
```

```
/*
```

```
=====
Now we can initialize SMTPLIB_SEND
structure to tell it the library what
tasks we want it to do...
```

```
=====
This if() test is only provided to illustrate
the different ways you can fill out the
SMTPLIB_SEND for the different modes of
message handling provided by the library
*/
```

```
if(iSendmode == 0)
```

```
{
```

```
    //Do this if you are sending direct using MX (No relay server)
    //SMTP SERVER name (Leave blank for MX sending)
```

```
    strcpy(pSend->szSMTPServerName, "EMACHINE");
```

```
    //Use ENV file provided to obtain sender and recipient(s) addresses
    //ENV file must have the SAME "root" name as the message, and end with .ENV
```

```
    pSend->bUseEnvelope = TRUE;
```

```
    //Clear messages total = Send all if using path only option
    //if you are sending multiple messages, but only want the library
    //to send x you can set this to maximum number here
```

```
    pSend->IMessagesTotal = 0;
```

```
    //We are NOT specifying a message file to send
    //All existing .TXT messages in this folder will be sent
```

```
    strcpy(pSend->szMessage, "C:\\ss4 test system\\outgoing");
```

```
}
```

```
else
```

```

{
    /*
    =====
    Initialize SMTPLIB_SEND structure
    so we can send message(s)
    This is an example of a SINGLE MESSAGE
    going to multiple recipients via a
    specified Relay server
    =====

    SMTP SERVER name (Leave blank for MX sending)
    if this is filled out, OR the pSend->SMTPServerIPAddress
    is filled out, the message(s) will be sent to that server
    */

    strcpy(pSend->szSMTPServerName, "");

    //We clear this field to force library to resolve the server name above
    //In other words, to do a DNS lookup

    strcpy(pSend->szSMTPServerIPAddress, "");

    //We are Sending to more than one recipient

    pSend->bUseList = TRUE;

    //we have chosen to provide message senders email address
    //and we are also providing multiple recipient addresses in
    //the hglobal pointer pSend->hRecipients

    strcpy(pSend->szSenderAddress, "iant@netcplus.com");

    //We are only sending one message
    //This is a fully qualified path and name of the message file to send.
    //If it includes a file name, just that one will be sent
    //but if it only contains a path, all files in that path will be sent

    strcpy(pSend->szMessage, "C:\\ss4 test system\\outgoing\\smtpstest.TXT");

    //We have chosen to specify a list of recipients
    //so we must now create the recipients list in global memory

    //get sufficient global memory.
    pSend->hRecipients = GlobalAlloc(GHND, 1024);

    //lock it so we can add our address list
    pSend->pRecipients = (LPSTR)GlobalLock(pSend->hRecipients);
    strcpy(pSend->pRecipients, "<olwent@netcplus.com> <garyw@netcplus.com>");

    /**YOU MUST ALWAYS UNLOCK** memory again
    GlobalUnlock(pSend->hRecipients);
}

/*
=====
GENERAL SETTINGS
=====
*/
//Port to send message on
pSend->Port = 25;

//The sending domain name - should always be filled out

```

```

//for completeness of SMTP protocol
strcpy(pSend->szSenderDomain, "NETCPLUS.COM");

//optional message file suffix setting
strcpy(pSend->szFileType, "TXT");

/*
=====
The following SMTPLIB library function checks the pSend->szMessage
field to see if it contains a period followed by valid file suffix

The file suffix can be specified in the SMTLIB_SEND structure
pSend->szFileType field, AS IS SHOWN ABOVE, or it can be set to a
default SZ_STRING value in the registry key
HKLM\SOFTWARE\NetCPlus\WinsockAPI\Options
SZ_STRING MsgExtension set to TXT or whatever you want.

The suffix MUST NOT EXCEED 10 characters in length, and you must NOT
include the leading period (eg: TXT or XYZ is fine, but .TXT will fail)

If as this example shows, you specify the suffix, all other settings
are ignored. If you don't, the registry setting, if any is used, and
if no entry exists in the registry either, then TXT is assumed.

If NO filename is included in the pSend->szMessage field, it assumes we
have a SINGLE user specified message file, and no further address
extractions are performed.
=====
*/

if(SMTPLIB_Get_Message_Details(pSend) == SMTPLIB_FAIL)
{
    // NB pSend->iErrorCode will have been set by the call to the function above
    return; //failed to get the relevant message based information
}

//=====
//ALL Ok, we can proceed to send the message
//=====

//If you cannot, or do not wish to use the callback function
//provided you only need to pass a NULL FPN pointer to the
//library but it must be passed as shown in the line below

//EXAMPLE OF CALL TO LIBRARY WITH NULL CALLBACK POINTER ONLY
//    result = SMTPLIB_send_message(&Session, pSend, (FPN)NULL);

//we want callback information, so we are passing
// a pointer to our callback function to the library

result = SMTPLIB_Send_Message(&Session, pSend, (FPN)sendmail_callback);

//Did it succeed ?
if(result == SMTPLIB_FAIL)
{
    //Hmm - something went wrong
    //check the error code in pSend->iErrorCode to find out what
    //and handle it gracefully
    logfile_write_log("Failed to send message(s)");
}

//Ensure we free multiple recipients memory if needed.

```

```

//If necessary and we had a problem, we can verify these
//ourselves as the library does not remove them

if(pSend->hRecipients != (HGLOBAL)NULL){
    GlobalUnlock(pSend->hRecipients);
    GlobalFree(pSend->hRecipients);
    pSend->hRecipients = (HGLOBAL)NULL;//clear it so we know they are gone
}

//close the SMTPLIB libray down for graceful completion
SMTPLIB_Closedown_Library(&Session, "NCFAXCLIENT");

    _endthread();
    return;
}

```

THAT'S ALL THERE IS TO IT.....

Not very difficult at all for any reasonable C programmer really ??

### Source code (only) version (no commenting)

```

VOID smtpplib_send_message_thread(VOID * args);
FPN sendmail_callback(VOID * pSendStructure, int iCallback, char * Msg, int iErrorCode);

```

```

//This is our publicly called function that spawns the real working thread
int faxsmtp_send_email(HWND hWnd, int iSendmode)
{
    THREADARGS ThreadArgs;
    memset(&ThreadArgs, 0, sizeof(THREADARGS));
    //debug only
    iSendmode = 0;

    sprintf(ThreadArgs.arglist, "%ld,%d", hWnd, iSendmode);
    _beginthread(smtpplib_send_message_thread, 0, (VOID*)&ThreadArgs);
    do {
        Sleep(10);
    }while (!ThreadArgs.bThreadLoaded);
    return WINSOCK_OK;
}

```

```

//This is the thread itself
VOID smtpplib_send_message_thread(VOID * args)
//=====
{
    //LOCAL DECLARATIONS
    char msg[256], *p1, *p2;
    THREADARGS ThreadArgs, *pThreadArgs;
    HWND hWnd;
    SMTPLIB_SEND smtpsend, *pSend;
    SMTPLIB_SESSION Session;
    int result;
    int iSendmode;

    //CODE START
    //Handle thread safe operation an assign variables received
    pThreadArgs = (THREADARGS *)args;
    ThreadArgs = *pThreadArgs;
    pThreadArgs->bThreadLoaded = TRUE;

```

```

p1 = pThreadArgs->arglist;
p2 = strchr(p1, ',');
lstrcpyn(msg, p1, (p2-p1)+1);
hWnd = (HWND)atoi(msg);
p1 = ++p2;
strcpy(msg, p1);
iSendmode = (int)atoi(msg);

//ACTUAL PROCESSING CODE
memset((VOID*)&Session, 0, sizeof(SMTPLIB_SESSION));
SMTPLIB_Initialize_Library(&Session, hWnd, "NCFAXCLIENT");
memset((VOID*)&smtpsend, 0, sizeof(SMTPLIB_SEND));
SMTPLIB_Initialize_structure(&Session, &smtpsend);
pSend = &smtpsend;

if(iSendmode == 0)
{
    strcpy(pSend->szSMTPServerName, "EMACHINE");
    pSend->bUseEnvelope = TRUE;
    pSend->IMessagesTotal = 0;
    strcpy(pSend->szMessage, "C:\\ss4 test system\\outgoing");
}
else
{
    strcpy(pSend->szSMTPServerName, "");
    strcpy(pSend->szSMTPServerIPAddress, "");
    pSend->bUseList = TRUE;
    strcpy(pSend->szSenderAddress, "iant@netcplus.com");
    strcpy(pSend->szMessage, "C:\\ss4 test system\\outgoing\\smtpstest.TXT");
    pSend->hRecipients = GlobalAlloc(GHND, 1024);
    pSend->pRecipients = (LPSTR)GlobalLock(pSend->hRecipients);
    strcpy(pSend->pRecipients, "<olwent@netcplus.com> <garyw@netcplus.com>");
    GlobalUnlock(pSend->hRecipients);
}

pSend->Port = 25;
strcpy(pSend->szSenderDomain, "NETCPLUS.COM");
strcpy(pSend->szFileType, "TXT");

if(SMTPLIB_Get_Message_Details(pSend) == SMTPLIB_FAIL)
    return;

result = SMTPLIB_Send_Message(&Session, pSend, (FPN)sendmail_callback);

if(result == SMTPLIB_FAIL)
    logfile_write_log("Failed to send message(s)");

if(pSend->hRecipients != (HGLOBAL)NULL){
    GlobalUnlock(pSend->hRecipients);
    GlobalFree(pSend->hRecipients);
    pSend->hRecipients = (HGLOBAL)NULL;//clear it so we know they are gone
}
SMTPLIB_Closedown_Library(&Session, "NCFAXCLIENT");
_endthread();
return;
}

```

## LICENSING the SMTPlib\_API LIBRARY

## **EVALUATION VERSIONS**

The trial version is provided for you to test out the full functionality provided by the library in your own program(s), but the evaluation period is limited to a maximum period of 15 days from the date of first installation of the library.

Once the evaluation period expires, the library will no longer allow you to use its functionality.

Once you register however, “no further downloads are required”, and the library will then function normally again, and the “nag screens” you will notice during the evaluation period will also not be displayed.

## **REGISTERING YOUR COPY**

The only part of the **SMTPIib\_API** library that you need to distribute with your own applications is the `smtpplib_api.dll` file. You do not need to ship either the `smtpplib_api.h` or `smtpplib_api.lib` files, in fact, PLEASE NOTE that you are expressly forbidden from distributing those files to any third party under the terms of the End User License Agreement (EULA) that you accepted when first installing **SMTPIib\_API**.

**To register your copy, phone us at +1 727 391 8966, fax to +1 727 392 3216, or email to [winsocsales@netcplus.com](mailto:winsocsales@netcplus.com).**

## **SOURCE CODE LICENSING**

If you are interested in licensing the source code for the library, we are willing to make this available, and the cost of such a license will be discussed when you contact us.

## **TECHNICAL SUPPORT**

NetCPlus offer totally free support to evaluation users, and all registered users of the **SMTPIib\_API** library for a period of three months after the date you register it. Extended support options are of course available, with full details being provided on our web site – <http://www.netcplus.com> - just use the Support button on the left hand side and follow the links provided.