SMTP/POP3/IMAP Email Engine

Reference Library

(SEE_REF)

Version 8.3

March 21, 2022

This software is provided as-is.
There are no warranties, expressed or implied.

MarshallSoft Computing, Inc.
Huntsville AL 35815 USA

Email: info@marshallsoft.com
Web: www.marshallsoft.com

TABLE OF CONTENTS

TABLE OF CONTENTS - continued

1. Introduction

The **SMTP/POP3/IMAP Email Engine Library (SEE)** is a developer toolkit that provides a simple interface to quickly develop SMTP, POP3/IMAP mail applications and can be used with any program capable of calling the Windows API.

The **SMTP/POP3/IMAP Email Engine** (SEE) is a component library of functions providing easy control of the SMTP (Simple Mail Transport Protocol), POP3 (Post Office 3), and IMAP 4 (Internet Message Access Protocol) protocols.

A simple interface allows sending and receiving mail, including multiple MIME base64 and quoted-printable encoded attachments from within an application. Knowledge of Winsock and TCP/IP is not needed.

We have versions of the **SMTP/POP3/IMAP Email Engine** for C/C++ (SEE4C), Delphi (SEE4D), Visual Basic (SEE4VB), PowerBASIC (SEE4PB), Visual FoxPro (SEE4FP), Visual dBase (SEE4DB), Alaska Xbase++ (SEE4XB), and COBOL (SEE4CB). All versions of **SEE** use the same DLLs (SEE32.DLL and SEE64.DLL) and can be called from any program or compiler that can call the Windows API.

The latest version of our SMTP/POP3/IMAP Email component software and complete technical documentation can be found online at

http://www.marshallsoft.com/email-component-library.htm

This **SMTP/POP3/IMAP Email Reference Manual** (SEE_REF) contains details on each individual **SEE** function**.**

1.1 General Remarks

All functions return an integer code. Negative values are always errors. Refer to Section 3.0 below, "SEE Error Return Code List"**)**. The file **seeErrors.txt** contains a list of all error codes and their corresponding numeric value.

Non-negative return codes are never errors. Note that the **seeErrorText** function is used to get the text message associated with any error code.

Each function argument is marked as:

```
(I) : 4-byte integer.
(L) : 4-byte integer.
(P) : 4-byte pointer.
```

Refer to the declaration files (see Section 1.3 below) for the exact syntax of each **SEE** function. Also note that the example programs show exactly how **SEE** functions are called.

1.2 SEE Arguments

Only variables (previously declared in your program) should be passed to SEE functions, as is demonstrated in the example programs.

Text strings passed to SEE functions must be terminated by a null character since SEE calls Windows API functions that require null terminated strings.

1.3 Documentation Set

The complete set of documentation consists of three manuals in Adobe PDF format. This is the third manual (SEE_REF.PDF) in the set.

- SEE_4x Programmer's Manual (SEE_4x.PDF)
- SEE User's Manual (SEE_USR.PDF)
- SEE Reference Manual (SEE_REF.PDF)

The **SMTP/POP3/IMAP Programmer's Manual** (SEE_4x.PDF) is the programming language dependent manual and provides information needed to compile your programs as well as the examples in the specified environment. The "x" in SEE_4x.PDF Programmer's Manual specifies the host language such as C for C/C++, VB for Visual Basic, etc.

The **SMTP/POP3/IMAP User's Manual** (SEE_USR.PDF) discusses language independent SMTP/POP3/IMAP email processing issues. License and purchase information is also provided.

The **SMTP/POP3/IMAP Reference Manual** (SEE_REF.PDF) contains details on each individual **SEE** function.

1.4 Declaration Files

The exact syntax for calling **SMTP/POP3/IMAP Email component** (**SEE)** functions is specific to the host language (C/C++, Delphi, VB, etc.) and is defined for each language in the "**SEE** declaration files". Each **SEE** product comes with the appropriate declaration file for the supported language. For example,

```
SEE4C       C/C++, .NET, C#           SEE.H
SEE4D       Codegear (Borland) Delphi SEE32.PAS and SEE64.PAS
SEE4VB      Visual Basic              SEE32.BAS and SEE64.BAS
            VB.NET                    SEE32.VB and SEE64.VB
            VBA (EXCEL,ACCESS,etc.)   SEE32.BAS and SEE64.BAS
SEE4PB      PowerBASIC                SEE32.PBI
SEE4FP      Visual FoxPro             SEE32.FOX
SEE4DB      Visual dBase              SEE32.CC
SEE4XB      Xbase++                   SEE32.CH
SEE4CB      Fujitsu COBOL             SEE32.CBI
```

If you are programming in a language that is capable of calling Windows API functions directly, then your application can also call **SEE** functions. Let us know if you need a declaration file such a language.

Most **SEE** functions are used in one or more of the example programs.

1.5 Language Notes

All language versions of the **SMTP/POP3/IMAP Email component** library include the example program **SEEVER**. Refer to this program and the declaration file as defined in Section 1.3 above to see how **SEE** functions are called. The **SEEVER** program is also the first program that should be compiled and run.

The best way to learn how a function is called is to find it used in one of the example programs.

1.5.1 C/C++, C++ .NET, C#

If you will be using **SEE** with C/C++ and another language, order the C/C++ version.

SEE works with Visual C++, Borland C/C++, Borland C++ Builder, Watcom C/C++, LCC, Visual C++ .NET and Visual C# .NET through Visual Studio 2013.

1.5.2 Delphi

Functions defined in the Delphi Unit SEEW.PAS begin with "f" rather than "see".

SEE works all version of Delphi from Delphi 2 through Delphi 2010 and Delphi XE-XE7.

1.5.3 Visual Basic

SEE works with all version of Visual Basic including VB.NET and Visual Basic for Applications (VBA).

1.5.4 PowerBASIC

Constants defined for PowerBASIC (SEE32.PBI) begin with the character '%' symbol.

SEE works with PBCC, PBDLL, and PBWIN.

1.5.5 Visual FoxPro

All strings passed to SEE functions must be prefixed with the '@' character.

SEE works with all versions of 32-bit Visual FoxPro.

1.5.6 Visual dBase

SEE works with all versions of Visual dBase.

1.5.7 Xbase++

Functions defined for Xbase++ begin with 'X'. All strings passed to **SEE** functions must be prefixed with the '@' character.

SEE works with all versions of Alaska Xbase++.

1.5.8 COBOL

SEE4CB supports Fujitsu COBOL, but we also have a few example programs for other COBOL compilers such as ACUCOBOL, Micro Focus COBOL, Realia COBOL, and RM COBOL.

1.5.9 Fortran

We have example programs for ABSOFT, Salford, Compaq Visual Fortran, and Digital Visual Fortran.

2 SEE Functions

2.1 seeAbort: Abort seeDriver.

SYNTAX

```
seeAbort(Chan)

  Chan : (I) Channel number.
```

REMARKS

The **seeAbort** function is used to abort the **SEE** state driver. This is used when calling the **SEE** state driver (**seeDriver**) directly and it is necessary to abort.

After calling **seeAbort**, subsequent calls to **seeDriver** will return 0 (IDLE). Thus, **SEE** is ready for the next command.

This function is not required unless the state driver **seeDriver** is being called directly.

RETURNS

- Return < 0 : An error has occurred. See Section 3 “SEE Error Return Code List”.

C/C++ Example

```
// Abort SEE
seeAbort(0);
```

BASIC Example

```
' Abort SEE
Code = seeAbort(0)
```

ALSO REFER TO

seeDriver

2.2 **seeAttach**: Initialize SMTP/POP3/IMAP Email Engine.

SYNTAX

```
seeAttach(NbrChans,KeyCode)

  NbrChans : (I) Number of channels or threads.
  KeyCode  : (L) Registration key code.
```

REMARKS

The **seeAttach** function must be the first **SEE** call made. Pass the maximum number of channels or threads that will be in use. Use NbrChans = 1 for non-threaded applications.

The 'Chan' parameter for subsequent calls to **SEE** functions must be in the range of 0 to NbrChans-1.

In Win32, up to 128 threads (numbered from 0 to 127) can be started, each of which can be connected to a different server and run independently.

When SEE is registered, you will receive a 'Registration KeyCode' which matches the 'KeyCode' within the registered DLL. The keycode is 0 for the evaluation version. Refer to file KEYCODE.

RETURNS

- Return < 0 : An error has occurred. See Section 3 “SEE Error Return Code List”.

EXAMPLES

All example programs call **seeAttach**.

C/C++ Example

```
// Initialize SEE (look in KEYCODE.H for SEE_KEY_CODE)
seeAttach(1, SEE_KEY_CODE);
```

BASIC Example

```
' Initialize SEE (look in KEYCODE.BAS for SEE_KEY_CODE)
Code = seeAttach(1, SEE_KEY_CODE)
```

ALSO REFER TO

seeSmtpConnect; **seePop3Connect**.

2.3 **seeAttachmentParams**: Specifies attachment Content-Type headers.

SYNTAX

```
seeAttachmentParams(ContentType,Encoding,Disposition,Description)

  ContentType : (P) Content-Type header.
  Encoding    : (P) Content-Transfer-Encoding header.
  Disposition : (P) Content-Disposition header.
  Description : (P) Content-Description header.
```

REMARKS

This function specifies the "Content-Type:" headers for up to the first 5 attachments. Additional attachments, if any, use the standard default headers. Passing a NULL or an empty string to any one of the four parameters sets that parameter to the default value “application/octet-stream.”

The purpose of this function is to enable the creation of specific types of multi-part MIME parts. For example, specifying

```
Content-Type: audio/x-wav
```

sets the Content-Type of the attachment file to (audio) WAV that will allow some email clients to play the WAV file when the attachment is clicked.

The **seeAttachmentParams** is used with SMTP servers only.

RETURNS

- Return < 0 : An error has occurred. See Section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// set content parameters for first attachment
char *ContentType = "Content-Type: audio/x-wav";
char *XferEncoding = "Content-Transfer-Encoding: base64 ";
char *Disposition = "Content-Disposition: attachment; filename= myfile.wav;"
char *Description = "Content-Description: This is an audio wave file"
Code = seeAttachmentParams(ContentType, XferEncoding, Disposition, Description)
```

BASIC Example

```
' set content parameters for first attachment
Dim ContentType, XferEncoding, Disposition, Description As String
ContentType = "Content-Type: audio/x-wav"
XferEncoding = "Content-Transfer-Encoding: base64 "
Disposition = "Content-Disposition: attachment; filename= myfile.wav"
Description = "Content-Description: This is an audio wave file"
Code = seeAttachmentParams(ContentType, XferEncoding, Disposition, Description)
```

ALSO REFER TO

seeSmtpConnect, **seePop3Connect**; **MParts** example program.

2.4 seeByteToShort :: Converts 8-bit character buffer to 16-bit

SYNTAX

```
seeByteToShort(Buffer)

  Buffer : (P) character buffer
```

REMARKS

The **seeByteToShort** function converts the (null terminated) character buffer 'Buffer' from 8-bit ASCII characters to 16-bit Unicode ASCII characters.

The buffer must be null terminated (last character is a hex 00) and the buffer must be at least twice the size (in bytes) of the character string (since 16-bit characters require twice the space as 8-bit characters).

This function is only necessary when working with 16-bit Unicode ASCII characters in C# and Delphi .NET.

RETURNS

- Length of the string in characters.

EXAMPLES

C/C++ Example

See C example CODETEST

```
int Code;
char AsciiString[] = "MarshallSoft\0\0\0\0\0\0\0\0\0\0\0\0";

// Convert 8-bit ASCII string to 16-bit ASCII
Code = seeByteToShort((char *)AsciiString);
```

ALSO SEE

seeShortToByte

2.5 **seeClose**: Closes connection opened by SEE.

SYNTAX

```
seeClose(Chan)

  Chan : (I) Channel number.
```

REMARKS

The **seeClose** function closes the connection created by calling **seeSmtpConnect**, **seePop3Connect**, **seeSmtpConnectSSL, seePop3ConnectSSL, seeImapConnect**, or **seeImapConnect**.

Call **seeClose** to terminate the connection before connecting again.

If the connect function fails, do not call **seeClose** since the connection is already closed.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// close connection to server
seeClose(0);
```

BASIC Example

```
' seeClose(0)
Code = seeClose(0)
```

ALSO REFER TO

seeSmtpConnect , **seePop3Connect, seeSmtpConnectSSL, seePop3ConnectSSL**. All example programs call **seeClose**.

2.6 **seeCommand**: Transmits user command to SMTP or POP3 server.

SYNTAX

```
seeCommand(Chan, Text)

  Chan : (I) Channel number.
  Text : (I) Command.
```

REMARKS

The **seeCommand** function sends an arbitrary text command to the SMTP, POP3, or IMAP4 server after connecting.

The **seeCommand** function is designed to allow the user to send commands that are specific to a particular SMTP or POP3 server. It can also be used to send SMTP, POP3, or IMAP4 commands that are not implemented in the **SEE** library.

Call **seeDebug** with SEE_GET_LAST_RESPONSE in order to get the text of the server's response.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// send NOOP command to server
char *X = "NOOP"
Code = seeCommand(0, X)
```

BASIC Example

```
' send NOOP command to server
X = "NOOP" + Chr$(0)
Code = seeCommand(0, X)
```

ALSO REFER TO

HELLO example program.

2.7 seeConfigSSL: Add lines to SSL configuration file.

SYNTAX

```
seeConfigCode(ConfigCode, ConfigPtr)

  ConfigCode : (I) Configuration code
  ConfigPtr  : (P) Configuration text
```

REMARKS

The **seeConfigSSL** function adds lines to the SSL configuration file. **seeConfigSSL** provides the ability to customize the SSL configuration file for a specific server as required.

Pass SSL_CONFIG_OPTIONS as the ConfigCode parameter to specify that the contents of ConfigPtr are to be copied to the OPTIONS section of the SSL configuration file.

RETURNS

- Return > 0 : Number of bytes copied to the SSL configuration file.
- Return < 0 : An error has occurred. See Section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
char *Option = "debug=7";
Code = seeConfigSSL(SSL_CONFIG_OPTIONS, Option);
```

BASIC Example

```
Dim Option As String
Option = "debug=7"
Code = seeConfigSSL(SSL_CONFIG_OPTIONS, Option)
```

ALSO REFER TO

MailSSL example program**.**

2.8 **seeDebug**: Returns debug information.

SYNTAX

```
seeDebug(Chan, Index, Buffer, BufLen)

  Chan   : (I) Channel number.
  Index  : (I) Command index.
  Buffer : (P) Buffer to place text into.
  BufLen : (I) Length of above Buffer.
```

REMARKS

The **seeDebug** function returns (textual) debug information depending on the value of Index.

```
  SEE_COPY_BUFFER        : Copies internal buffer.
  SEE_GET_ATTACH_NAMES   : Gets list of attachment file names.
  SEE_GET_LAST_RECIPIENT : Gets last recipient acknowledged by server.
  SEE_GET_LAST_RESPONSE  : Gets last server response.
  SEE_GET_LOCAL_IP       : Gets local IP address.
  SEE_GET_REGISTRATION   : Gets the SEE registration string.
  SEE_GET_SERVER_IP      : Get server IP address in dotted notation.
  SEE_GET_ATTACH_TYPES   : Get list of attachment types.
```

SEE_COPY_BUFFER is used to copy the internal **SEE** buffer created when calling **seeGetEmailLines** with NULL passed for 'Buffer'. Refer to **seeGetEmailLines** for details. ALSO REFER TO SEE_WRITE_BUFFER in **seeStringParam**.

SEE_GET_ATTACH_NAMES gets the list of filenames received from downloading email from a POP3 server. Call only after all email has been read.

SEE_GET_LAST_RECIPIENT gets the last recipient acknowledged by the POP3 server. This is only useful when running in direct mode (calls **seeDrive**r) when a list of email addresses is being used in one call to **seeSendEmail**.

SEE_GET_LAST_RESPONSE gets the text of the last server response.

SEE_GET_LOCAL_IP gets the local IP address in dotted decimal notation. Use this only after connecting to an SMTP or POP3 server.

SEE_GET_REGISTRATION gets the user's SEE registration string.

SEE_GET_SERVER_IP gets the server IP address in dotted decimal notation. Use this only after connecting to an SMTP or POP3 server.

SEE_GET_ATTACH_TYPES gets a list of attachment types separated by semi-colons. Use only after completely downloading the email message.

RETURNS

- Return > 0 : Number of bytes copies to 'Buffer'.
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// Get text of last server response
char Buffer[128];
Code = seeDebug(0, SEE_GET_LAST_RESPONSE, (char *)Buffer, 128);
```

BASIC Example

```
' Get text of last server response
Dim Buffer As String * 128
Code = seeDebug(0, SEE_GET_LAST_RESPONSE, Buffer, 128)
```

ALSO REFER TO

seeStatistics; SEEVER and **GETRAW** example programs.

2.9 seeDecodeBuffer: Decodes buffer using base64.

SYNTAX

```
seeDecodeBuffer(CodedPtr, ClearPtr, Length)

  CodedPtr : (P) Buffer of base-64 coded chars.
  ClearPtr : (P) Buffer to put decoded bytes.
  Length   : (I) Length of above buffer.
```

REMARKS

The **seeDecodeBuffer** function decodes the buffer 'CodedPtr' of length 'Length' into 'ClearPtr', returning the length in 'ClearPtr'.

The buffer 'CodedPtr' MUST contain base-64 encoded text, as created by **seeEncodeBuffer**.

The buffer 'ClearPtr' will contain the ASCII or binary data that was encoded.

RETURNS

- Return > 0 : Number of bytes copied to ClearPtr.
- Return < 0 : An error has occurred. See Section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// BASE64 decode coded buffer
char CodedBuffer[] = "TWFyc2hhbGxTb2Z0";
char ClearBuffer[50];
CodedLength = strlen(CodedBuffer)
ClearLength = seeDecodeBuffer(CodedBuffer, ClearBuffer, CodedLength);
```

BASIC Example

```
' BASE64 decode coded buffer
CodedBuffer = "TWFyc2hhbGxTb2Z0"
ClearBuffer = Space$(50)
CodedLength = LEN(CodedBuffer)
ClearLength = seeDecodeBuffer(CodedBuffer, ClearBuffer, CodedLength)
```

ALSO REFER TO

seeEncodeBuffer; CodeTest example program**.**

2.10 seeDecodeUTF8: Decode UTF encoded character.

SYNTAX

```
seeDecodeUTF8(UTF8Buffer, UnicodeBuffer)

  UTF8Buffer   : (P) Buffer for UTF8 multibyte character.
  UnicodeBuffer: (P) Pointer to buffer for 16-bit Unicode character.
```

REMARKS

The **seeDecodeUTF8** function is used to decode a multibyte UTF8 character into a wide (16-bit) Unicode character. Upon return, the first two bytes of 'UnicodeBuffer' will contain the two bytes that make up the 16-bit Unicode value in low byte, high byte order.

More information on Unicode can be found at http://www.unicode.org

RETURNS

- Return > 0 : Number of bytes in 'UTF8Buffer' consumed, which will be 1, 2, or 3.
- Return = 0 : An error was found in the UTF8 string segment.
- Return < 0 : An error has occurred (see section 3 "SEE Error Return Code List")

EXAMPLES

C/C++ Example

See CODETEST.C

BASIC Example

See CODETEST.FRM, CODETEST.VB

ALSO REFER TO

seeDecodeUTF8String, **CODETEST** example program.

2.11 **seeDecodeUTF8String**: Decode UTF encoded string.

SYNTAX

```
seeDecodeUTF8String(UTF8Buffer, UnicodeBuffer)

  UTF8Buffer   : (P) Buffer for UTF8 string.
  UnicodeBuffer: (P) Pointer to buffer for 16-bit Unicode character.
```

REMARKS

The **seeDecodeUTF8String** function is used to decode a string of multibyte UTF8 characters into a string of wide (16-bit) Unicode characters.

More information on Unicode can be found at http://www.unicode.org

RETURNS

- Return > 0 : Number of bytes in 'UTF8Buffer' consumed.
- Return = 0 : An error was found in the UTF8 string segment.
- Return < 0 : An error has occurred (see section 3 "SEE Error Return Code List")

EXAMPLES

C/C++ Example

See CODETEST.C

BASIC Example

See CODETEST.FRM, CODETEST.VB

ALSO REFER TO

seeDecodeUTF8, **CODETEST** example program.

2.12 seeDecodeUU: Decode UU encoded line.

SYNTAX

```
seeDecodeUU (CodedBuf, ClearBuf)

  CodedBuf : (P) Buffer containing UU encoded text.
  ClearBuf : (P) Buffer into which to copy decoded text.
```

REMARKS

The **seeDecodeUU** function is used to decode a UU-encoded line passed in 'CodedBuf'. The line must be terminated with a carriage return (CR) character or by a NULL character.

For example, the following UU-encoded text contains the single line "MarshallSoft Computing.".

```
begin 666 test.txt
836%R<VAA;&Q3;V9T($-O;7!U=&EN9RX`
`
end
```

The leading byte (8) corresponds to the length of the line (24 characters) and is not passed to **seeDecodeUU**. Refer to the example below.

Unlike MIME encoded attachments, UU-encoded attachments cannot always be located algorithmically. For this reason, the programmer must write code to find any UU-encoded attachments, and decode one line at a time.

UU-encoding has been obsolete since the advent of MIME encoding. Nevertheless, some UU-encoded attachments are still being emailed, and therefore this function is provided.

RETURNS

- Return > 0: Number of bytes copied to 'ClearBuf'.
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// decode test UU-encoded string
lstrcpy((char *)CodedBuff,(char *)"36%R<VAA;&Q3;V9T($-O;7!U=&EN9RX`\r\n");
CodedLen = lstrlen((char *)CodedBuff);
ClearLen = seeDecodeUU((char *)CodedBuff,(char *)ClearBuff);
ClearBuff[ClearLen] = '\0';
```

BASIC Example

```
' decode test UU-encoded string
CodedBuff = "36%R<VAA;&Q3;V9T($-O;7!U=&EN9RX`" + Chr$(13) + Chr$(10)
CodedLen = Len(CodedBuff)
ClearLen = seeDecodeUU(CodedBuff, ClearBuff)
ClearBuff = Left$(ClearBuff,ClearLen)
```

ALSO REFER TO

seeDecodeBuffer.

2.13 **seeDeleteEmail**: Deletes email from the Server.

SYNTAX

```
seeDeleteEmail(Chan, MsgNbr)

  Chan   :  (I) Channel number.
  MsgNbr :  (I) Message number.
```

REMARKS

The **seeDeleteEmail** function deletes the email numbered 'MsgNbr' from the server.

The first message is always number 1. Call **seeGetEmailCount** first to get the number of email messages on the server. Email is not renumbered or physically deleted until you call **seeClose**.

Be careful! Once an email has been deleted from the server, it cannot be recovered.

The **seeDeleteEmail** function is used with POP3 and IMAP servers only.

RETURNS

Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// delete email message 1 (first email message) on server
seeDeleteEmail(0, 1)
```

BASIC Example

```
' delete email message 1 (first email message) on server
Code = seeDeleteEmail(0, 1)
```

ALSO REFER TO

seeGetEmailUID, **seeGetEmailCount; READER** example program**.**

2.14 **seeDriver**: Executes next SEE state.

SYNTAX

```
seeDriver(Chan)

  Chan : (I) Channel number.
```

REMARKS

The **seeDriver** function executes the next state in the **SEE** state engine. The purpose of this function is to allow the programmer to get control after the driver executes each state.

The **seeDriver** function is explicitly called only after the AUTO_DRIVER_CALL flag has been disabled (see function **seeIntegerParam**). If the AUTO_DRIVER_CALL flag has not been disabled (the default), then **seeDriver** does not need to be called.

Refer to the Section 6 "Theory of Operation" in the **SMTP/POP3/IMAP Email** User's Manual (SEE_USR) for more details on the operation of **seeDriver**.

RETURNS

- Return = 0 : The driver is done.
- Return > 0 : The returned value is the state just executed.
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// execute next state
Code = seeDriver(0);
```

BASIC Example

```
' execute next state
Code = seeDriver(0)
```

ALSO REFER TO

seeIntegerParam, seeSmtpConnect, and **seePop3Connect**; **READER** example program.

2.15 **seeEncodeBuffer**: Encodes buffer using base64.

SYNTAX

```
seeEncodeBuffer(ClearBuf, CodedBuf, Length)

  ClearBuf : (P) Buffer of characters to encode.
  CodedBuf : (P) Buffer to put base-64 encoded.
  Length   : (I) Length of above.
```

REMARKS

The **seeEncodeBuffer** function encodes 'ClearBuf' into 'CodedBuf' using Base-64 encoding.

The 'ClearBuf' buffer may contain any ASCII or binary data.

The 'CodedBuf' buffer will contain 7-bit ASCII data broken into lines of 76 characters followed by a carriage return (hex 0D) and line feed (hex 0A). That is, 'CodedBuf' will contain multiple lines.

RETURNS

- Return > 0 : Number of bytes copied to 'CodedBuf'.
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

C/C++ Example

```
// BASE64 encode buffer
char ClearBuffer[] = "MarshallSoft";
char CodedBuffer[50];
ClearLength = strlen(CodedBuffer)
CodedLength = seeEncodeBuffer(ClearBuffer, CodedBuffer, ClearLength);
```

BASIC Example

```
' BASE64 encode buffer
ClearBuffer = "MarshallSoft\r\n"
CodedBuffer = Space$(50)
ClearLength = LEN(ClearBuffer)
CodedLength = seeEncodeBuffer(ClearBuffer, CodedBuffer, ClearLength)
```

ALSO REFER TO

seeQuoteBuffer and the **CODESTEST** example program.

2.16 seeEncodeUTF8: Encodes Unicode to UTF8.

SYNTAX

```
seeEncodeUTF8(UnicodeValue, UTF8Buffer)

  UnicodeValue : (I) 16-bit unicode character.
  UTF8Buffer   : (P) Buffer for multibyte UTF8 character.
```

REMARKS

The **seeEncodeUTF8** function encodes a wide (16-bit) Unicode character into a multibyte UTF8 character.

Upon return, the 'UTF8Buffer' buffer will contain the UTF8 string corresponding to the Unicode value. The length of this string is returned.

More information on Unicode can be found at http://www.unicode.org

RETURNS

- Return > 0 : Number of bytes in 'UTF8Buffer', which will be 1, 2, or 3.
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

See CODETEST.C

BASIC Example

See CODETEST.FRM

ALSO REFER TO

SeeDecodeUTF8 and the **CODESTEST example program.**

2.17 seeEncodeUTF8String: Encodes Unicode buffer to UTF8.

SYNTAX

```
seeEncodeUTF8(UnicodeValue, UTF8Buffer)

  UnicodeBuffer : (I) 16-bit unicode characters.
  UTF8Buffer    : (P) Buffer for multibyte UTF8 string.
```

REMARKS

The **seeEncodeUTF8** function encodes a string of wide (16-bit) Unicode characters into a string of multibyte UTF8 characters..

The length of UTF8 string is returned.

More information on Unicode can be found at http://www.unicode.org

RETURNS

- Return > 0 : Number of bytes in 'UTF8Buffer'
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

See CODETEST.C

BASIC Example

See CODETEST.FRM

ALSO REFER TO

seeDecodeUTF8String and **seeEncodeUTF8**

2.18 **seeErrorText**: Get text associated with error code.

SYNTAX

```
seeErrorText(Chan, ErrCode, Buffer, BufLen)

  Chan   : (I) Channel number.
  ErrCode: (I) Error code returned by SEE function.
  Buffer : (P) Buffer to place error text into.
  BufLen : (I) Length of above Buffer.
```

REMARKS

The **seeErrorText function** is used to get the error text associated with an error code as returned by one of the other **SEE** functions.

When an error occurs, **seeErrorText** can be used to get the error text so that it can be displayed for the user. See Section 3 "SEE Error Return Code List." The file, ERRORS.TXT, contains a list of all error codes and their corresponding numeric value.

RETURNS

- Return > 0 : Length of text message copied into 'Buffer'.
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// get text associated with ErrCode
char Buffer[80];
Code = seeErrorText(0, ErrCode, (char *)Buffer, 80)
```

BASIC Example

```
' // get text associated with ErrCode
Dim Buffer As String * 80
Code = seeErrorText(0, ErrCode, Buffer, 80)
```

Most example programs call **seeErrorText**.

2.19 **seeExtractLine**: Extract specified line from buffer.

SYNTAX

```
seeExtractLine(Src, Line, Buffer, BufSize)

  Src     : (P) Text buffer to search.
  Line    : (I) Line number (1,2,...) wanted.
  Buffer  : (P) Buffer for line if found.
  BufSize : (I) Size of 'Buffer'.
```

REMARKS

The **seeExtractLine** function is used to extract line 'Line' from buffer 'Src'. If the specified line number is found, then the entire line, up to a maximum of 'BufSize' bytes, is copied to 'Buffer'.
The primary purpose of **seeExtractLine** is to extract header lines (by line number) from the buffer after calling **seeGetEmailLines**. Recall that all lines returned by **seeGetEmailLines** are terminated by a carriage return, linefeed pair. Lines are numbered from 1 rather than 0.

The **seeExtractLine** does not require a connection to an SMTP or POP3 server.

RETURNS

- Return > 0 : Number of bytes placed in 'Buffer'.
- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// extract line # 4 from (multi-line) buffer ListBuffer
char LineBuffer[128];
seeExtractLine((char *)ListBuffer, 4, (char *)LineBuffer, 128);
```

BASIC Example

```
' extract line # 4 from (multi-line) buffer ListBuffer
Dim LineBuffer As String * 128
seeExtractLine(ListBuffer, 4, LineBuffer, 128)
```

ALSO REFER TO

seeGetEmailLines and **seeExtractText**.; **STATUS** example program.

2.20 seeExtractText: Extract specified text from buffer.

SYNTAX

```
seeExtractText(Src, Text, Buffer, BufSize)

  Src     : (P) Text buffer to search.
  Text    : (P) Text searching for.
  Buffer  : (P) Buffer for line if found.
  BufSize : (I) Size of 'Buffer'.
```

REMARKS

The **seeExtractText** function is used to search the text buffer 'Src' for text 'Text'. If the specified text is found, then the entire line, up to a maximum of 'BufSize' bytes, is copied to 'Buffer'.

The primary purpose of **seeExtractText** is to extract header lines from the buffer after calling **seeGetEmailLines**. Recall that all lines returned by **seeGetEmailLines** are terminated by a carriage return, linefeed pair.

The **seeExtractText** does not require a connection to a SMTP or POP3 server.

RETURNS

- Return > 0 : Number of bytes placed in 'Buffer'.
- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// extract line from Buffer containing "Reply-To: "
char Buffer[255];
n = seeExtractText(Buffer, (char *)"Reply-To: ", Buffer, 255)
```

BASIC Example

```
' extract line from Buffer containing "Reply-To: "
ExString = "Reply-To: "
Dim Buffer As String * 255
n = seeExtractText(Buffer, ExString, Buffer, 255)
```

ALSO REFER TO

seeGetEmailLines, **seeExtractLine**; **STATUS** and **FROM** example programs.

2.21 seeForwardEmail: Forward email.

SYNTAX

```
seeForward(Chan, To, CC, BCC, Subj, Msg, Forward)

  Chan   : (I) Channel number.
  To     : (P) Recipient, separated by semi-colons.
  CC     : (P) CC list, separated by semi-colons.
  BCC    : (P) BCC list, separated by semi-colons.
  Subj   : (P) Subject text.
  Msg    : (P) Message or message filename.
  Forward: (P) Filename (of undecoded email) to forward.
```

REMARKS

The **seeForward** function is used to forward an email to a new recipient. The filename of the (undecoded) email to be forwarded must be attached as the last argument, and is encoded as a message/rfc822 MIME part.

For an explanation of how to download an undecoded copy of an email, refer to the REMARKS section provided for the **seeGetEmailFile** function.

The **seeForwardEmail** function is used with SMTP servers only.

RETURNS

- Return > 0 : The number of bytes read.
- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// forward (undecoded) email file Email2Forward to <info@yourisp.com>
Code = seeForwardEmail(0, (char *)"<info@yourisp.com>",
  (char *)NULL, (char *)NULL, (char *)"Test",
  (char *)"Forwarding test", Email2Forward);
```

BASIC Example

```
' forward (undecoded) email file Email2Forward to <info@yourisp.com>
To = "<info@yourisp.com>"
CC = Chr$(0)
BCC = Chr$(0)
Subject = "Test"
Message = "Forwarding test"
Code = seeForwardEmail(0, To, CC, BCC, Subject, Message, Email2Forward)
```

Refer to the FORWARD example program.

ALSO REFER TO

seeGetEmailFile; FORWARD example program.

2.22 **seeGetEmailCount**: Get number of email messages on server.

SYNTAX

```
seeGetEmailCount(Chan)

  Chan : (I) Channel number.
```

REMARKS

The **seeGetEmailCount** function returns the number of messages waiting on the server, independent of whether they have been previously read.

If you have disabled the driver AUTO_CALL capability, the message count must be found by calling

```
seeStatistics(0, SEE_GET_MSG_COUNT)
```

after calling **seeDriver** until it returns 0.

The **seeGetEmailCount** function is used with POP3 and IMAP servers only.

RETURNS

- Return > 0 : The number of email messages waiting.
- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// get # messages waiting on server
NbrMsg = seeGetEmailCount(0);
```

BASIC Example

```
' get # messages waiting on server
NbrMsg = seeGetEmailCount(0)
```

ALSO REFER TO

seeGetEmailLines; STATUS example program.

2.23 **seeGetEmailFile**: Read email message and save to a file.

SYNTAX

```
seeGetEmailFile(Chan, MsgNbr, EmailName, EmailDir, AttachDir)

  Chan      : (I) Channel number.
  MsgNbr    : (I) Message #.
  EmailName : (P) Email filename.
  EmailDir  : (P) Directory for email.
  AttachDir : (P) Directory for attachments.
```

REMARKS

The **seeGetEmailFile** reads the email message 'MsgNbr', saving it to disk as filename 'EmailName' in directory 'EmailDir', and saving MIME attachments to directory 'AttachDir'. The current directory is specified as '.'.

Be sure that the specified directories exist before calling this function. Use '.' to specify the current directory. Also note that an older file of the same name as the newer file will be overwritten.

An undecoded copy of the email being downloaded can be saved to disk by calling

seeIntegerParam(Chan, SEE_SET_RAWFILE_PREFIX, prefix-char)

before calling **seeGetEmailFile**. For example, if the prefix character is the underscore '_' and the filename passed to **seeGetEmailFile** is "`mail.txt`", then the undecoded copy will be named "`_mail.txt`".

The **seeGetEmailFile** function is used with POP3 and IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
    char FileName[] = "Email4.txt";
    char EmailDir[] = "\\SEE4C\\APPS";
    char AttachDir[] = "\\SEE4C\\APPS";
    Code = seeGetEmailFile(0, 4, (char *)FileName,(char *)EmailDir,(char
*)AttachDir);
```

BASIC Example

```
    Dim FileName, EmailDir, AttachDir As String
    FileName = "Email4.txt"
    EmailDir = "\SEE4C\APPS"
    AttachDir = "\SEE4C\APPS"
    Code = seeGetEmailFile(0, 4, FileName, EmailDir, AttachDir)
```

ALSO REFER TO

seeGetEmailLines and **seePop3Source**; **READER** example program.

2.24 **seeGetEmailLines**: Read lines from email message.

SYNTAX

```
seeGetEmailLines(Chan, MsgNbr, Lines, Buffer, Size)

  Chan   : (I) Channel number.
  MsgNbr : (I) Message #.
  Lines  : (I) Number of body lines.
  Buffer : (P) Pointer to (static) Buffer.
  Size   : (I) Size of buffer.
```

REMARKS

The **seeGetEmailLines** function reads all header lines plus the number of body lines specified by the 'Lines' argument into 'Buffer', up to a maximum of 'Size' bytes.

The primary purpose of this function is to read the header lines without having to read the entire message.

If you have disabled the driver AUTO_CALL capability, the size must be found by calling

```
seeStatistics(Chan, SEE_GET_BUFFER_COUNT)
```

after calling **seeDriver** until it returns 0.

If a 0 is passed as the 4th argument (Buffer), **SEE** will use an internal buffer, the contents of which can be accessed by subsequently calling `seeDebug(Chan, SEE_COPY_BUFFER,...)` or calling `seeStringParam(Chan, SEE_WRITE_BUFFER,...)`. This technique is ONLY necessary in direct mode (calling **seeDriver** directly) from languages (such as Visual Basic, dBase 2000, etc.) that cannot statically allocate memory. Refer to the **SMTP/POP3/IMAP Email Programmer's Manual**.

The **seeGetEmailLines** function is used with POP3 and IMAP servers only.

RETURNS

- Return > 0 : The number of bytes read.
- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// read email message 5 (headers only) w/o decoding
char Buffer[10000];
Code = seeGetEmailLines(0, 5, 0, (char *)Buffer, 10000);
```

BASIC Example

```
' read email message 5 (headers only) w/o decoding
Dim Buffer As String * 10000
Code = seeGetEmailLines(0, 5, 0, Buffer, 10000)
```

ALSO REFER TO

seeGetEmailFile; STATUS example program.

2.25 **seeGetEmailSize**: Get size of email message in bytes.

SYNTAX

```
seeGetEmailSize(Chan, MsgNbr)

  Chan   : (I) Channel number.
  MsgNbr : (I) Message number
```

REMARKS

The **seeGetEmailSize** function returns the size in bytes of the specified message 'MsgNbr'.

seeGetEmailSize returns the size of the entire email message, including any attachments. Note that attachments will be encoded (MIME, UUENCODE, etc.), and thus take up more room than after they are decoded.

If you have disabled the driver AUTO_CALL capability, the size must be found by calling

```
seeStatistics(0, SEE_GET_MSG_SIZE)
```

after calling **seeDriver** until it returns 0.

The **seeGetEmailSize** function is used with POP3 and IMAP servers only.

RETURNS

- Return > 0 : The size of the email in bytes on the server.
- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// get the size of email message # 2
MsgSize = seeGetEmailSize(0, 2);
```

BASIC Example

```
' get the size of email message # 2
MsgSize = seeGetEmailSize(0, 2)
```

ALSO REFER TO

seeGetEmailCount.; **STATUS** and **READER** example programs

2.26 **seeGetEmailUID**: Get user ID from the server.

SYNTAX

```
seeGetEmailUID(Chan, MsgNbr, Buffer, Size)

  Chan   : (I) Channel number.
  MsgNbr : (I) Message (-1 for all) number.
  Buffer : (P) Pointer to Buffer.
  Size   : (I) Size of buffer.
```

REMARKS

The **seeGetEmailUID** function is used to ask the POP3 server for the unique user ID string for a particular email message, or for all email messages on the server.

The UID string is always unique for a particular email message, regardless of the email message number or when it was received. The purpose of the UID is to allow the client to determine if a particular email has been seen by it previously. The UID is useful is situations in which mail is left on the server and is accessed by more than one email client program.

Most POP3 servers can provide such a unique ID string. A few POP3 servers do not provide ID strings.

The **seeGetEmailUID** function is used with POP3 and IMAP servers only.

RETURNS

- Return > 0 : The number of bytes in moved into 'Buffer'.
- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// get UID string for email message # 1
char Buffer[256];
n = seeGetEmailUID(0, 1, (char *)Buffer, 256);
```

BASIC Example

```
' get UID string for email message # 1
Dim Buffer As String * 256
n = seeGetEmailUID(0, 1, Buffer, 256)
```

ALSO REFER TO

seeGetEmailCount; STATUS example program.

2.27 **seeGetHeader**: Get Header

SYNTAX

```
seeGetHeader(Chan, Param, Buffer, Size)

  Chan   : (I) Channel number.
  Param  : (I) Parameter (see below).
  Buffer : (P) Pointer to Buffer.
  Size   : (I) Size of buffer.
```

REMARKS

The **seeGetHeader** function returns the selected header once an email has been read.

Parameter	Description
SEE_GET_DATE	Get "Date:" header.
SEE_GET_FROM	Get "From:" header.
SEE_GET_REPLY_TO	Get "Reply-To:" header.
SEE_GET_SUBJECT	Get "Subject:" header.
SEE_GET_TO	Get "To:" header.
SEE_GET_CC	Get "CC:" header.
SEE_GET_CONTENT_TYPE	Get "Content-Type:" header.

RETURNS

- Return > 0 : The number of character in the returned header
- Return -1 : Parameter is not recognized.

EXAMPLES

C/C++ Example

```
char Buffer[256];
// get "Subject:" header
Code = seeGetHeader(0, SEE_GET_SUBJECT, (char *)Buffer, 255);
if(Code>0) printf("Subject: %s\n", Buffer);
```

BASIC Example

```
Dim Buffer As String
' get "Subject:" header
Buffer = SPACE{256)
Code = seeGetHeader(0, SEE_GET_SUBJECT, Buffer, 255)
```

2.28 **seeGetTicks**: Get System Ticks.

SYNTAX

```
seeGetTicks(Divisor)

  Divisor   : (I) Divisor
```

REMARKS

The **seeGetTicks** function returns the number of milliseconds since the system was booted, divided by the divisor.

For example; to get milliseconds, call seeGetTicks(1). To get seconds, call seeGetTicks(1000).

RETURNS

- Return > 0 : The number of milliseconds since the system was booted, divided by the divisor.
- Return = 0 : If Divisor < 1

EXAMPLES

C/C++ Example

```
// get ticks in units of 1/100 second.
n = seeGetTicks(10);
```

BASIC Example

```
' get ticks in units of 1/100 second
n = seeGetTicks(10)
```

2.29 **seeImapConnect:** Connects to IMAP server.

SYNTAX

```
seeImapConnect(Chan, Pop3Ptr, UserPtr, PassPtr)

    Chan : (I) Channel number
 Pop3Ptr : (P) IMAP server name
 UserPtr : (P) Sender's email address
 PassPtr : (P) Reply-To header
```

REMARKS

The **seeImapConnect** function is used to connect to an IMAP server. Note that **seeImapConnect** uses the same arguments as **seePop3Connect**.

The well-known IMAP (default) port is 143 but can be changed by calling

```
seeIntegerParam(Chan, SEE_IMAP_PORT, new-port-to-use)
```

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// connect to POP3 server "mail.yourisp.com"
char ImapHost[] = "mail.yourisp.com";
char ImapUser[] = "bill";
char ImapPass[] = "abc";
Code = seeImapConnect(0, (char *)ImapHost, (char *)ImapUser,
                         (char *)ImapPass);
```

BASIC Example

```
' connect to Imap server "mail.yourisp.com"
Dim ImapHost, ImapUser. ImapPass As String
ImapHost = "mail.yourisp.com"
ImapUser = "bill"
ImapPass = "abc"
Code = seeImapConnect(0, ImapHost, ImapUser, ImapPass)
```

ALSO REFER TO

seeImapConnectSSL

2.30 **seeImapConnectSSL:** Connects to IMAP Server with SSL

SYNTAX

```
seeImapConnectSSL(Chan, ProxyPort, ImapPort, ImapServer,
                  ImapUser, ImapPass, ProxyIP)

        Chan : (I) Channel number
   ProxyPort : (I) Port to connect to proxy on
    ImapPort : (I) IMAP port (normally 993)
  ImapServer : (P) IMAP server name
    ImapUser : (P) Sender's email address
    ImapPass : (P) Reply-To header
     ProxyIP : (P) Host name or IP address of proxy server
```

REMARKS

The **seeImapConnectSSL** function is used to connect to an IMAP server with SSL. Note that **seeSetProxySSL** must be called before calling **seeImapConnectSSL**.

'ProxyPort' can be an unused port. 'ImapPort' for SSL will normally be 993. 'ImapServer' is the name or IP address of the IMAP server. 'ImapUser' is the user name and 'ImapPass' is the password. 'ProxyIP' is the host name or IP address of the proxy server. Pass NULL or an empty string to specify this computer (127.0.0.1).

Stunnel must first be installed. **seeImapConnectSSL** will automatically start and stop Stunnel as needed. To set up Stunnel, see "Using Stunnel" in the SEE User's Manual (SEE_USR.PDF) or online at http://www.marshallsoft.com/stunnel.htm.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// connect to IMAP server "imap.gmail.com"
char *ImapServer = "imap.gmail.com";
char *ImapUser = "bill@gmail.com";
char *ImapPass = "abc";
Code = seeImapConnectSSL(0,8803,993,ImapServer,ImapUser,ImapPass,NULL);
```

BASIC Example

```
' connect to IMAP server "imap.gmail.com"
Dim ImapHost, ImapUser. ImapPass As String
ImapServer = "imap.gmail.com"
ImapUser = "bill@gmail.com"
ImapPass = "abc"
Code = seeImapConnectSSL(0,8803,993,ImapServer,ImapUser,ImapPass,Chr(0))
```

ALSO REFER TO

seeImapConnect

2.31 seeImapCopyMBmail: Copy messages to specified mailbox.

SYNTAX

```
seeImapCopyMBmail(Chan, Message, Mailbox)

     Chan : (I) Channel (0, 1, 2, ...)
  Message : (I) Message number to copy (1, 2, ...)
  Mailbox : (P) Name of destination mailbox
```

Note: seeImapCopyMBmail() is a "copy to", not a "copy from".

REMARKS

The **seeImapCopyMBmail** function is used to copy messages from the selected mailbox to the specified mailbox. That is, an email message is copied from the currently selected mailbox to the mailbox specified by the argument "Mailbox".

The **seeImapCopyMBmail** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
char SavedBox[128];
strcpy((char *)SavedBox, "SavedBox");
Code = seeImapCopyMBmail(0, 7, SavedBox);
```

BASIC Example

```
Dim SavedBox As String
SavedBox = "SavedBox"
Code = seeImapCopyMBmail(0, 7, SavedBox)
```

2.32 seeImapCreateMB: Create a new mailbox.

SYNTAX

```
seeImapCreateMB(Chan, Mailbox)

    Chan : (I) Channel (0, 1, 2, ...)
 Mailbox : (P) Name of mailbox to create
```

REMARKS

The **seeImapCreateMB** function is used to create a new mailbox (MB). Once created, mail can be transferred between the new mailbox, the InBox, and any other previously created mailbox.

The **seeImapCreateMB** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
char SavedBox[128];
strcpy((char *)SavedBox, "SavedBox");
Code = seeImapCreateMB(0, SavedBox);
```

BASIC Example

```
Dim SavedBox As String
SavedBox = "SavedBox"
Code = seeImapCreateMB(0, SavedBox)
```

2.33 seeImapDeleteMB: Delete a mailbox.

SYNTAX

```
seeImapDeleteMB(Chan, Mailbox)

      Chan : (I) Channel (0, 1, 2, ...)
   Mailbox : (P) Name of mailbox to delete
```

REMARKS

The **seeImapDeleteMB** function is used to delete a mailbox (MB) previously created. However, the InBox cannot be deleted.

The **seeImapDeleteMB** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
char SavedBox[128];
strcpy((char *)SavedBox, "SavedBox");
Code = seeImapDeleteMB(0, SavedBox);
```

BASIC Example

```
Dim SavedBox As String
SavedBox = "SavedBox"
Code = seeImapDeleteMB(0, SavedBox)
```

2.34 seeImapFlags: Get, set, or delete IMAP message flags.

SYNTAX

seeImapFlags(Chan, MsgNbr, Command, FlagsMask)

```
    Chan     : (I) Channel (0, 1, 2, ...)
    MsgNbr   : (I) Message Number (1, 2, 3, ...)
    Command  : (I) Command (see below)
  FlagsMask  : (I) Flag mask. May be combined. (see below)

Command Symbol : Value, Meaning
IMAP_GET_FLAGS : 1, Get message flags
IMAP_SET_FLAGS : 2, Set message flags
IMAP_DEL_FLAGS : 3, Delete message flags

FlagsMask Symbol   : Value, Meaning
IMAP_FLAG_SEEN     :  1, Message has been read
IMAP_FLAG_ANSWERED :  2, Message has been answered
IMAP_FLAG_FLAGGED  :  4, Message is "flagged" for special attention
IMAP_FLAG_DELETED  :  8, Message is "deleted" for removal
IMAP_FLAG_DRAFT    : 16, Message has been marked as a draft
IMAP_FLAG_RECENT   : 32, Message has arrived since the previous
                         time this mailbox was selected.

IMAP_FLAG_SEEN refers to "\Seen" flag.
IMAP_FLAG_ANSWERED refers to "\Answered" flag.
IMAP_FLAG_FLAGGED refers to "\Flagged" flag.
IMAP_FLAG_DELETED refers to "\Deleted" flag.
IMAP_FLAG_DRAFT refers to "\Draft" flag.
IMAP_FLAG_RECENT refers to "\Recent" flag.

Note "\Recent" may be fetched but not stored.
```

REMARKS

The **seeImapFlags** function is used to get, set, or delete message flags.

The **seeImapFlags** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".
- Return > 0 : IMAP flags that are set. Refer to the **ImapFlags** example program.

EXAMPLES

C/C++ Example

```
Flags = IMAP_FLAG_ANSWERED | IMAP_FLAG_FLAGGED;
Code = seeImapFlags(0, 1, IMAP_SET_FLAGS, Flags);
```

BASIC Example

```
Flags = IMAP_FLAG_ANSWERED + IMAP_FLAG_FLAGGED
Code = seeImapFlags(0, 1, IMAP_SET_FLAGS, Flags)
```

2.35 **seeImapListMB:** List all available mailboxes on IMAP server

SYNTAX

```
seeImapListMB(Chan, Buffer, BufLen)

   Chan : (I) Channel (0, 1, 2, ...)
 Buffer : (P) Buffer into which the list of mailboxes are copied.
 BufLen : (I) Length of above buffer.
```

REMARKS

The **seeImapListMB** function is used to ask for a full list of all mailboxes. The exact format of the list will vary according to the server's operating system.

Under normal circumstances, the user will already know which mailboxes have been created.

The **seeImapListMB** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
char Buffer[3000];
Code = seeImapListMB(0, (char *)Buffer, 3000);
```

BASIC Example

```
Dim Buffer As String
Buffer = SPACE(3000)
Code = seeImapListMB(0, Buffer, 3000)
```

2.36 seeImapMsgNumber: Gets message #'s filled by seeImapSearch.

SYNTAX

```
seeImapMsgNumber(Chan, Command)

     Chan : (I) Channel (0, 1, 2, ...)
  Command : (I) Command (see below)

  Command Symbol         :  Value, Meaning
  IMAP_SEARCH_MSG_COUNT  :  1, Get # messages found by seeImapSearch
  IMAP_SEARCH_FIRST_MSG  :  2, Get first message of set found.
  IMAP_SEARCH_NEXT_MSG   :  3, Get next message of set found.
```

REMARKS

The **seeImapMsgNumber** function called immediately after **seeImapSearch** and is used to return:

(1) the number of messages found by **seeImapSearch**
(2) the first message found by **seeImapSearch**
(3) the next message found by **seeImapSearch** (call iteratively)

See the ImapSearch example program.

The **seeImapMsgNumber** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".
- Return > 0 : # messages found (if passed IMAP_SEARCH_MSG_COUNT)
- First message number found (if passed IMAP_SEARCH_FIRST_MSG)
- Next message number found (if passed IMAP_SEARCH_NEXT_MSG)

EXAMPLES

C/C++ Example

```
NbrMsg = seeImapMsgNumber(0, IMAP_SEARCH_MSG_COUNT);
```

BASIC Example

```
NbrMsg = seeImapMsgNumber(0, IMAP_SEARCH_MSG_COUNT)
```

ALSO REFER TO

ImapSearch example program.

2.37 seeImapRenameMB: Rename IMAP mailbox.

SYNTAX

```
seeImapRenameMB(Chan, FromName, ToName)

     Chan : (I) Channel (0, 1, 2, ...)
 FromName : (P) Existing name of mailbox
   ToName : (P) New name of mailbox
```

REMARKS

The **seeImapRenameMB** function is used to rename a new mailbox (MB).

The **seeImapRenameMB** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
char SourceBox[128];
char TargetBox[128];
strcpy((char *)SourceBox, "SavedBox");
strcpy((char *)TargetBox, "ArchiveBox");
Code = seeImapRenameMB(0, SourceBox, TargetBox);
```

BASIC Example

```
Dim SourceBox As String
Dim TargetBox As String
SourceBox = "SavedBox"
TargetBox = "ArchiveBox"
Code = seeImapRenameMB(0, SourceBox, TargetBox)
```

2.38 seeImapSearch: Search for IMAP messages with specified flags.

SYNTAX

```
seeImapSearch(Chan, SearchArgs, Buffer, BufLen)

  Chan       : (I) Channel (0, 1, 2, ...)
  SearchArgs : (P) Search string.
  Buffer     : (P) Result buffer where message numbers are copied.
  BufLen     : (I) Size of result buffer.
```

REMARKS

The **seeImapSearch** function is used to search for messages with specified strings.

Example search strings as passed to **seeImapSearch**():

```
SEEN
SEEN NOT ANSWERED
FLAGGED SINCE 1-Feb-2008 NOT FROM "Smith"
LARGER 10000 NOT SEEN
```

Refer to ImapSearch.txt, RFC 1730, or http://www.marshallsoft.com/ImapSearch.htm for a complete list.

After calling **seeImapSearch,** the function **seeImapMsgNumber** must be called in order to get the message numbers that **seeImapSearch** found that match the search criteria.

The **seeImapSearch** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
strcpy((char *)SearchArgs, (char *)"LARGER 10000 NOT SEEN" );
Code = seeImapSearch(0, SearchArgs, (char *)Buffer, 128);
```

BASIC Example

```
DIM SearchArgs As String
SearchArgs = "LARGER 10000 NOT SEEN"
Code = seeImapSearch(0, SearchArgs, Buffer, 128)
```

ALSO REFER TO

ImapSearch example program.

2.39 seeImapSelectMB: Selects IMAP mailbox.

SYNTAX

```
seeImapSelectMB(Chan, Mailbox)

      Chan : (I) Channel (0, 1, 2, ...)
   Mailbox : (P) Name of mailbox to select
```

REMARKS

The **seeImapSelectMB** function is used to select a mailbox (MB) for processing. The standard inbox is "InBox".

The **seeImapSelectMB** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
char SavedBox[128];
strcpy((char *)SavedBox, "SavedBox");
Code = seeImapSelectMB(0, SavedBox);
```

BASIC Example

```
Dim SavedBox As String
SavedBox = "SavedBox"
Code = seeImapSelectMB(0, SavedBox)
```

2.40 seeImapSource: Specified file from which to read IMAP email

SYNTAX

```
seeImapSource(Chan, ImapFilename)

  Chan     : (I) Channel number.
  Filename : (P) IMAP filename.
```

REMARKS

The **seeImapSource** function is used to specify the path to a file containing an undecoded email. After calling this function, **seeGetEmailFile** can be called to decode the email as if it were being downloaded from an IMAP server.

Note that there is no IMAP connection. The email is read directly from a file.

The **seeImapSource** function is used with IMAP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// decode email file
char Source[] = "c:\\see4c\\apps\\raw_email.txt";
Code = seeImapSource(0, (char *)Source);
```

BASIC Example

```
'decode email file
Dim Source As String
Source = "c:\see4c\apps\raw_email.txt"
Code = seeImapSource(0, Source)
```

ALSO REFER TO

seeImapConnect and the POP3PRD example program.

2.41 **seeIntegerParam**: Sets numeric parameter to control email processing.

SYNTAX

```
seeIntegerParam(Chan, ParamIndex, ParamValue)

  Chan       : (I) Channel number.
  ParamIndex : (I) Parameter name (see below).
  ParamValue : (L) Value of parameter to set.
```

REMARKS

The **seeIntegerParam** function is used to set an integer parameter that is passed to the **SEE** library to provide additional control of email processing. The numeric value for each of the integer parameters is defined in the **SEE** declaration files. Section 1.3 “Declaration Files” provides a list the "Declaration Files". All times are in milliseconds. Defaults values are as follows:

Parameter Name	Value
SEE_ADDRESS_DELIMITER	';' [semicolon]
SEE_ALLOW_8BITS	0 [FALSE]
SEE_ATTACH_DELIMITER	';' [semicolon]
SEE_ATTACH_BASE_NUMBER	0 [FALSE]
SEE_AUTHENTICATE_PROTOCOL	1 [CRAM-MD5 authentication]
SEE_AUTO_CALL_DRIVER	1 [TRUE]
SEE_BLOCKING_MODE	1
SEE_CONNECT_WAIT	60000
SEE_DECODE_UNNAMED	0 [FALSE]
SEE_ENABLE_APOP	0 [FALSE]
SEE_ENABLE_ESMTP	0 [FALSE]
SEE_ENABLE_IMAGE	1 [TRUE]
SEE_FILE_PREFIX	0 [FALSE]
SEE_FORCE_INLINE	0 [FALSE]
SEE_GUT_ATTACHMENTS	0 [FALSE]
SEE_HIDE_HEADERS	0 [FALSE]
SEE_HIDE_SAVED_MSG	0 [FALSE]
SEE_HIDE_TO_ADDR	0 [FALSE]
SEE_IGNORE_REJECTED	0 [FALSE]
SEE_IMAP_PORT	143
SEE_KEEP_RFC822_INTACT	0 [FALSE]
SEE_MAX_LINE_LENGTH	76
SEE_MAX_RESPONSE_WAIT	25000
SEE_MIN_RESPONSE_WAIT	250
SEE_PATH_DELIMITER	';' [semicolon]
SEE_POP3_PORT	110
SEE_QUOTED_PRINTABLE	0 [FALSE]
SEE_RAW_MODE	0 [FALSE]
SEE_REPLACE_UNDERSCORES	1 [TRUE]
SEE_REPLACE_WITH_COMMAS	1 [TRUE]
SEE_SET_CONNECT_ATTEMPTS	5
SEE_SET_RAWFILE_PREFIX	0
SEE_SLEEP_TIME	50
SEE_SMTP_PORT	25
SEE_WRITE_CONTENT_TYPE	0 [FALSE]
SEE_WSACLEANUP	1 [TRUE]

SEE_ADDRESS_DELIMITER sets the delimiter to use for separating multiple email addresses.

SEE_ALLOW_8BITS will allow 8-bit data inside an email message. Note that 7-bit ASCII is the RFC822 standard. Not recommended!

SEE_ATTACH_BASE_NUMBER sets the first numeric value to be used as the prefix for attachments. Use 0 to not prefix attachment filenames.

SEE_ATTACH_DELIMITER is the character that delimits attachment name from file name (used to specify an attachment name different from the filename). For example, “FileName.txt|AttachName.txt”.

SEE_AUTHENTICATE_PROTOCOL specifies the protocol to use in performing SMTP authentication. Values are AUTHENTICATE_CRAM (value 1), AUTHENTICATE_LOGIN (value 2), AUTHENTICATE_PLAIN (value 4), and AUTHENTICATE_XOAUTH2 (value 8).

SEE_AUTO_CALL_DRIVER controls whether **seeDriver** is called automatically (to completion) after a SEE function has been called.

SEE_BLOCKING_MODE allows connect attempts to block waiting for the server response.

SEE_CONNECT_WAIT is the maximum time allowed to complete a connection to the email server.

SEE_DECODE_UNNAMED will force (if non-zero) decoding of all base64 attachments which do not have names.

SEE_ENABLE_APOP directs that the APOP command will be used for authenticating POP3 connections rather than USER and PASS. Requires that POP3 server recognizes the APOP command.

SEE_ENABLE_ESMTP enables ESMTP (rather than SMTP) connections when calling **seeSmtpConnect**.

SEE_ENABLE_IMAGE allows (if non-zero) files ending with ".GIF", ".BMP", ".JPG", or ".TIF" to be attached as image types so that they can be displayed by the recipient's email client.

SEE_FILE_PREFIX controls whether "1-", "2-", etc. is prefixed to the filename of each attachment. If two attachments are named FILEONE.ZIP and FILETWO.ZIP, they will be saved as 1-FILEONE.ZIP and 2-FILETWO.ZIP. This feature should always be used unless you are downloading to a directory specifically for downloaded attachments. Pass an integer >= 1 to specify the first prefix.

SEE_FORCE_INLINE specifies if text attachments are inline or not. Possible values are INLINE_TEXT_OFF (0, not forced inline), INLINE_TEXT_INLINE (1, text attachments coded inline), or INLINE_TEXT_ATTACHMENT (2, text attachments attached as file).

SEE_GUT_ATTACHMENTS specifies that the contents of all attachments should be removed. Pass 1 to enable, 0 to disable [default).

SEE_HIDE_HEADERS causes headers (such as “From:”, “Subject:”, etc.) to <u>not</u> be written to the email output file. This flag overrides any conflicting flags.

SEE_HIDE_SAVED_MSG is used to hide the "Attachment saved to " message in incoming email.

SEE_HIDE_TO_ADDR is used to hide the "To:" field in outgoing email.

SEE_IGNORE_REJECTED directs **SEE** to ignore error returned if recipient is rejected.

SEE_IMAP_PORT changes the IMAP port.

SEE_KEEP_RFC822_INTACT causes SEE to keep RFC822 messages intact (not decoded) when downloaded. The RFC822 message will be saved to disk as a text attachment.

SEE_MAX_LINE_LENGTH specifies the maximum length of the lines sent to the SMTP server. Note that the RFC822 standard specifies 76 characters.

SEE_MAX_RESPONSE_WAIT is the time after which a "timeout" error occurs if the server has not responded.

SEE_MIN_RESPONSE_WAIT is the delay before looking for the server's response.

SEE_PATH_DELIMITER is the character that delimits multiple file paths.

SEE_POP3_PORT changes the POP3 port.

SEE_QUOTED_PRINTABLE controls whether messages are or are not encoded as quoted-printable. The 3rd parameter should be one of :

```
QUOTED_HTML          QUOTED_ISO_8859_1    QUOTED_ISO_8859_2
QUOTED_ISO_8859_3    QUOTED_ISO_8859_4    QUOTED_ISO_8859_7
QUOTED_ISO_8859_8    QUOTED_OFF           QUOTED_PLAIN
QUOTED_RICH          QUOTED_USER          QUOTED_UTF8
QUOTED_WIN_1250      QUOTED_WIN_1252      QUOTED_WIN_1255
```

SEE_RAW_MODE disables all decoding of email when downloaded by **seeGetEmailFile**.

SEE_REPLACE_UNDERSCORES directs SEE to replace underscore with spaces (if ParamValue = TRUE) or not (if ParamValue = FALSE) in incoming email attachment filenames. The default is TRUE (as per RFC 2047).

SEE_REPLACE_WITH_COMMAS causes the replacement of address delimiters in TO, CC, and BCC headers with commas.

SEE_SET_RAWFILE_PREFIX causes an undecoded copy of an email downloaded by **seeGetEmailFile** to be saved with the same name as the decoded copy except prefixed by the specified prefix character.

SEE_SET_CONNECT_ATTEMPTS sets the number of time SEE will attempt to connect to a server. Valid values are 1 to 12.

SEE_SLEEP_TIME is the time **SEE** sleeps when waiting on a Winsock.

SEE_SMTP_PORT changes the SMTP port.

SEE_WRITE_CONTENT_TYPE causes (if `ParamValue` not 0) the Content-Type header to be written to the output email file (POP3 connections). Normally, this header is not written. Also refer to SEE_SET_CONTENT_TYPE_PREFIX.

RETURNS

- Return > 0 : Integer parameter requested.
- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
//enable Extended SMTP (needed for SMTP authentication)
Code = seeIntegerParam(0,  SEE_ENABLE_ESMTP, 1);
```

BASIC Example

```
' enable Extended SMTP (needed for SMTP authentication)
Code = seeIntegerParam(0,  SEE_ENABLE_ESMTP, 1)
```

2.42 **seeIsConnected**: Determine if still connected to server.

SYNTAX

```
seeIsConnected(Chan)

  Chan       : (I) Channel number.
```

REMARKS

The **seeIsConnected** function tests to see if there is a live connection to the email server.

RETURNS

- Return = 0 : No connectivity to server.

EXAMPLES

C/C++ Example

```
if(seeIsConnected(0)) printf("Connection is live");
else printf("Connection has been dropped");
```

BASIC Example

```
If seeIsConnected(0) <> 0 Then
  PRINT "Connection is live"
Else
  PRINT "Connection has been dropped"
End If
```

2.43 **seeKillProgram**: Terminates External Program.

SYNTAX

```
seeKillProgran(ProcessID, ExitCode)

  ProcessID : (I) Process ID (returned from seeStartProgram)
  ExitCode  : (P) Exit code.
```

REMARKS

The **seeKillProgram** function kills (terminates) the external program (process) that was started by **seeStartProgram**. The ProcessID is the process returned by **seeStartProgram**.

RETURNS

- Return < 0 : Cannot kill program.

EXAMPLES

C/C++ Example

```
int hProcess;
// kill STUNNEL
Code = seeKillProgram(hProcess, 0);
```

BASIC Example

```
Dim hProcess As Long
' kill STUNNEL
Code = seeKillProgram(hProcess, 0)
```

ALSO REFER TO

seeStartProgram

2.44 **seeMakeSubject**: Make quoted subject string.

SYNTAX

```
seeMakeSubject(Text, Code, Ptr, Len)

  Text : (P) Text string containing 8-bit ISO characters.
  Code : (I) Page code.
  Ptr  : (P) pointer to subject string buffer
  Len  : (I) size of subject string buffer
```

where "Code" above is one of: QUOTED_ISO_8859_1, QUOTED_ISO_8859_2, QUOTED_ISO_8859_3, QUOTED_ISO_8859_4, QUOTED_ISO_8859_7, QUOTED_ISO_8859_8, QUOTED_WIN_1250, QUOTED_WIN_1252, QUOTED_WIN_1255, QUOTED_UTF8.

REMARKS

The **seeMakeSubject** function creates a quoted ISO or UTF8 string ready to use as the email subject. The text passed to this function (as the first argument) should contain either ISO or UTF8 characters.

RETURNS

- Return > 0 : length of string created in subject buffer
- Return < 0 : see error list.

EXAMPLES

C/C++ Example

```
char *Text = "Ce message est écrit en français.";
char Subject[256];
// construct quoted subject string in Subject[]
Code = seeMakeSubject(Text, QUOTED_ISO_8859_1, (char *)Subject, 256);
```

BASIC Example

```
Dim Text As String
Dim Subject As String
Text = "Ce message est écrit en français."
Subject = Space(256)
' construct quoted subject string in Subject[]
Code = seeMakeSubject(Text, QUOTED_ISO_8859_1, Subject, 256)
```

ALSO REFER TO

seeQuoteBuffer

2.45 **seePop3Connect**: Connects to POP3 Server.

SYNTAX

```
seePop3Connect(Chan, Pop3Ptr, UserPtr, PassPtr)

  Chan    : (I) Channel number.
  Pop3Ptr : (P) POP3 server name.
  UserPtr : (P) POP3 user name.
  PassPtr : (P) POP3 password.
```

REMARKS

The **seePop3Connect** function establishes a connection with the POP3 server as specified by the Server argument.

The POP3 server name will typically be named "mail.XXX.com" where XXX is your email address, such as name@XXX.com. Your POP3 server name can also be found in the setup information for your normal email client, such as Eudora or Microsoft Outlook.

The POP3 server name can also be specified in dotted decimal notation. For example, "10.23.231.1".

SEE can not be connected to both the SMTP server and the POP3 server at the same time. Call **seeClose** to terminate the connection before connecting again

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// connect to POP3 server "mail.yourisp.com"
char Pop3Host[] = "mail.yourisp.com";
char Pop3User[] = "bill";
char Pop3Pass[] = "abc";
Code = seePop3Connect(0, (char *)Pop3Host, (char *)Pop3User, (char *)Pop3Pass);
```

BASIC Example

```
' connect to POP3 server "mail.yourisp.com"
Dim Pop3Host, Pop3User, Pop3Pass As String
Pop3Host = "mail.yourisp.com"
Pop3User = "bill"
Pop3Pass = "abc"
Code = seePop3Connect(0, Pop3Host, Pop3User, Pop3Pass)
```

ALSO REFER TO

seeSmtpConnect and **seeClose**; **STATUS** and **READER** example programs.

2.46 seePop3ConnectSSL: Connects to POP3 Server with SSL

SYNTAX

```
seePop3ConnectSSL(Chan, ProxyPort, Pop3Port, Pop3Server,
                  Pop3User, Pop3Pass, ProxyIP)

        Chan : (I) Channel number.
   ProxyPort : (I) Proxy port (to contact proxy on).
    Pop3Port : (I) POP3 port (normally 995)
  Pop3Server : (P) POP3 Server name
    Pop3User : (P) POP3 user name.
    Pop3Pass : (P) POP3 password.
    ProxyIP  : (P) Host name or IP address of proxy server
```

REMARKS

The **seePop3ConnectSSL** function is used to connect to a POP3 server with SSL. Note that **seePop3ConnectSSL** uses the same arguments as **seeImapConnectSSL**.

'ProxyPort' can be an unused port. 'Pop3Port' variable will normally be 995. 'Pop3Server' is the name or IP address of the POP3 server. 'Pop3User' is the user name and 'Pop3Pass' is the password. 'ProxyIP' is the host name or IP address of the proxy server. Pass NULL or an empty string to specify this computer (127.0.0.1).

seeSetProxySSL must be called before calling **seePop3ConnectSSL**.

Stunnel must first be installed. **seePop3ConnectSSL** will automatically start and stop Stunnel as needed. To set up Stunnel, see "Using Stunnel" in the SEE User's Manual (SEE_USR.PDF) or online at http://www.marshallsoft.com/stunnel.htm.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// connect to POP3 server "pop.gmail.com"
char *Pop3Server = "pop.gmail.com"
char *Pop3User = "superman@gmail.com";
char *Pop3Pass = "sorry";
Code = seePop3ConnectSSL(0, 8802, 995,Pop3Server,Pop3User, Pop3Pass,NULL);
```

BASIC Example

```
' connect to POP3 server "pop.gmail.com"
Dim Pop3Server,Pop3User,Pop3Pass As String
Pop3Server = "pop.gmail.com"
Pop3User = "superman@gmail.com"
Pop3Pass = "sorry"
Code = seePop3ConnectSSL(0, 8802,995, Pop3Server,Pop3User, Pop3Pass,Chr(0))
```

ALSO REFER TO

seeSmtpConnectSSL and **ReadSSL** example program.

2.47 **seePop3Source**: Specified file from which to read undecoded email

SYNTAX

```
seePop3Source(Chan, Pop3Filename)

  Chan         : (I) Channel number.
  Pop3Filename : (P) POP3 filename.
```

REMARKS

The **seePop3Source** function is used to specify the path to a file containing an undecoded email. After calling this function, **seeGetEmailFile** can be called to decode the email as if it were being downloaded from a POP3 server.

Note that there is no POP3 connection.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// decode email file
char Source[] = "c:\\see4c\\apps\\raw_email.txt";
Code = seePop3Source(0, (char *)Source);
```

BASIC Example

```
'decode email file
Dim Source As String
Source = "c:\see4c\apps\raw_email.txt"
Code = seePop3Source(0, Source)
```

ALSO REFER TO

seePop3Connect; **POP3RD** example program.

2.48 seeQuoteBuffer: Constructs ISO-8859 String.

SYNTAX

```
seeQuoteBuffer(String, Buffer, BufLen)

  String : (P) ISO 8859 text.
  Buffer : (P) Buffer for ISO-8859 encoded string.
  BufLen : (I) Size of above buffer.
```

REMARKS

The **seeQuoteBuffer** function creates an ISO-8859 encoded string in 'Buffer' from the ISO-8859 (8-bit character) ISO 8859 text in 'String'. The buffer length (3rd parameter) should be twice the size of the length of the string (1st parameter).

The primary use for the **seeQuoteBuffer** function is in constructing ISO-8859 compliant "Subject:" headers.

The default delimiter used to separate email addresses and path names was changed from a comma to a semi-colon in Version 3.6 of **SEE**.

RETURNS

- Return > 0 : The number of characters copied to 'Buffer'
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// construct quoted subject string, identifying it as quoted iso-8859-1
char Text[] = "Ce message est écrit en français";
char Buffer[128];
Code = seeQuoteBuffer((char *)Text, (char *)Buffer, 128)
```

BASIC Example

```
' construct quoted subject string, identifying it as quoted iso-8859-1
Dim Text As String
Dim Buffer As Buffer
Text = "Ce message est écrit en français"
Buffer = Space(128)
Code = seeQuoteBuffer(Text, Buffer, 128)
```

ALSO REFER TO

seeUnQuoteBuffer and **ISO8859** example program.

2.49 seeReadQuoted: Quotes File Contents.

SYNTAX

```
seeReadQuoted(PathName, Buffer, BufLen, Width)

  PathName : (P) Pathname of file to be read.
    Buffer : (P) Buffer into which data is written.
    BufLen : (I) Size of above buffer.
     Width : (I) Width of quoted lines.
```

REMARKS

The **seeReadQuoted** function reads the file specified by 'PathName' and creates a quoted text string in 'Buffer' The resulting quoted string will consists of multiple (quoted) lines of the length specified by 'Width'. The value of 'Width' must be less than 254. Use 0 to specify that the default width (73 chars) be used.

The primary use for the **seeReadQuoted** function is in the construction of non-standard email messages that must be quoted, such as EDIFACT (Electronic Data Interchange for Administration, Commerce, and Transport) email.

RETURNS

- Return > 0 : The number of characters copied to 'Buffer'
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
// read file & write (quoted) to 'Buffer'
char FileName[] = "EDIFACT.txt";
char Buffer[128];
Code = seeReadQuoted((char *)FileName, (char *)Buffer, 128, 0)
```

BASIC Example

```
' read file & write (quoted) to 'Buffer'
Dim FileName As String
Dim Buffer As String
FileName = "EDIFACT.txt"
Buffer = Space(128)
Code = seeReadQuoted(FileName, Buffer, 128, 0)
```

ALSO REFER TO

seeEncodeBuffer

2.50 **seeRelease**: Releases SEE.

SYNTAX

```
seeRelease
```

REMARKS

The **seeRelease** function releases the **SEE** system. **SeeRelease** is called only once and should be the very last **SEE** function called.

seeClose should be called for all channels before calling **seeRelease**.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// release SEE
seeRelease();
```

BASIC Example

```
' release SEE
Code = seeRelease()
```

Most of the example programs call **seeRelease**.

ALSO REFER TO

seeAttach.

2.51 **seeSendEmail**: Sends email and attachments.

SYNTAX

```
seeSendEmail(Chan, To, CC, BCC, Subj, Msg, Attach)

  Chan   : (I) Channel number.
  To     : (P) Recipient, separated by semi-colons.
  CC     : (P) CC list, separated by semi-colons.
  BCC    : (P) BCC list, separated by semi-colons.
  Subj   : (P) Subject text.
  Msg    : (P) Message or message filename.
  Attach : (P) File attachment list.
```

REMARKS

The **seeSendEmail** function is used to send email once a connection has been made to your SMTP server after calling **seeSmtpConnect**. Note that all email addresses (in To, CC, and BCC strings) must be bracketed, and the CC and BCC strings may contain multiple email addresses, separated by semi-colons. For example:

```
<info@marshallsoft.com>
"Billy Bob<bbob@isp.com>;Buster<bm@isp.com>"
```

If the first character of the message (sixth argument) is a '@', then it is considered as the filename which contains the message to send.

'Attach' may contain one or more attachments, separated by semi-colons, with no embedded spaces. For example,

```
"file1.zip;file2.doc;file3.bmp"
```

The default delimiter used to separate email addresses and path names was changed from a comma to a semi-colon in Version 3.6 of **SEE**. The semi-colon delimiter can be changed to any character with:

```
seeIntegerParam(Chan, SEE_ADDRESS_DELIMITER, new-character)
seeIntegerParam(Chan, SEE_ATTACH_DELIMITER, new-character)
```

If the function

```
seeIntegerParam(Chan, SEE_ENABLE_IMAGE, 1)
```

has been called previously, attachment files ending with ".GIF", ".BMP", or ".TIF" are attached as image types rather than regular images. This allows some email clients to display the images.

The **seeSendEmail** function is used with SMTP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// email file Message.txt to info@yourisp.com
char To[] = "<info@yourisp.com>"
char CC[] = "";
char BCC[] = "";
char Subject[] = "Test"
char Message[] = "@Message.txt"
char Attach[] = "";
Code = seeSendEmail(0, (char *)To, (char *)CC, (char *)BCC,
      (char *)Subject, (char *)Message, (char *)Attach);
```

BASIC Example

```
' email file Message.txt to info@yourisp.com
Dim To, CC, BCC, Subject, Message, Attach
To = "<info@yourisp.com>"
CC = Chr$(0)
BCC = Chr$(0)
Subject = "Test"
Message = "@Message.txt"
Attach = Chr$(0)
Code = seeSendEmail(0, To, CC, BCC, Subject, Message, Attach)
```

ALSO REFER TO

seeSendHTML; **MAILER** example program.

2.52 **seeSendHTML**: Sends HTML encoded email and attachments.

SYNTAX

```
seeSendHTML(Chan, To, CC, BCC, Subject, Message,
            Images, AltText, Attach)

  Chan   : (I) Channel number.
  To     : (P) Recipient, separated by semi-colons.
  CC     : (P) CC list, separated by semi-colons.
  BCC    : (P) BCC list, separated by semi-colons.
  Subject: (P) Subject text.
  HTML   : (P) HTML message text or [@filename]
  Images : (P) List of embedded images
  AltText: (P) Alternate text
  Attach : (P) File attachment list.
```

REMARKS

The **seeSendHTML** function is used to send HTML encoded email. See the entry for 'seeSendEmail' for an explanation of the To, CC, and BCC fields.

If the first character of the message (6th argument) or alternate text (8th argument) is a '@', then it is considered as the filename which contains the message to send.

The 'Images' field contains the filenames of images that are to be embedded in the email message. The first image must be referenced in the text of the HTML encode email message as

```
<IMG SRC="cid:message-root.1">
```

The second image (if any) must be referenced as

```
<IMG SRC="cid:message-root.2">
```

Continue in this way for all embedded images.

'AltText' is used to provide a plain ASCII text equivalent of the message for those email clients that cannot decode HTML.

'Attach' may contain one or more attachments, separated by semicolons, with no embedded spaces. For example,

```
"file1.zip;file2.doc;file3.bmp"
```

The semi-colon delimiter used above can be changed to a new character with:

```
seeIntegerParam(Chan, SEE_PATH_DELIMITER, new-character)
```

The **seeSendHTML** function is used with SMTP servers only.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

C/C++ Example

```
// email file html.htm to <info@yourisp.com>
char To[] = "<info@yourisp.com>"
char CC[] = "";
char BCC[] = "";
char Subject[] = "HTML Test"
char File[] = "@html.htm"
char Images[]  = "image1.gif;image2.gif"
char AltText[] = "@AltText.txt";
char Attach[] = "";
Code = seeSendHTML(0, (char *)To, (char *)CC, (char *)BCC,
                      (char *)Subject, (char *)File, (char *)Images,
                      (char *)AltText, (char *)Attach);
```

BASIC Example

```
' email file html.htm to <info@yourisp.com>
Dim To, CC, BCC, Subject, Message, Images, AltText, Attach
To = "<info@yourisp.com>"
CC = Chr$(0)
BCC = Chr$(0)
Subject = "HTML Test"
File = "@html.htm"
Images  = "image1.gif;image2.gif"
AltText = "@AltText.txt"
Attach = Chr$(0)
Code = seeSendHTML(0, To, CC, BCC, Subject, File, Attach)
```

ALSO REFER TO

SENDHTML (C/C++) or **HTML** example program.

2.53 seeSetCertAuth:: Set Certificate Authority (CA) File

SYNTAX

```
seeSetCertAuth(CA_Dir, CA_File))

  CA_Dir  : (P) CA directory
  CA_File : (P) CA file in PEM format
```

REMARKS

The **seeSetCertAuth** function specifies the directory CA_Dir in which the certificates file CA_File is located. The certificate authority (CA) file contains multiple CA certificates in PEM format used by Stunnel in authenticating the client.

The CA_Dir may be NULL or an empty string if CA_File contains the full path for the CA certs file CA_File.

The purpose of specifying a CA (certificate authority) file is so that Stunnel can verify the peer certificate chain starting from the root CA.

This function requires Stunnel version 5.35 or later.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

C/C++ Example

```
char *CA_Dir = "c:\\SEE4C\\SSL";
char *CA_File = "ca_certs.pem";
seeSetCertAuth(CA_Dir, CA_File);
```

BASIC Example

```
Dim CA_Dir As String
Dim CA_File As String
CA_Dir = "c:\SEE4C\SSL"
CA_File = "ca_certs.prm"
seeSetCertAuth(CA_Dir, CA_File)
```

2.54 seeSetErrorText:: Specifies Text of Error Messages

SYNTAX

```
seeSetErrorText(Chan, ErrorCode, ErrorText)

  Chan      : (I) Channel number.
  ErrorCode : (I) Error code.
  ErrorText : (P) Error text.
```

REMARKS

The **seeSetErrorText** function specifies the text to be used for a particular error code. The primary reason for this function is to support error messages in a language other than English.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

C/C++ Example

```
seeSetErrorText(SEE_NO_ERROR,  (char *)"pas d'erreur");
```

BASIC Example

```
Dim Text As String
Text = "pas d'erreur"
Code = seeSetErrorText(SEE_NO_ERROR, Text)
```

2.55 seeSetFileBuffer:: Passes buffer (not file) as Attachment

SYNTAX

```
seeSetFileBuffer(Chan, Index, BufAdr, BufLen)

  Chan  : (I) Channel number.
  Index : (I) index value (0,1,...,7) in file buffer table
  Addr  : (P) Address of data to attach
  Size  : (I) number of bytes to pass
```

REMARKS

The **seeSetFileBuffer** function passes data (rather than a filename) as an attachment.

Up to 8 attachment data buffers can be specified, numbered 0 to 7.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

C/C++ Example

```
// Attachment name
// The '*' signifies that this filename will use a data buffer #0
char AttachName[] = "MyFile.txt*0";
// attachment data
char AttachData[] = "Text or binary data";
// pass address of attachment data to SEE
Index = 0; // attachment name should end in "*0"
Code = seeSetFileBuffer(0, Index, &AttachData[0], strlen(AttachData) );
```

BASIC Example

```
' Attachment name & data
Dim AttachName As String
Dim AttachData As String
' The '*' signifies that this filename will use a data buffer
AttachName = "MyFile.txt*0"
AttachData = "Text or binary data"
' pass address of attachment data to SEE
Index = 0 ' attachment name should end in "*0"
Code = seeSetFileBuffer(0, Index, AttachData, Len(AttachData) )
```

2.56 seeSetProxySSL:: Specifies Proxy Server Parameters

SYNTAX

```
seeSetProxyAutoSSL(ProxyCode, ProxyFlags, ProxyDir, ProxyCert,
                   ProxyExe)

  ProxyCode : (I) proxy code (not used)
  ProxyFlags: (I) proxy server flags
  ProxyDir  : (P) proxy directory (on this machine)
  ProxyCert : (P) proxy certificate
  ProxyExe  : (P) proxy executable (STUNNEL.EXE)
```

REMARKS

The **seeSetProxySSL** program sets parameters for the proxy server (Stunnel.exe) and must be called before calling **seeImapConnectSSL**, **seePop3ConnectSSL**, or **seeSmtpConnectSSL.**

Pass STUNNEL_TASKBAR_ICON (value 1) for 'ProxyFlags' if you want an icon for the proxy server displayed on the task bar. Pass STUNNEL_DISABLE_LOGGING (value 2) if you do not want a Stunnel log file created at runtime. For both, pass the sum STUNNEL_TASKBAR_ICON +STUNNEL_DISABLE_LOGGING (value 3).

The 'ProxyDir' argument is the path to the directory containing the proxy server.

'ProxyCert' is the X509 certificate in PEM format to use. Use the included certificate STUNNEL.PEM if you do not have your own PEM certificate.

'ProxyExe' is the name of the proxy executable (STUNNEL.EXE in this version of SEE).

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
char *ProxyDir = "c:\\SEE4C\\SSL\\";  // proxy server directory
char *ProxyExe = "stunnel.exe";       // proxy server executable
char *ProxyCert = "stunnel.pem";      // proxy certificate (PEM format)

Code = seeSetProxySSL(0,      // proxy code (always 0)
STUNNEL_TASKBAR_ICON,         // display Stunnel icon on task bar
ProxyDir,                     // directory containing proxy server
ProxyCert,                    // proxy certificate
ProxyExe);                    // proxy program (executable)
```

BASIC Example

```
ProxyDir = "c:\SEE4VB\SSL\"   ' proxy server directory
ProxyExe = "stunnel.exe"      ' proxy server executable

Code = seeSetProxySSL(0,      ' proxy code (always 0)
STUNNEL_TASKBAR_ICON,         ' display Stunnel icon on task bar
ProxyDir,                     ' directory containing proxy server
ProxyCert,                    ' proxy certificate
ProxyExe);                    ' proxy program (executable)
```

ALSO SEE

seeImapConnectSSL, **seePop3ConnectSSL**, and **seeSmtpConnectSSL**.

2.57 seeShortToByte :: Converts 16-bit ASCII character buffer to 8-bit

SYNTAX

```
seeShortToByte(Buffer)

 Buffer : (P) character buffer
```

REMARKS

The **seeShortToByte** function converts the (null terminated) character buffer 'Buffer' from 16-bit Unicode ASCII characters to 8-bit ASCII characters.

The buffer must be null terminated (last character is a hex 00).

This function is only necessary when working with 16-bit Unicode ASCII characters in C# and Delphi 2005.

RETURNS

- Length of string in words.

EXAMPLES

C/C++ Example

```
int Code;
// define 16-bit ASCII string
wchar_t UnicodeString[] = L"MarshallSoft";

// Convert 16-bit ASCII string to 8-bit ASCII
Code = seeShortToByte((char *)UnicodeString);
```

ALSO SEE

seeByteToShort

2.58 seeSleep: Sleeps Specified Milliseconds

SYNTAX

```
seeSleep(MilliSecs)

  MilliSecs : (I) Number of milliseconds to sleep
```

REMARKS

seeSleep is intended primarily for programming in languages that do not have a native Sleep function.

RETURNS

- Milliseconds slept.

EXAMPLES

C/C++ Example

```
// sleep 1 second
seeSleep(1000);
```

BASIC Example

```
' sleep 1 second
Code = seeSleep(1000)
```

2.59 seeSmtpConnect: Connects to SMTP server.

SYNTAX

```
seeSmtpConnect(Chan, Server, From, ReplyTo)

  Chan    : (I) Channel number.
  Server  : (P) SMTP server.
  From    : (P) Your email address in brackets.
  ReplyTo : (P) Email address to reply to.
```

REMARKS

The **seeSmtpConnect** function establishes a connection with the SMTP server as specified by the 'Server' argument.

Your SMTP server name will typically be named "mail.XXX.com" where XXX is your email address, such as name@XXX.com. Your SMTP server name can also be found in the setup information for your normal email client, such as Eudora or Microsoft Outlook.

The SMTP server name can also be specified in dotted decimal notation. For example, "10.23.231.1".

The 'From' string is required and must be enclosed in "<>" brackets, such as <you@yourisp.com>.

The 'ReplyTo' string is optional and is used for the "Reply-To:" header line. If used, the email address must be enclosed in "<>" brackets.

SEE can not be connected to both the SMTP server and the POP3 server at the same time. Call **seeClose** to terminate the connection before connecting again.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// connect to SMTP server
char Server[] = "smtp.yourisp.com"
char From[] = "<you@yourisp.com>"
char Reply[] = ""
Code = seeSmtpConnect(0, (char *)Server, (char *)From, (char *)Reply)
```

BASIC Example

```
' connect to SMTP server
Server = "smtp.yourisp.com"
From = "<you@yourisp.com>"
Reply = Chr$(0)
Code = seeSmtpConnect(0, Server, From, Reply)
```

ALSO REFER TO

seeClose; MAILER example program.

2.60 seeSmtpConnectSSL: Connects to SMTP Server with SSL.

SYNTAX

```
seeSmtpConnectSSL(Chan, ProxyPort, SmtpPort, SmtpServer,
                  SmtpUser, SmtpPass, SmtpFrom, SmtpReply, ProxyIP)

        Chan : (I) Channel number
   ProxyPort : (I) Port to connect to proxy on
    SmtpPort : (I) SMTP port(normally 465 or 587)
  SmtpServer : (P) SMTP server name
    SmtpUser : (P) User name
    SmtpPass : (P) Password
    SmtpFrom : (P) Sender's email address
   SmtpReply : (P) Reply-To email address
     ProxyIP : (P) Host name or IP address of proxy server
```

REMARKS

The **seeSmtpConnectSSL** function is used to connect to an SMTP server with SSL.

seeSetProxySSL must be called before calling **seeSmtpConnectSSL**.

'ProxyPort' can be an unused port. 'SmtpPort' for SSL will normally be 465 or 587. 'SmtpServer' is the name or IP address of the SMTP server. 'SmtpUser' is the user name and 'SmtpPass' is the password. The 'SmtpFrom' string is required and must be enclosed in "<>" brackets, such as <you@yourisp.com>. The 'SmtpReply' string is optional and is used for the "Reply-To:" header line. If used, the email address must be enclosed in "<>" brackets. 'ProxyIP' is the host name or IP address of the proxy server. Pass NULL or an empty string to specify this computer (127.0.0.1).

Stunnel must first be installed. **seeSmtpConnectSSL** will automatically start and stop Stunnel as needed. To set up Stunnel, see "Using Stunnel" in the SEE User's Manual (SEE_USR.PDF) or online at http://www.marshallsoft.com/stunnel.htm.

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
// connect to SMTP server "smtp.gmail.com" via port 8801
char *Server = "smtp.gmail.com"
char *User = "BillyBob@gmail.com";
char *Pass = "sorry";
char *From = "<BillyBob@gmail.com>"
char *Reply = ""
Code = seeSmtpConnectSSL(0,8801,465,Server,User,Pass,From,Reply,NULL)
```

BASIC Example

```
' connect to SMTP server "smtp.gmail.com" via port 8801
Server = "smtp.gmail.com"
User = "BillyBob@gmail.com"
Pass = "sorry"
From = "<BillyBob@gmail.com>"
Reply = Chr$(0)
Code = seeSmtpConnectSSL(0,8801,465,Server,User,Pass,From,Reply, Chr(0))
```

2.61 seeSmtpTarget: Specifies SMTP output file.

SYNTAX

```
seeSmtpTarget(FileName, EmailAddr, ReplyAddr)

  FileName  : (I) File to write SMTP output to
  EmailAddr : (I) Return email address
  ReplyAddr : (I) Reply-To address
```

REMARKS

seeSmtpTarget is called instead of **seeSmtpConnect** so that when **seeSendEmail** or **seeSendHTML** is called, the email is written to the specified file in RFC822 compliant format rather than sent to the server.

seeSmtpTarget is called instead of seeSmtpConnect.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
Code = seeSmtpTarget(0,
       (char *)"MyFile.txt",
       (char *)"<you@yourisp.com>",
       (char *)"<you@yourisp.com>");
```

BASIC Example

```
Code = seeSmtpTarget(0, "MyFile.txt",
                     "<you@yourisp.com>",
                     "<you@yourisp.com>")
```

ALSO REFER TO

seePop3Source

2.62 seeStartProgram: Starts External Program.

SYNTAX

```
seeStartProgram(CommandLine)

  CommandLine : (I) Command line for external program.
```

REMARKS

The **seeStartProgram** function starts the specified external program. The command line contains the pathname of the executable plus any additional command line arguments, if any. **seeStartProgram** can start any Win32 program.

The primary purpose of **seeStartProgram** is to start external programs such as proxy servers.

C/C++ Example

```
char Stunnel[]= "c:\\stunnel\\stunnel.exe c:\\stunnel\\SMTPgmail.txt";
int hProcess;
// Starting STUNNEL
hProcess = seeStartProgram((char *)Stunnel);
```

BASIC Example

```
Dim Stunnel As String
Dim hProcess As Integer
Stunnel = "c:\stunnel\stunnel.exe c:\stunnel\SMTPgmail.txt"
' Starting STUNNEL
hProcess = seeStartProgram(Stunnel)
```

RETURNS

- Return = -1 : Cannot start process.
- Return > 0 : Process ID

ALSO REFER TO

seeKillProgram

2.63 seeStatistics: Returns runtime statistics.

SYNTAX

```
seeStatistics(Chan, Index)

  Chan  : (I) Channel number.
  Index : (I) Specifies which statistic.
```

REMARKS

The **seeStatistics** function is used to return runtime statistics in the **SEE DLL**. The values of 'Index' are defined in the **SEE** declaration file (see Section 1.3 “Declaration Files”) as follows.

```
SEE_GET_ATTACH_BYTES_READ  : Gets attachment bytes read.
SEE_GET_ATTACH_BYTES_SENT  : Gets attachment bytes sent.
SEE_GET_ATTACH_COUNT       : Gets attachments received.
SEE_GET_BUFFER_COUNT       : Gets bytes in buffer for seeGetEmailLines.
SEE_GET_BUILD              : Gets version build number.
SEE_GET_CONNECT_STATUS     : Returns positive number if connected.
SEE_GET_COUNTER            : Gets times driver called.
SEE_GET_LAST_RECIPIENT     : Gets last recipient ack'd by server.
SEE_GET_MESSAGE_BYTES_READ : Gets message bytes read.
SEE_GET_MESSAGE_BYTES_SENT : Gets message bytes sent.
SEE_GET_MSG_COUNT          : Gets emails waiting.
SEE_GET_MSG_SIZE           : Gets size of email.
SEE_GET_RESPONSE           : Gets last SMTP response code.
SEE_GET_SOCK_ERROR         : Gets last socket error.
SEE_GET_SOCKET             : Gets socket number.
SEE_GET_TOTAL_BYTES_READ   : Gets total bytes read.
SEE_GET_TOTAL_BYTES_SENT   : Gets total bytes sent.
SEE_GET_VERSION            : Gets the SEE version number.
```

The number of message bytes sent will usually be larger than your message size because of SMTP protocol overhead.

The number of attachment bytes sent will be at least one-third larger than the actual attachment since every three (3) bytes are encoded as four (4) 7-bit ASCII bytes before being transmitted.

The purpose of "BYTES_READ" and "BYTES_SENT" is to provide the ability to track the transmission progress of large messages and attachments.

SEE_GET_ATTACH_BYTES_READ gets the number of attachment bytes read.

SEE_GET_ATTACH_BYTES_SENT gets the number of attachment bytes sent.

SEE_GET_ATTACH_COUNT gets the number of attachments received.

SEE_GET_BUFFER_COUNT gets the bytes in the buffer after calling **seeGetEmailLines**.

SEE_GET_BUILD gets the version build number.

REMARKS (continued)

`SEE_GET_CONNECT_STATUS` returns a positive number if currently connected.

`SEE_GET_COUNTER` gets the number of times that the driver has been called.

`SEE_GET_LAST_RECIPIENT` gets last recipient (1, 2,...) acknowledged by server.

`SEE_GET_MESSAGE_BYTES_READ` gets the number of message bytes read.

`SEE_GET_MESSAGE_BYTES_SENT` gets the number of message bytes sent.

`SEE_GET_MSG_COUNT` gets the number of emails waiting on the server.

`SEE_GET_MSG_SIZE` gets the size of an email message.

`SEE_GET_RESPONSE` gets the last server response code.

`SEE_GET_SOCK_ERROR` gets the last socket error.

`SEE_GET_SOCKET` gets the socket number.

`SEE_GET_TOTAL_BYTES_READ` gets the total number of bytes read.

`SEE_GET_TOTAL_BYTES_SENT` gets the total number of bytes sent.

`SEE_GET_VERSION` gets the **SEE** version number (see SEEVER example).

RETURNS

- Return < 0 : An error has occurred. See section 3 “SEE Error Return Code List”.

EXAMPLES

C/C++ Example

```
' get SEE version
 Code = seeStatistics(0, SEE_GET_VERSION);
```

BASIC Example

```
' get SEE version
 Code = seeStatistics(0, SEE_GET_VERSION)
```

ALSO REFER TO

seeDebug, seeIntegerParam, and **seeStringParam**; **READER, MAILER, BCAST, STATUS** examples.

2.64 **seeStringParam**: Sets SEE string parameter to control email processing.

SYNTAX

```
seeStringParam(Chan, ParamName, ParamString)

  Chan       : (I) Channel number.
  ParamName  : (I) Parameter.
  ParamString: (P) Parameter string.
```

REMARKS

The **seeStringParam** is used to set an string (text) parameter that is passed to the **SEE** library. The numeric value for each of the integer parameters is defined in the **SEE** declaration files. Section 1.3 “Declaration Files” provides a list the "Declaration Files".

```
SEE_ADD_HEADER        : Adds header.
SEE_LOG_FILE          : Specifies the log filename.
SEE_SET_CONTENT_TYPE  : Sets user defined content.
SEE_SET_CONTENT_TYPE_PREFIX  : Write prefix to Content-Type header.
SEE_SET_FILE_PREFIX   : Specifies file prefix character.
SEE_SET_FROM          : Sets "From:" header after connecting.
SEE_SET_HEADER        : Sets header line.
SEE_SET_IMAP_LIST_ARG : Species the IMAP list command.
SEE_SET_REPLY         : Sets the "Reply To" string.
SEE_SET_SECRET        : Sets password for SMTP authentication
SEE_SET_TRANSFER_ENCODING : Sets user defined transfer encoding.
SEE_SET_USER          : Sets user for SMTP authentication.
SEE_WRITE_BUFFER      : Writes internal buffer to disk.
SEE_WRITE_TO_LOG      : Write string to LOG file.
SEE_SET_FOOTER        : Append footer to all outgoing email.
SEE_SET_HELO_STRING   : Sets custom HELO / EHLO string.
```

SEE_IMAP_LIST_ARGUMENT is used to specify the IMAP argument used to request a list. The default is ["" "*"] (without the brackets). A common alternative is [~/ *] (without the brackets).

SEE_ADD_HEADER is used to add a user specified header line.

SEE_LOG_FILE specifies the name of the LOG file to create. The log file is used to debug a SMTP or POP3 session. Be advised that log files can be quite large. Don't use them unless necessary.

SEE_SET_CONTENT_TYPE is used to specify the content type string to use when enabling quoting with **seeIntegerParam**(Chan, SEE_QUOTED_PRINTABLE, QUOTED_USER)

SEE_SET_FROM is used to specify the "From:" header line after connecting to the SMTP server.

SEE_SET_HEADER is used to set one or more header lines. Each header line except the last should end with a carriage return line feed pair.

SEE_SET_REPLY to change the "Reply-To:" header after connecting to the server, just before sending an email.

SEE_SET_SECRET is used to specify the user's password for ESMTP authentication.

SEE_SET_TRANSFER_ENCODING is used to specify the content transfer encoding to use when enabling quoting with **SeeIntegerParam**(Chan, SEE_QUOTED_PRINTABLE, QUOTED_USER)

SEE_SET_USER is used to specify the user name for ESMTP authentication.

SEE_WRITE_BUFFER is used to write the internal buffer (created by **seeGetEmailLines**) to disk. See the GETRAW example program.

SEE_WRITE_TO_LOG is used to write text to the LOG file.

SEE_SET_CONTENT_TYPE_PREFIX specifies a string that is prefixed to the Content-Type header line that is written to the email output file (POP3 connection) provided that SEE_WRITE_CONTENT_TYPE is set to TRUE.

SEE_SET_FILE_PREFIX is used to specify the character (default value '@') that is used to specify (in the seeSendEmail function) that the message text is a filename rather than the actual text of the email message. Pass an empty string (null character) to disable (message text is never a filename).

SEE_SET_FOOTER is used to append footer text (up to 256 chars) to all outgoing email.

SEE_SET_HELO_STRING is used to set a custom HELO / EHLO string used when connecting to a SMTP server.

RETURNS

- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
' specify SEE log file name
char LogFile[] = "log.txt";
Code = seeStringParam(0, SEE_LOG_FILE, (char *)LogFile)
```

BASIC Example

```
' specify SEE log file name
Dim LogFile As String
LogFile = "log.txt"
Code = seeStringParam(0, SEE_LOG_FILE, LogFile)
```

ALSO REFER TO

seeIntegerParam.

2.65 seeTestConnect: Test connection To Server

SYNTAX

```
seeTestConnect(Chan, Server, Port, BufPtr, BufLen)

  Chan   : (I) Channel number
  Server : (P) Server name
  Port   : (I) Port
  BufPtr : (P) Buffer to receive greeting message
  BufLen : (I) Length of above buffer
```

REMARKS

The **seeTestConnect** function is used to test that a SMTP, POP3, or IMAP server is accepting connections on the specified port. Note that a user name and password is not used until after the server has accepted the connection.

When attempting to configure parameters for a new email account, the first step is to determine the correct server name and port. Once this is done, the second step is to determine the correct user name, password, and if authentication is required.

RETURNS

- Return > 0 : Connection accepted. Length of greeting message copied into 'BufPtr'.
- Return = 0 : Connection accepted. Greeting not available from TLS/SSL servers.
- Return < 0 : An error has occurred. See section 3 "SEE Error Return Code List".

EXAMPLES

C/C++ Example

```
char *Server = "smtp.gmail.com";
char Buffer[256];
// connect to server
Code = seeTestConnect(0, (char *)Server, 587, (char *)Buffer, 256);
```

BASIC Example

```
Dim Server As String
Dim Buffer As String
Server = "smtp.gmail.com";
Buffer = Space(256)
// connect to server
Code = seeTestConnect(0, Server, 587, Buffer, 256)
```

ALSO REFER TO

seeSmtpConnect, seePop3Connect, seeImapConnect
seeSmtpConnectSSL, seePop3ConnectSSL, seeImapConnectSSL

2.66 seeTestFileSet: Test Files for Existence.

SYNTAX

```
seeTestFileSet(FileSet, Buffer, BufLen)

  FileSet : (P) List of files to test.
  Buffer  : (P) Buffer for filename if it cannot be opened.
  BufLen  : (I) Size of above buffer (should be >= 256 bytes)
```

REMARKS

The **seeTestFileSet** function is used to verify that each file in the (comma or semicolon delimited) list of files can be opened for read access. This function provides an easy way to test that message and attachment files exist and can be opened by SEE.

If all the files in the string 'FileSet' can be opened, then **seeTestFileSet** returns a 0. Otherwise, the filename of the first file that cannot be opened is copied to 'Buffer' and the length of the filename is returned.

Filenames in 'FileSet' must be separated by either commas or semicolons.

RETURNS

=0 : All files can be opened.
>0 : Length of filename in 'Buffer' that cannot be opened.

EXAMPLES

C/C++ Example

```
int Code;
char Buffer[256];
char *FileSet= "c:\\see4c\\apps\\image1.gif;c:\\see4c\\apps\\image2.gif";
Code = seeTestFileSet(FileSet, (char *)Buffer, 256);
if(Code>0) printf("%s cannot be opened\n", Buffer);
```

BASIC Example

```
Dim Code As Integer;
Dim Buffer As String
Dim FileSet As String
FileSet= "c:\see4vb\apps\image1.gif;c:\see4vb\apps\image2.gif"
Buffer = Space(256)
Code = seeTestFileSet(FileSet, Buffer, 256)
If Code > 0 Then
  Buffer = Left(Buffer, Code)
  MsgBox(Buffer + " cannot be opened");
```

2.67 seeUnquoteBuffer: Unquote Buffer

SYNTAX

```
seeUnquoteBuffer(SrcBuffer,  DstBuffer, DstLength)

  SrcBuffer : (P) Source string (to unquote).
  DstBuffer : (P) Destination buffer
  DstLength : (I) Size of DstBuffer
```

REMARKS

The seeUnquoteBuffer function "unquotes" quoted strings in email messages. Although "unquoting" is done automatically by SEE, there are situations where an unquote function may be needed.

The size of the destination buffer (DstBuffer) should be at least as big as the size of the source string (SrcBuffer).

RETURNS

TRUE (non zero) if verified by server.

EXAMPLES

C/C++ Example

```
char *Quoted = "Ce message est =E9crit en fran=E7ais.";
char Buffer[256];
Code = seeUnQuoteBuffer(Quoted, (char *)Buffer, 255);
```

BASIC Example

```
Dim Quoted As String
Dim Buffer As String
Quoted = "Ce message est =E9crit en fran=E7ais."
Buffer = SPACE(256)
Code = seeUnQuoteBuffer(Quoted, Buffer, 255)
```

ALSO REFER TO

seeQuoteBuffer and the **iso8859** example program.

2.68 seeVerifyFormat: Check email address format.

SYNTAX

```
seeVerifyFormat(String)

  String : (P) Email address to check.
```

REMARKS

The **seeVerifyFormat** function is used to test an individual email address for proper formatting. If this function returns 0 or greater, then the email address is properly formatted. But, if this function returns a negative value, then the email address is either badly formatted, or it uses characters (such as '%') that are not normally used as part of an email address.

Note that left and right brackets ('<' and '>') must surround the email address.

RETURNS

TRUE (non zero) if verified.

EXAMPLES

C/C++ Example

```
// verify email address format
char EmailAddr[] = "<you@yourisp.com>";
Code = seeVerifyFormat((char *)EmailAddr);
```

BASIC Example

```
' verify email address format
Dim EmailAddr As String
EmailAddr = "<you@yourisp.com>"
Code = seeVerifyFormat(EmailAddr)
```

ALSO REFER TO

seeErrorText and **seeVerifyUser**; **VERUSR** example program.

2.69 seeVerifyUser: Verify email address with SMTP server.

SYNTAX

```
seeVerifyUser(Chan, String)

  Chan   : (I) Channel number.
  String : (P) Email address to verify.
```

REMARKS

The **seeVerifyUser** function is used to verify an individual email address with the email server which "owns" the email address.

seeVerify will connect to the specified server and request verification of the user. Many SMTP servers may refuse connection of any client not directly connected to them or may refuse all "verify user" requests. Web based email servers (hotmail, gmail, yahoo mail, etc.) typically will not honor any "verify user" requests.

Note that the connection must be to the SMTP server that owns the email address rather than the SMTP server normally used to send email.

The **seeVerifyUser** function is used with SMTP servers only.

RETURNS

TRUE (non zero) if verified by server.

EXAMPLES

C/C++ Example

```
// verify user "billy" on connected POP3 server "yourisp.com"
char User[] = "billy";
Code = seeVerifyUser(0, (char *)User);
```

BASIC Example

```
' verify user "billy" on connected POP3 server "yourisp.com"
Dim User As String
User = "billy"
Code = seeVerifyUser(0, User)
```

ALSO REFER TO

seeErrorText and **seeDebug; VERUSR** example program.

3. SEE Error Return Code List

The complete list of SEE error codes follows. These error messages can also be found by calling the **seeErrorText** function.

```
  0 SEE_CANNOT_COMPLY         Cannot comply. Not always an error.
  1 SEE_NO_ERROR              No error.
 -1 SEE_EOF                   End of file (socket has been closed).
 -4 SEE_IS_BLOCKING           Socket is currently blocking.
 -7 SEE_INVALID_SOCKET        Invalid socket.
 -8 SEE_TIMED_OUT             Socket timed out awaiting data.
 -9 SEE_NO_SOCK_ADDR          No socket address.
-12 SEE_NO_HOST               No host name.
-14 SEE_ABORTED               The DLL has been corrupted.
-18 SEE_CANNOT_CREATE_SOCK    Cannot create socket.
-30 SEE_ALREADY_CONNECTED     Already connected to server.
-31 SEE_BACK_OVERFLOW         Response buffer has overflowed.
-32 SEE_BAD_ADDRESS_CHAR      Bad character in email address.
-34 SEE_CANNOT_ATTACH         Cannot access DLL.
-35 SEE_CANNOT_OPEN           Cannot open file (for read).
-36 SEE_CONNECT_ERROR         Error attempting to connect.
-37 SEE_EMPTY_ADDRESS         EMPTY email address.
-38 SEE_FROM_NULL_ARG         Required 'From:' argument is NULL.
-39 SEE_MISSING_AT_CHAR       Missing '@' character in email address.
-40 SEE_MISSING_FROM          Missing 'From:' email address.
-41 SEE_MISSING_LEFT          Missing '<' delimiter in email address.
-43 SEE_MISSING_RIGHT         Missing '>' terminating email address.
-44 SEE_NOT_CONNECTED         Not connected to server.
-45 SEE_NO_RECIPIENTS         Must have at least one recipient.
-46 SEE_NO_SERVER             Cannot find SMTP/POP3/IMAP server.
-47 SEE_NULL_POINTER          Unexpected NULL pointer.
-49 SEE_SMTP_ERROR            SMTP returned error.
-50 SEE_EMAIL_NULL_ARG        SMTP/POP3/IMAP server not specified.
-51 SEE_SOCK_READ_ERROR       Socket read error.
-52 SEE_SOCK_WRITE_ERROR      Socket write error.
-53 SEE_TOO_MANY_AT_CHARS     Too many '@' symbols in email address.
-55 SEE_CANNOT_ALLOC          Cannot allocate memory.
-56 SEE_NOT_SERVER,           Illegal chars in server name.
-57 SEE_NO_APOP_TIMESTAMP     POP3 server did not provide a timestamp.
-58 SEE_SMTP_ONLY             Must be connected to SMTP server.
-59 SEE_POP3_ONLY             Must be connected to POP3 server.
-60 SEE_OBSOLETE_PARAMETER    Parameter is obsolete.
-61 SEE_USER_NULL_ARG         Expected USER name not specified.
-62 SEE_PASS_NULL_ARG         Required POP3 password argument missing.
-63 SEE_POP3_ERROR            Error returned by POP3 server.
-64 SEE_MSG_NBR_RANGE         Message number out of range.
-65 SEE_FILENAME_NULL_ARG     Required filename is missing.
-66 SEE_EMAIL_PATH_NULL_ARG   Required file path is missing.
-67 SEE_CANNOT_CREATE         Cannot create file.
-68 SEE_BUFFER_NULL_ARG       Required buffer is missing.
-69 SEE_BUFFER_SIZE_ARG       Buffer size argument is not positive.
```

SEE Error Return Code List - continued

```
-70 SEE_ATTACH_PATH_NULL_ARG Attachment argument is missing.
-71 SEE_NOT_ATTACHED         Must call seeAttach first.
-72 SEE_ALREADY_ATTACHED     seeAttach already called.
-73 SEE_CHAN_OUT_OF_RANGE    Channel number out of range.
-74 SEE_BAD_KEY_CODE         Bad keycode (2nd argument in seeAttach)
-75 SEE_NO_SUCH_FILE         No such file.
-76 SEE_PATH_NOT_ALLOWED     Filename only - path not allowed.
-77 SEE_NO_SUCH_PATH         No such path.
-78 SEE_IMAP_ERROR,          IMAP returned error.
-79 SEE_IMAP_ONLY,           IMAP function ONLY.
-80 SEE_POP3_IMAP_ONLY,      POP3 or IMAP function ONLY.
-81 SEE_IMAP_FLAG_ERROR,     IMAP flag error.
-82 SEE_IMAP_SEARCH_ERROR,   IMAP search error.
-83 SEE_BUFFER_OVERFLOW      Buffer overflow.
-84 SEE_PROXY_START_FAILS    Could not start proxy server.
-85 SEE_PROXY_NOT_CONFIGURED Proxy server not configured.
-86 SEE_NO_SUCH_OPTION       No such option.
-87 SEE_HEADER_NOT_ALLOWED   User not allowed to set system headers.
-88 SEE_NO_SUCH_CODE         No such page code.
-98 SEE_EXPIRED              Evaluation version expired.
-99 SEE_INTERNAL_ERROR       Internal SEE error.
```