

# A-Shell Tech Note 1003

## Sending Internet Mail with EMAILX.SBR

---

### Overview

EMAILX.SBX is an external XCALL subroutine which allows a A-Shell Basic program to send Internet email. This document describes its usage.

\* *The purpose of the A02 update to this document was to indicate the requirement for A-Shell 4.9 if using EMAILX.SBX edit 113 or higher, and also to discuss SMTP authentication.*

### Requirements

The current requirements are as follows:

- You must have at least A-Shell/UNIX 4.9 or A-Shell/Windows 4.9. This is because EMAILX uses TCPX.SBR, which was introduced in A-Shell 4.9. (The subroutine itself, however, is physically independent of A-Shell.) You may continue to use EMAILX.SBX edit 112 or before with previous versions of A-Shell.
- You must have an SMTP server which is within “routing distance.” That is, we need to be able to open a socket connection to your SMTP server. The SMTP server may be on the same machine as you, or on the local network, or at your ISP connected via a full-time or dial-up connection. Wherever it is, it must accept mail from you. (Some SMTP servers may have filters to protect “unauthorized users”.)
- As of edit 113, EMAILX does not support SMTP authentication. This is usually not an issue for sites with static IP addresses, as the ISP's SMTP service will likely authenticate you using your IP address. But for sites using dynamic addresses, more and more SMTP servers are requiring some form of authentication. EMAILX version 114 will support the "PLAIN" form of authentication. (Let us know if this is a requirement for you and we'll accelerate its release.)

## Licensing and Distribution Plan

A time-limited demo version is available on demand. Beyond that, we offer two licensing models:

- \$250 for a single-site license
- \$500 for an unlimited site distribution license. (This is intended for dealers who want to include EMAILX as part of their application.) The only requirement is that the subroutine be included with the dealer's custom software and not sold separately. This license also includes full source code.

## Calling Syntax and Parameters

EMAILX.SBR may be called as follows:

```
XCALL EMAILX,OPFLAGS,CFGFILE,TO,HEADER,BODY,STATUS &
      {,FROM,ATTACHMENTS,SIGNATURE,ERRMSG,SOCKET,SERVER, &
      HOST}
```

Where:

**OPFLAGS** specifies various operation flags, using the sum of one or more of the following (from EMAILX.MAP):

```
MAP1 EMF'OPFLAGS                ! EM'OPFLAGS symbols
MAP2 EMF'NORMAL,B,2,0           ! normal operation
MAP2 EMF'LEAVEOPEN,B,2,1       ! don't close session
MAP2 EMF'ALREADYOPEN,B,2,2     ! don't open; (already open)
MAP2 EMF'OPENONLY,B,2,4        ! open only; ignore all but
                                ! SERVER & HOST
MAP2 EMF'CLOSEONLY,B,2,8       ! close only
MAP2 EMF'HDRCAT,B,2,16         ! concat HEADER+BODY(no CRLF)
MAP2 EMF'HTMLBODY,B,2,32       ! BODY contains <HTML>
```

Note that in most cases you can get by with EMF'NORMAL. The next four flags (EMF'LEAVEOPEN, EMF'ALREADYOPEN, EMF'OPENONLY, and EMF'CLOSEONLY) are only of interest if you are sending a lot of emails in a batch and want to eliminate the overhead of having to open and close the connection with the SMTP server for each message. EMF'HDRCAT is only used when you want to include email header items in both the **HEADER** and **BODY** parameters. And EMF'HTMLBODY is only recommended when you have used your own HTML formatting within the body of your message.

**CFGFILE**:(string, may be null) specifies the filename (AMOS or native) of a configuration file containing additional parameters of the type that are thought likely to vary by installation but otherwise remain fixed for that installation. See the subsequent section describing the configuration file in more detail. Note that all of the parameters after **STATUS** are optional on the XCALL, but only to the extent

that the the corresponding parameter is defined in the configuration file. Alternatively, you can omit the configuration file entirely, in which case you must specify all of the parameters on the XCALL line.

**TO** (string) specifies the email address(es) of recipient(s), each using standard unadorned email address format. (e.g. jack@microsabio.com). If you are specifying multiple addresses, separate them with chr(13) characters. (Up to 100 addresses allowed.)

Note that if you plan to use the CC: or BCC: header items to send copies of the message to other people, those people must be listed here in the **TO** parameter. (This parameter is strictly used by the SMTP server for routing and does not (usually) show in the recipient's message header.)

**HEADER** (string) specifies zero or more header fields, separated by CRLF (i.e. chr(13)+chr(10)) and each conforming to the formatting rules in the appropriate RFC's (such as RFC 822 and RFC 1521). EMAILX does not check your header items for validity. Note that you may decide to skip this parameter and put all of your header items in the body of the message (see **BODY**). Or you can put some here and some in the **BODY** parameter (provided you then specify the EMF 'HDRCAT flag in **OPFLAGS**.) Here are some sample header lines:

```
To      : "George Washington" <gw@cherrytree.com>
From    : "Paul Revere" <prevere@patriots.net>
Subject : The Redcoats are Coming
Reply-To: <admin@federalists.org>
cc      : <tj@montebello.net>
```

- \* *Although headers are technically part of the body of the message as far as the SMTP server is concerned, there is a core set of headers that SMTP servers, and especially email readers understand, plus an extended set that most understand, plus a specification for how to define your own headers. See the section on Header Specifications below for more details.*
- \* *The Internet email (and most other) standards are documented in a series of "RFC's" (Request For Comments) which fortunately are very accessible via the Internet. Just search for "RFC 822" for example in any decent search engine.*

**BODY** (string) specifies the body of the message, either directly by including the entire body in the variable (with each line terminated by a CRLF), or indirectly (by giving the name of a file which contains the body of the message.) If you are specifying a filename (either AMOS or native) rather than the body itself, this parameter must contain a string less than 128 characters long with no carriage returns. Otherwise, the parameter will be treated as the literal body of the message. Note that if the **HEADER** parameter was not blank, and the EMF 'HDRCAT flag was not specified, then a blank line will be automatically inserted into the message to separate the header from the body. (This is part of the standard mail format rules, but is so likely to be overlooked by Basic programmers that we do it automatically here.)

- \* *If you include HTML formatting within the body of the message, you should specify the EMF 'BODYHTML flag in the OPFLAGS parameter. Most modern email readers are smart enough to figure it out for themselves, but some older ones may fail to recognize HTML formatting without a "Content-Type: text/html" header, which EMAILX will add for you if you specify the flag.*

**STATUS** (F,6) Return code. 0=success, else error. Negative numbers represent system or network errors (such as unable to connect, no route to host, etc.). The string associated with the error will be returned in **ERRMSG** (or output to the specified file.) Positive numbers>100 are the actual codes returned by the SMTP server (with the remainder of the message returned in **ERRMSG**.) Positive numbers between 71 and 99 are application or misc errors. The more common **STATUS** values are defined in EMAILX.MAP (++include in your application) as shown below:

```

MAP1 EMS'STATUS                                ! STATUS definitions
MAP2 EMS'OK,B,1,0                               ! ok
MAP2 EMS'TOOFEW,B,1,71                          ! too few params
MAP2 EMS'TIMEOUT,B,1,72                         ! timed out
MAP2 EMS'NOTO,B,1,73                            ! no 'TO' address
MAP2 EMS'NOBODY,B,1,74                          ! no 'BODY'
MAP2 EMS'BADRESP,B,1,75                         ! Bad SMTP response
MAP2 EMS'NOSOCKET,B,1,76                       ! No socket open
MAP2 EMS'NOCFG,B,1,77                           ! Specified CFG file not present
MAP2 EMS'BADCFG,B,1,78                         ! Syntax error within CFGFILE
MAP2 EMS'BADSIG,B,1,79                         ! Bad signature file
MAP2 EMS'BADATT,B,1,80                         ! Bad attachment
MAP2 EMS'B64ERR,B,1,81                         ! Base64 encoding error

```

\* *Values 1-70 would indicate Basic error conditions. (Consult SYS:ERRMSG.USA or use DERR.SBR to get the text message associated with the error number.). To get the error message associated with a negative value of STATUS, use XCALL MIAMEG,86,ABS(STATUS),MESSAGE\$. Also see the ERRMSG parameter, which may contain a useful message, typically the last command sent to the server or last response from it.*

**FROM** (string, optional) specifies the email address of the sender. This is optional because it is likely to be the same for the entire application and thus can be specified in the configuration file (via the **RTNADDR** parameter), but if specified here, it will override the configuration file. Note that this is the address that will receive messages from the SMTP server itself about problems delivering the mail.

**ATTACHMENTS** is a formatted parameter which may specify any number of attachments:

```

MAP1 ATTACHMENTS
MAP2 ATT'COUNT,B,2                             ! # of attachments
MAP2 ATTX(n)                                    ! n >= value of ATT'COUNT
MAP3 ATT'FILE,S,128                             ! attachment filespec
MAP3 ATT'CONTENT'TYPE,S,127                    ! content'type string
MAP3 ATT'ENCODING,B,1                           ! See CTE'xx in EMAILX.MAP

```

\* *If you are not using attachments, you can specify the ATTACHMENTS argument as a null string (mapped or literal) for simplicity.*

The size of the ATTX() array (n as shown above) is up to you, but must be at least as large as ATT'COUNT, which in turn must equal the number of items in the array (i.e. attachments) which are to be used. The maximum is currently 10 (i.e. up to 10 attachments allowed.)

The ATT'FILE() items must contain the name of the file to be attached (in either AMOS or native format).

ATT'CONTENT'TYPE() is optional; if non-blank, it should be a legal MIME Content-Type header, for example:

```
Content-Type: application/octet-stream; name="xyz.doc"
```

- \* *You may omit the "Content-Type:" preface to the string, since if not present, it will automatically be added. You can also omit the name= field, in which case the name will be set to the actual name of the attachment (stripped of its directory.) If the entire string is blank, EMAILX will build one using "application/octet-stream" as the type.*

ATT'ENCODING() indicates the MIME Content-Transfer-Encoding to be used. The default (0) is base64, which should work for all attachments because it delivers an exact copy of the attached file. The only downside is that the encoding increases the size of the mail message by about 34%.

- \* *Note that EMAILX will automatically perform the Base64 encoding for you; you just supply the name of the file.*

The encoding types are all defined in EMAILX.MAP as follows:

```
MAP1 CTE'FLAGS                ! Content-Transfer-Encoding
MAP2 CTE'BASE64,B,1,0         ! Base64 (default)
MAP2 CTE'7BIT,B,1,1          ! 7 bit unencoded text
MAP2 CTE'8BIT,B,1,2          ! 8 bit unencoded text
MAP2 CTE'BINARY,B,1,3        ! binary unencoded data
MAP2 CTE'QUOTED,B,1,4        ! Quoted Printable encoding
```

CTE'7BIT, CTE'8BIT, and CTE'BINARY are all similar in that they indicate that no encoding has been done to the attachment. Of these, the first two should only be used with text consisting of reasonably short delimited lines. CTE'BINARY indicates that the data is truly raw, possibly not fitting into the line length requirements of SMTP. Both CTE'8BIT and CTE'BINARY require that all of the MTA's (Mail Transfer Agents) can handle 8 bit data, which is not necessarily guaranteed. (This is why CTE'BASE64 is preferred, since it encodes anything using a 7 bit character set.)

- \* *Also note that most mail readers will display 7bit text/plain attachments as if they were simply part of the message body rather than clearly separating them as attachments.*

CTE'QUOTED indicates that the attachment is already in "quoted printable" format. This is a format that consists mostly of plain text, with an escape mechanism use to encode special characters. The advantage of this over Base64 is primarily that it is still somewhat readable even if the mail reader doesn't understand the encoding. (Base64, on the other hand, is not human-readable at all.) However, by now, virtually all mail readers support Base 64, so we have not bothered to implement quoted printable encoding, other than to allow you to do your own encoding and then indicate that it has been done.

- \* *It is worth repeating that if you want to use quoted printable (CTE'QUOTED) encoding, you must do it yourself. EMAILX will only supply the proper header for you.*
- \* *Refer to RFC 1521 for more details on attachments and MIME headers.*

**SIGNATURE** (string, optional) filespec (AMOS or native) to be appended to bottom of message body. If blank, we use **SIGNATURE** from the configuration file instead. Like **BODY**, can also contain the actual text rather than a filename.

**ERRMSG** (string, optional) If present will receive a (possibly) useful text string if **STATUS** # 0 (elaborating on the error condition.).

**SOCKET** (numeric, optional) Ignored on input unless you use the **OPFLAGS** to avoid opening and closing the SMTP session for each message; in which case it receives the socket # of the session on the open call and must be passed to all the subsequent calls. (Note that you must manually close the socket if you use the EMF 'LEAVEOPEN option.)

**SERVER** (string, optional) Name or IP address of the SMTP server. (append :<port> if not using port 25); if blank, we use **SERVER** from the configuration file instead.

**HOST** (string, optional) Name to identify your domain to the SMTP server in the initial protocol exchange with the SMTP server. If blank, we use **HOST** from the configuration file. If blank there also, we use nothing, which may lead to your mail being rejected, or possibly sent with a warning that it comes from an unidentified source.

## Configuration File

The EMAILX configuration file is designed to contain those parameters which are most likely to be user-definable yet constant for a given site or application. The file is normally called EMAILX.CFG, although you specify the name (and location; there is no search path) yourself in the XCALL EMAILX statement. The file is a typical text configuration file consisting of lines of the form:

```
<parameter name> = <value> ; <optional comment>
```

Here is a sample configuration file with comments which should be sufficient to allow you to modify it to your needs:

```
;EMAILX.SBX configuration file
WAIT      = 10                ; max wait (secs) for SMTP response
SERVER    = localhost         ; SMTP server
HOST      = microsabio.com    ; HELO (sending domain)
DOMAIN    = microsabio.com    ; default domain to add to
                                     ; unqualified addr
RTNADDR   = ashell@microsabio.com ; return address (default
                                     ; FROM address)
REPLYTO   = replyto@microsabio.com ; default Reply-To address
```

```

SIGNATURE = EMAILX.SIG           ; signature file (text, amos or
                                   ; native spec)
LOGFILE   = EMAILX.LOG           ; log file name (amos or native)
LOGLVL    = 3                     ; 0=off, 1=errors only,
                                   ; 2=info summary (to,from,#bytes),
                                   ; 3=full msg (to,from,header,body),
                                   ; 4=partial debug (most cmds,responses
                                   ; 5=full debug (everything)

```

Each of these parameters is discussed in further detail below.

**WAIT** specifies the maximum number of seconds we will wait for a response from the SMTP server before aborting and returning `STATUS = EMS ' TIMEOUT`. If not specified, default is 30 seconds. Note that if you are using a dial-on-demand connection to your SMTP server, 30 seconds may not be long enough. On the other hand, if your SMTP server is on the local LAN, it is probably unnecessarily long (although the only harm there is that the user may get tired of waiting for the timeout, particularly during initial testing when the connection parameters may need tweaking.)

**SERVER** fulfills the same purpose here as the **SERVER** parameter on the XCALL line and is used only if the **SERVER** parameter in the XCALL is blank or not specified. Since a given installation will probably always use the same server, it may be much more convenient to put the information here in the configuration file than having to embed it in the application. Note that although SMTP servers almost always use port 25, if for some reason yours doesn't, you can append the port number to the host name or IP address with a colon separating them. For example:

```

SERVER = localhost:10025           ; local SMTP server on port 10025
SERVER = 1.2.3.4:9925             ; SMTP server at 1.2.3.4 port 9925
SERVER = mail.myisp.net           ; SMTP server at ISP

```

**HOST** fulfills the same purpose here as it does in the XCALL parameter list, and is used only if the **HOST** is blank or not specified in the XCALL. This is typically your domain name, and is used to identify your computer or domain to the SMTP server when making the initial connection. Theoretically, the SMTP server may decide to accept or reject you based on whether it recognizes or can resolve this domain name.

**DOMAIN** is a strictly optional parameter which specifies the domain to be added to any unqualified address, thus allowing you to abbreviate the addresses, say, for your local company by dropping the @ and everything after it. For example, if `DOMAIN=microsabio.com`, then if you specify the address "admin" in the **TO** or **FROM** fields, it will automatically be expanded to "admin@microsabio.com".

**RTNADDR** is another optional parameter which specifies the default **FROM** address. Most email clients have a similarly named field in their setup parameters which saves you from having to type the "From" address on every message. (In the case of EMAILX.SBR, you aren't likely to be "typing" any addresses, so the point may be moot.) Note that you can always supply this information directly in the message header (either in **HEADER** or **BODY**) by including a line consisting of "From:" followed by your email address. Note that you can use the extended addressing syntax here (optionally including a

text name followed by the real email address in angle brackets, e.g. "Administration <admin@microsabio.com>". You can of course skip this parameter and include it directly within the **HEADER** and/or **BODY** parameters.

**REPLYTO** is an important parameter often confused with the return address. This is where mail will be sent if the reader clicks the "**Reply**" button in their email client. It may or may not be the same as the address of the person (or cyber-entity) that actually sent the message. If specified, it inserts a line in the message header reading "Reply-To: " followed by the specified address.

**SIGNATURE** is equivalent to the XCALL parameter of the same name, and is used only if the XCALL parameter is blank or not present. The theory behind putting it here is that you may wish to include the same "signature" text on every message, yet every installation will surely have their own (typically listing important contact information for the company), so why bother having to reference it explicitly on every email. Note that there is nothing magical about the "signature" – it is just text (possibly "styled text") which could just as easily be included with the **BODY** parameter or not at all.

**LOGFILE** is an optional string specifying the name of a log file to which EMAILX will output (possibly) useful tracing and/or debugging information. The amount, or degree of detail, is determined by the next parameter (**LOGLVL**.) Beware that if you send a lot of mail, and especially if you opt for a high degree of detail, this file can rapidly grow quite large. The expectation is that installations might temporarily use it for debugging problems with the SMTP server (for which it is **extremely** useful) and otherwise turn the feature off or set it to errors only. As with all of the other parameters which specify filenames, you can use AMOS or native syntax here.

**LOGLVL** specifies the degree of detail to be written to the log file. It is ignored if **LOGFILE** is blank or not present, and it defaults to 1 (Errors only) if **LOGLVL** is not specified. The options are perhaps not completely self-explanatory, but the best way to understand them is just to experiment with a few sample messages and different settings. Typically you would just leave this set to 1 and check it periodically for error messages. Note that in the debug modes (4 and 5) the log file provides a rather graphic tutorial on how the SMTP protocol works, since it documents both the commands sent to the server and the responses received.

## Header Specifications

The specification for header fields is spelled out (in excruciating detail) in RFC 822, which any decent search engine can take you right to. However, if you are not interested in that level of detail, here are is a mercifully (but not very) brief overview.

The first thing to note is that from the standpoint of the SMTP protocol, headers are actually part of the body or "data" of the message, and thus is it quite possible that an SMTP server would accept virtually anything, including nothing, in the way of headers at the start of your message. However, that is not to say that SMTP servers do not typically parse and possibly interpret your headers and add their own, nor that email readers do not depend on certain minimum standards for message headers. Format wise, the main syntactic requirement is that the header section of the message consist of one or more lines with header keywords (e.g. **To**, **From**, **Date**, etc) followed by optional whitespace, followed by a colon,

followed by more optional whitespace, followed by some kind of text, which may have further syntactic requirements depending on the header keyword. For example, the **To:** field must contain validly formatted email addresses. If the data following the colon is too long to fit on one line, you may continue it by inserting a CRLF followed by **at least one whitespace character** followed by the continuation of your field. (The whitespace at the start of the continuation line is essential to mark it as a continuation of the prior header field and not a new header field.)

\* *Header keywords (e.g. **To**, **From**, **Subject**, etc.) are not case sensitive. However, in order to slightly reduce the confusion between similarly named header keywords (e.g. **To**, **From**) and EMAILX parameters (e.g. **TO**, **FROM**) we will from here on reference the former using upper and lower case, followed by a colon (e.g. **To:**) and the latter using upper case (e.g. **TO**). We highlight both in bold to make them stand out as keywords from the surrounding text.)*

The end of the headers and the start of the body of the message is marked by a blank line (two CRLF's in a row.) (This may explain the need for the EMF 'HDRCAT flag in **OPFLAGS**; it helps EMAILX decide if it should insert a blank line between your **HEADER** parameter and your **BODY** parameter.)

As a bare minimum, every message should have **Date:**, **To:**, and **From:** header fields. However, the SMTP server will automatically insert the **Date:** header field for you, so there is no need to manually include one.

The **To:** header field should not be confused with the **TO** parameter. The former is used mainly by the email reader application to format the message, whereas the latter is a critical parameter given directly to the SMTP server prior to the message. The SMTP server only wants an unadorned address for routing purposes, whereas the email reader may appreciate a fancier one, possibly including a text name in addition to the actual address. The actual **TO** parameter is typically not visible to the reader at all, whereas the **To:** header field is possibly the first thing he or she sees. An important ramification of this dichotomy between the **TO** parameter and the **To:** header field is that if you want to send your message to multiple people, you must specify them all in the **TO** parameter, but you do not have to (and may not want to) do so in the **To:** header field. (You might want to use the **cc:** or **bcc:** header fields instead, or not at all.). To see an example of this, just look at the headers generated by any any mass-mailed message (or junk mail message.) Typically, the actual **To:** header line will have nothing to do with you, and your email address will not appear anywhere in the header at all.

\* *Taking the example of traditional postal mail for comparison, the **TO** parameter is analogous to the address on the outside of the envelope, whereas the **To:** header field is analogous to the heading or salutation at the top of the letter inside the envelope. Similarly, if you want to copy your letter to multiple people, you have the option of including a **cc:** header in your letter, but the only way the message will actually be delivered multiple recipients is if you use multiple envelopes (i.e. multiple **TO** addresses.)*

The **From:** header field differs from the **FROM** parameter in nearly the same way as the **To:** header differs from the **TO** parameter. The **FROM** parameter is a separate command to the SMTP server, and is used to identify the sender to the SMTP server, not necessarily to the recipient of the message. Typically, however, they are the same, with the possible addition of an informal text version of the sender's name in the **From:** header (e.g. "Big Brother <bigbro@us.gov>" instead of just bigpro@us.gov"). SMTP servers will often insist that the **FROM** parameter specify a valid address

(since it needs this if it has to return the message because it was undeliverable.) They may or may not put restrictions on the **From:** header, but apparently this is not too common or we would not be deluged with junk-email with no usable **From:** address. Note that since your **From:** header is likely to be the same for all messages, you may want to put it in the configuration file and not bother with it in each message.

Other typical (and universally recognized) headers include:

**Subject :** (a free format line describing the message.) This is so common that most email readers have a space dedicated to it.

**cc :** (A list of names this message is being copied to.) To reiterate a critical point in the discussion above about the difference between the **To:** header and the **TO** parameter, the **cc:** header field is just for aesthetics. If you want to actually *send* the message to the people listed, you must include them in the **TO** parameter.

**Bcc :** (A list of people this message is being blindly copied to.) The comment above for the **cc:** field applies here as well, except that in practice, virtually all SMTP servers and/or email readers will interpret this field and hide the information from some or all of the recipients of the message.

**Reply-To :** (address to send replies to.) Virtually all email readers will interpret this field and use it if you select the “reply” option within the email program.

**Sender :** (address of the person sending the message, as distinct from the composer of the message.) For example, a secretary’s email address.

**Comment :** (a free-format comment which is not technically part of the message body.) For example, you might use this to include some explanatory remarks when a message is being forwarded, or when the message is being sent by someone other than the actual originator of the message. It might be considered analogous to a post-it note stuck to a typewritten letter.

**In-Reply-To:** (an indication of what (and/or who) this message is in reply to.)

## User Defined Header Fields

If you want to define your own header fields, the specification requires that you start them with “X-“ to avoid any possible confusion with previously defined or other header fields that may be recognized and interpreted by the SMTP server or mail reader. The most common example of this is the field **X-Mailer:** which most email clients now add to identify the name of the software or server used to generate the message. For example, here are two commonly seen **X-Mailer:** headers:

```
X-Mailer: Microsoft Outlook Express 5.00.2314.1300
X-Mailer: QUALCOMM Windows Eudora Version 5.0
```

EMAILX adds its own X-Mailer header indicating its version number.

## EMAILX Scenarios

Even though EMAILX.SBR goes a long way towards simplifying the process of sending email from an application, the need for flexibility combined with the underlying complexity of the email format rules make for a lot of confusion. To help clear away some of that confusion, we discuss a few “scenarios” here in more detail.

### Scenario 1: Simple Plain Text Messages

This case does not require much elaboration. Just put your message either entirely in the **BODY** parameter (either as a text string with CRLF between each line, or as a file) or put the header items in the **HEADER** parameter and the rest of the message in the **BODY** parameter. The **SIGNATURE** is optional, and if present is treated logically just like a continuation of the message body. No special flags are needed in **OPFLAGS**. Here is an extremely simple example which does not even use on a configuration file:

```
MAP1 EMAIL'PARAMS
  MAP2 EM'TO,S,200
  MAP2 EM'FROM,S,50
  MAP2 EM'HEADER,S,1000
  MAP2 EM'BODY,S,2000
  MAP2 EM'SERVER,S,20,"mail.king.net"
  MAP2 EM'HOST,S,20,"looseheads.net"
  MAP2 EM'ERRMSG,S,100
  MAP2 CRLF$,S,2
  MAP2 EM'STATUS,F

EM'TO = "louisxvi@versailles.gov" + chr(13) &
      + "robespierre@reignofterror.org"
EM'FROM = "marie@foodnet.org"
CRLF$ = chr(13) + chr(10)
EM'HEADER = "To: Louis" + CRLF$ + "From: Marie" + CRLF$ &
          + "Subject: cake recipes"
EM'BODY = "Dear Louis," + CRLF$ &
          + "What is this 'cake' I keep hearing so much about?" &
          + CRLF$ + "Je t'aime, Marie"

XCALL EMAILX,0,"",EM'TO,EM'HEADER,EM'BODY,EM'STATUS,EM'FROM, &
      "", "",EM'ERRMSG,SOCKET,SERVER,HOST
IF EM'STATUS # 0 then print "Error #";EM'STATUS;" - ";EM'ERRMSG
```

## Scenario 2: Simple HTML Text Messages

This case is identical to the plain text case above, except that you can include HTML markup commands in your text to give it “style”. Most modern email readers understand HTML commands, even if you don’t bother to indicate in the header that you are using them. (But to be correct, you should set the EMF’HTMLBODY flag in **OPFLAGS**.) If the email reader does not understand HTML, then the reader will see your markup commands as if they were part of the text. But fortunately due to the nature of HTML commands, the text is still likely to be “readable”, although the extent may depend on how complicated you make your HTML.

Here is a slightly more complicated example using a configuration file and a file for the body of the message:

EMAILX.CFG -----

```
;EMAILX.SBX configuration file
WAIT      = 10                ; 10 sec wait for server
SERVER    = localhost         ; SMTP server on local host
HOST      = sabionet.com      ; our domain id
DOMAIN    = microsabio.com    ; default domain for addr's
RTNADDR   = email@microsbio.com ; default 'FROM' addr
REPLYTO   = replyto@microsbio.com ; reply-to addr
LOGFILE   = EMAILX.LOG        ; name of log file
LOGLVL    = 2                 ; info summary
```

BODY1.TXT -----

```
<HTML>
<HEAD><TITLE>This is the title</TITLE></HEAD>
<BODY>
<H1>This is Header #1</H1>
<H2>This is Header #2</H2>
<P>This is paragraph #1.
As you can see the formatting is based
on
the
P and /P notations and not on the
actual line breaks</P>
<P>This is paragraph #2</P>
</BODY></HTML>
```

```

EMHTML.BAS -----
MAP1 EMAIL'PARAMS
MAP2 EM'OPFLAGS,B,2
MAP2 EM'TO,S,200
MAP2 EM'CFGFILE,S,20,"EMAILX.CFG"
MAP2 EM'FROM,S,50
MAP2 EM'HEADER,S,1000
MAP2 EM'BODY,S,30
MAP2 EM'ERRMSG,S,100
MAP2 CRLF$,S,2
MAP2 EM'STATUS,F

++INCLUDE EMAILX.MAP          ! contains needed symbols

EM'OPFLAGS = EMF'BODYHTML
EM'TO = "test@microsabio.com"
CRLF$ = chr(13) + chr(10)
EM'HEADER = "To: HTML Test Department" + CRLF$ &
            + "From: Field Research" + CRLF$ &
            + "Subject: Testing HTML formatted message"
EM'BODY = "BODY1.TXT"

XCALL EMAILX,EM'OPFLAGS,EM'CFGFILE,EM'TO,EM'HEADER,EM'BODY, &
        EM'STATUS,EM'FROM,"","",EM'ERRMSG
IF EM'STATUS # 0 then print "Error #";EM'STATUS;" - ";EM'ERRMSG

```

### Scenario 3: Messages with Attachments

Attachments require the construction of MIME headers and usually the encoding of the attachment using “base64”. Most people do not want to know anything about how that is accomplished, and would prefer to imagine that it is as simple as just listing the names of the attached files in the header of the message (an illusion happily fostered by most email client applications.) Fortunately, EMAILX.SBR is willing to go along with this charade, allowing you to specify nothing other than the names of the files you want to attach.

Most email clients make assumptions about the types of file attachments based either on the extension of the filename, or possibly by scanning the file to see if it consists of anything other than text. Rather than get mired in such hocus-pocus, EMAILX adopts a simpler approach. If you don’t tell it otherwise, it automatically treats each attachment as being of type “application/octet-stream” which is the most general type. If you want to be more specific, you can specify one of the content types defined in the email standard (see RFC 1521 and 1522), such as:

```

application/postscript
image/jpeg
image/gif
audio/basic
video/mpeg

```

The following example uses the same configuration file (EMAILX.CFG) as in the previous example, and is otherwise very similar except for the inclusion of two attachments and the addition of a signature block:

```
EMSIG.TXT -----
```

```

*****
***** EMAIL Field Research and Testing *****
*****

```

```
EMATT.BAS -----
```

```

MAP1 EMAIL'PARAMS
MAP2 EM'OPFLAGS,B,2
MAP2 EM'TO,S,200
MAP2 EM'CFGFILE,S,20,"EMAILX.CFG"
MAP2 EM'FROM,S,50
MAP2 EM'HEADER,S,1000
MAP2 EM'BODY,S,1000
MAP2 EM'ERRMSG,S,100
MAP2 CRLF$,S,2
MAP2 EM'STATUS,F

MAP1 EM'ATTACHMENTS
MAP2 ATT'COUNT,B,2
MAP2 ATTX(10)
MAP3 ATT'FILE,S,128
MAP3 ATT'CONTENT'TYPE,S,127
MAP3 ATT'ENCODING,B,1

++INCLUDE EMAILX.MAP

EM'OPFLAGS = EMF'NORMAL
EM'TO = "test@microsabio.com"
CRLF$ = chr(13) + chr(10)
EM'HEADER = "To: Attachment Test Department" + CRLF$ &
  + "From: Field Research" + CRLF$ &
  + "Subject: Testing Attachments"
EM'BODY = "There are two attachments to this message:" &
  + CRLF$ + " EMAILX.SBX (binary) and miame.ini (text)" &
  + CRLF$

```

```
ATT'COUNT = 2
ATT'FILE(1) = "BAS:EMAILX.SBX"
ATT'CONTENT'TYPE(1) = ""           ! (let it default)
ATT'ENCODING(1) = CTE'BASE64       ! (base64 encoding)
ATT'FILE(2) = "/vm/miame/miame.ini"
ATT'CONTENT'TYPE(2) = "text/plain"
ATT'ENCODING(2) = CTE'7BIT         ! 7 bit text, no encoding

XCALL EMAILX,EM'OPFLAGS,EM'CFGFILE,EM'TO,EM'HEADER,EM'BODY, &
      EM'STATUS,EM'FROM,EM'ATTACHMENTS,"EMSIG.TXT",EM'ERRMSG
IF EM'STATUS # 0 then print "Error #";EM'STATUS;" - ";EM'ERRMSG
```

- \* *Note that there was no particularly good reason to not use base64 encoding (CTE'BASE64) for the second attachment as well, other than to save a few bytes. In fact, one possible problem with text/plain encoding is that the line terminators will be converted to CRLF, even if they were originally just LF. Base64 encoding eliminates this possibility.*