

CROSS Mail Suite Documentation

The C.R.O.S.S. Mail Generation Suite is a set of programs and subroutines written in BASIC for U2. They support the following functions:

- Email can be sent using Simple Mail Transport Protocol (SMTP)
- Email can be retrieved using Post Office Protocol version 3 (POP3)
- Multipurpose Internet Mail Extensions (MIME) compliant email can be generated and decoded. This allows multiple attachments to be sent/received with each message.
- Email addresses can be validated and the host names of addresses verified using Domain Name Service (DNS).
- DNS queries may be sent directly from U2
- A user interactive program is supplied which allows a user to send email with files attached from a file, a HOLD file or the UniVerse spooler. These files may optionally be converted to PDF before sending.

Copyright (c) 2006 C.R.O.S.S. Pty Limited. All rights reserved. Unauthorised reproduction or publication in any form is prohibited. Property of C.R.O.S.S. Pty Limited.

All trademarks are the property of their respective owners.

This publication and the information herein are furnished AS IS, are subject to change without notice, and should not be construed as a commitment by C.R.O.S.S. Pty Ltd. C.R.O.S.S. Pty Ltd assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringement of third-party rights.

Contents

| | |
|--|----|
| Document Conventions..... | 3 |
| Using the Mail Suite | 4 |
| Sending Mail..... | 5 |
| Generating a MIME message..... | 8 |
| Sending a text message with no attachments..... | 9 |
| Sending a HTML message with attachments..... | 10 |
| Retrieving Mail | 11 |
| The POP3 Protocol | 11 |
| Using POP3..... | 11 |
| Connecting/Disconnecting from a POP3 Server..... | 12 |
| Authenticating with using USER Authentication..... | 14 |
| Authenticating with using APOP Authentication..... | 15 |
| Authenticating with using NONE Authentication..... | 16 |
| QUIT from a session..... | 17 |
| Retrieving Mailbox status (STAT) | 17 |
| Retrieving a Mailbox Listing (LIST)..... | 18 |
| Retrieving a Message (RETR)..... | 18 |
| Marking a message for deletion (DELE)..... | 19 |
| The NOOP command..... | 19 |
| Unmarking messages for deletion (RSET) | 20 |
| Retrieving part of a message (TOP)..... | 20 |
| Message Unique ID Listing (UIDL)..... | 21 |
| Decoding a MIME Message | 22 |
| Validating an Email Address | 26 |
| Performing a DNS query | 27 |
| Performing a DNS query from TCL..... | 27 |

Document Conventions

When reading this documentation, the following conventions are used to emphasise meaning. Program code and other literals or values expected to be used directly in BASIC are formatted in fixed pitch font, for example:

```
CROSS.POP3.CONNECT(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, SERVER, PORT, TLS, TIMEOUT, SECURITYCONTEXT)
```

In BASIC code examples equated values always use the underscore character in their names while variables or subroutine names use periods. For example

```
POP3_DEFAULT_PORT
```

is an equated value while

```
POP3.SESSION
```

is a variable. When equated values are presented for the first time, they will usually show the value they are equated to in brackets after the equate name, so

```
POP3_DEFAULT_PORT (110)
```

shows that `POP3_DEFAULT_PORT` is equated to 110.

Using the Mail Suite

The Mail suite is supplied with a set of BASIC equates to simplify coding with the Mail routines. The equates are specific to different parts of the suite and are held in

| | |
|-------------------------------------|---------------------------------------|
| <code>CROSS.MAIL.INC</code> | Common includes for all programs |
| <code>CROSS.POP3.INC</code> | Post Office Protocol version 3 |
| <code>CROSS.SMTP.INC</code> | Simple Mail Transfer Protocol |
| <code>CROSS.MIME.INC</code> | Multipurpose Internet Mail Extensions |
| <code>CROSS.RFC822.INC</code> | Internet Message Format |
| <code>CROSS.EMAILADDRESS.INC</code> | Address Validation |

The first four parameters of all the subroutines in the suite are the same. The first parameter indicates the success or failure of the subroutine call. This parameter is designed to allow the routines to be used with functional definitions to allow if/then/else branching based on the result of the subroutine call.

Two equated values are provided to test the success or failure of the calls. These are `MAIL_RETURN_SUCCESS (0)` and `MAIL_RETURN_FAIL (1)`.

So for example `CROSS.DNS.QUERY.SUB` can be used either as a subroutine and the first parameter tested:

```
CALL CROSS.DNS.QUERY.SUB(ERR, SLRESP, ALRESP, RESPDATA, MAT DNS.QUERY, MAT
DNS.RESPONSE)

IF ERR NE MAIL_RETURN_SUCCESS THEN
    * Error/Failure handling routine
END
```

Or the routine can be defined as a function and used in line for example:

```
DEFFUN CROSS.DNS.QUERY.SUB(ERR, SLRESP, ALRESP, RESPDATA, MAT DNS.QUERY, MAT
DNS.RESPONSE)

IF CROSS.CROSS.DNS.QUERY.SUB(SLRESP, ALRESP, RESPDATA, MAT DNS.QUERY, MAT
DNS.RESPONSE) THEN
    * Error/Failure handling routine
END
```

If a subroutine call indicates failure then `SLRESP`, `ALRESP` and `RESPDATA` provide more detailed information. If set, `SLRESP` contains a system level error code. This will typically be an error code returned by UniVerse after a socket call or file operation or an error code returned from the a mail or DNS server. If set `ALRESP` contains an application level error code. This will indicate an errors from the mail suite such as incorrect input parameters or a contextual error code to supplement `SLRESP`. In either case `RESPDATA` will contain a readable description of the error.

Sending Mail

To send email using the suit, use the `CROSS.SENDEMAIL.SUB` routine. This routine allows an email message to be sent with multiple attachments.

The call signature for this routine is:

```
CROSS.SENDEMAIL.SUB(ERR, SLRESP, ALRESP, RESPDATA, MESSAGE.FROM,  
MESSAGE.TO, SUBJECT, BODYPARAMS, BODY, ATTACHPARAMS, MAT ATTACHMENTS,  
MAILSERVER, OPTIONS, MAT RECIPEERS)
```

In addition to the standard four described above, the following parameters may be passed into the subroutine:

| | |
|---------------------------|---|
| <code>MESSAGE.FROM</code> | <p>The first attribute of this parameter holds the sender email address. The second attribute may optionally contain a display name to associate with the address. For example:</p> <pre>MESSAGE.FROM<1> = "craig@cross.net.au" MESSAGE.FROM<2> = "Mr Craig Bennett"</pre> |
| <code>MESSAGE.TO</code> | <p>Contains multiple recipient email addresses one address per attribute. The second value in each attribute determines how the recipient will be added to the email:</p> <ul style="list-style-type: none">• If it contains <code>T</code> or is empty, the recipient will appear in the <code>TO</code> header.• If it contains <code>C</code> the recipient will appear in the <code>CC</code> header.• If it contains any other value, the recipient will receive the email, but will not appear in any headers. <p>For example:</p> <pre>* TO recipient MESSAGE.TO = "craig@cross.net.au" * CC recipient MESSAGE.TO<2> = "craig@cross.net.au":@VM:"C" * BCC recipient MESSAGE.TO<3> = "craig@cross.net.au":@VM:"B"</pre> |
| <code>SUBJECT</code> | <p>The subject header to be sent with the email. For example:</p> <pre>SUBJECT = "Test Email"</pre> |

| | |
|---------------------|---|
| BODYPARAMS | <p>This determines how the body is attached to the email. If a plain text body is supplied this can be left empty.</p> <p>Attribute 1 holds the content type (for a list of known types see http://www.iana.org/assignments/media-types/). The default context type is <code>text/plain</code>.</p> <p>Attribute 2 holds the encoding for the body. The default encoding is 7bit ASCII (characters 0 to 127). Valid values for encoding are:</p> <pre> MIME_ENCODE_BASE64 (1) MIME_ENCODE_BASE64_ALREADY (2) MIME_ENCODE_QUOTED_PRINTABLE (3) MIME_ENCODE_QUOTED_PRINTABLE_ALREADY (4) MIME_ENCODE_7BIT (5) MIME_ENCODE_7BIT_ALREADY (6) </pre> <p>If one of the <code>_ALREADY</code> encoding values is chosen it is the responsibility of the caller to ensure that the attachment is correctly encoded.</p> |
| BODY | <p>This is the text body of the email. If this is a plain text body then lines should be terminated by a <code>CRLF</code> sequence (Characters 13 and 10). For example:</p> <pre> BODY = "Email Body Line 1":CHAR(13):CHAR(10) BODY = "Email Body Line 2":CHAR(13):CHAR(10) </pre> |
| ATTACHPARAMS | <p>If attachments are to be sent with the email, this parameter determines how each element of the <code>ATTACHMENTS</code> array is attached to the email. All attributes are multi-valued.</p> <p>Attribute 1 holds the content types. The default content type is <code>application/octet-stream</code>.</p> <p>Attribute 2 holds the encodings. The default encoding is <code>MIME_ENCODE_BASE64</code>. Valid values for encoding are as for <code>BODYPARAMS</code>.</p> <p>Attribute 3 holds file names for the attached files for example: <code>Sales.pdf</code>.</p> <p>Attribute 4 holds an optional unique ID for the attachment.</p> <p>Attribute 5 holds an optional short description for example: <code>June Sales Results</code>.</p> |

| | |
|------------------------|---|
| | <p>Attribute 6 determines whether the attachment should be displayed inline or as an attached file. Possible values are:</p> <p>MIME_DISPOSITION_INLINE (1) The mail application should show the attachment with the message body if possible.</p> <p>MIME_DISPOSITION_ATTACHMENT (2) The mail application should show the attachment as a separate file.</p> |
| MAT ATTACHMENTS | <p>This is a dimensioned array with one element for each attachment to be sent with the email. If this is dimension one and empty, no attachments will be sent. For example to send six attachments:</p> <pre> DIM ATTACHMENTS (6) MAT ATTACHMENTS = "" ATTACHMENTS (1) = MYFIRSTATTACHMENT ATTACHMENTS (2) = MYFIRSTATTACHMENT ATTACHMENTS (3) = MYFIRSTATTACHMENT ATTACHMENTS (4) = MYFIRSTATTACHMENT ATTACHMENTS (5) = MYFIRSTATTACHMENT ATTACHMENTS (6) = MYFIRSTATTACHMENT </pre> |
| MAILSERVER | <p>The IP address or hostname of the Mail service you wish to connect to. If this is not specified it defaults to 127.0.0.1 which is the loop back address of the local machine (localhost). This is a useful default on most Unix/Linux machines but windows servers do not usually run an SMTP service by default.</p> |
| OPTIONS | <p>This variable sets other options for the message.</p> <p>If attribute 1 evaluates to true, a MIME email will be built and returned in BODY, but no message will be sent. This is a useful option for debugging.</p> <p>If attribute 2 evaluates to true, email address syntax will not be checked and the program will allow invalid sender and recipient addresses.</p> <p>If attribute 3 evaluates to true, the domain portion of email addresses will be validated using DNS before sending (this will be validated by testing for the existence of a Mail Exchanger (MX) record for the domain).</p> <p>Attribute 4 holds the GMT offset for local system time. This should be formatted as +NNNN or -NNNN. This is an important parameter to supply if you want email send times to appear correctly. By default local system time is assumed to be set to Universal Time (+0000). For example for Eastern Australian Winter Time use +1000.</p> |

| | | | | | | | | | |
|-------------------------------|---|--------------------------|------------|-----------------------|-------------|------------------------|------------|-------------------------------|------------|
| | <p>Attribute 5 holds an optional timeout in milliseconds for socket operations. The default value is 10 seconds (10000).</p> <p>Attribute 6 allows the socket connection mode to be set. Possible values are:</p> <p>0 – Use non-blocking mode sockets</p> <p>1 – Use blocking mode sockets</p> <p>The default value is 0.</p> | | | | | | | | |
| MAT RECIPEERRS | <p>This holds extended error detail for email recipients. This array should be dimension 4.</p> <p>Element 1 holds a count of the number of recipients.</p> <p>Element 2 holds a multi-valued list of recipient SMTP status codes.</p> <p>Element 3 holds a multi-valued list of recipient error codes.</p> <table data-bbox="544 936 1284 1055"> <tr> <td>SMTP_MAIL_SUCCESS</td> <td>(0)</td> </tr> <tr> <td>SMTP_MAIL_FAIL</td> <td>(-1)</td> </tr> <tr> <td>SMTP_MAIL_DEFER</td> <td>(1)</td> </tr> <tr> <td>SMTP_MAIL_NOT_YET_SENT</td> <td>(2)</td> </tr> </table> <p>Element 4 holds the corresponding recipient email addressed.</p> | SMTP_MAIL_SUCCESS | (0) | SMTP_MAIL_FAIL | (-1) | SMTP_MAIL_DEFER | (1) | SMTP_MAIL_NOT_YET_SENT | (2) |
| SMTP_MAIL_SUCCESS | (0) | | | | | | | | |
| SMTP_MAIL_FAIL | (-1) | | | | | | | | |
| SMTP_MAIL_DEFER | (1) | | | | | | | | |
| SMTP_MAIL_NOT_YET_SENT | (2) | | | | | | | | |

Generating a MIME message

As documented above, **CROSS.SENDEMAIL.SUB** can also be used to generate MIME message without sending by setting attribute 1 of the **OPTIONS** parameter to true.

Sending a text message with no attachments

The following is example code to send a plain text email with no attachments:

```
* RECIPEERS is used to pass back recipient error detail if any
DIM RECIPEERS(4)
MAT RECIPEERS = ""

* Set up mail parameters
MESSAGE.FROM = "craig@cross.net.au"

* MESSAGE.TO
MESSAGE.TO = "craig@cross.net.au"

SUBJECT = "Test email"

BODYPARAMS = ""

BODY = ""
BODY<1> = "A test email suitable for all to read and enjoy."
BODY<3> = "regards,"
BODY<4> "Craig"
* Convert attribute marks to CRLF
BODY = EREPLACE(BODY, @AM, CHAR(13):CHAR(10))

ATTACHPARAMS = ""

DIM ATTACHMENTS(1)
MAT ATTACHMENTS = ""

MAILSERVER = "smtp.mynetwork.com"

OPTIONS = ""
* Eastern Australia Winter Time is GMT +10
OPTIONS<4> = "+1000"
* 60 second timeout - slow connection to mail server
OPTIONS<5> = "60000"

CALL CROSS.SENDEMAIL.SUB(ERR, SLRESP, ALRESP, RESPDATA, MESSAGE.FROM,
MESSAGE.TO, SUBJECT, BODYPARAMS, BODY, ATTACHPARAMS, MAT ATTACHMENTS,
MAILSERVER, OPTIONS, MAT RECIPEERS)
IF ERR THEN
  PRINT "ERROR : ":SLRESP:" ":ALRESP:" ":RESPDATA
  FOR I = 1 TO RECIPEERS(1)
    IF RECIPEERS(3) EQ SMTP_MAIL_SUCCESS ELSE
      PRINT RECIPEERS(4)<I>:" ":
      PRINT RECIPEERS(3)<I>:" ":
      PRINT RECIPEERS(2)<I>
    END
  NEXT I
END ELSE
  PRINT "MAIL SENT"
END
```

Sending a HTML message with attachments

The following is example code to send an email with a HTML body and several attachments:

```
DIM RECIPEERS(4); MAT RECIPEERS = ""

* To address has an optional display name
MESSAGE.FROM = ""
MESSAGE.FROM<1> = "craig@cross.net.au"
MESSAGE.FROM<2> = "Mr Craig Bennett"

* TO, CC and BCC recipients
MESSAGE.TO<1> = "craig@cross.net.au"
MESSAGE.TO<2> = "craig@cross.net.au":@VM:"C"
MESSAGE.TO<3> = "craig@cross.net.au":@VM:"B"

SUBJECT = "Extended Test email"

* Specify that the body is HTML and request quoted printable encoding
BODYPARAMS = ""
BODYPARAMS<1> = "text/html"
BODYPARAMS<2> = MIME_ENCODE_QUOTED_PRINTABLE

BODY = "<html><head></head><body style='background-color: silver;'><table
align='center' width='100%' style='background-color: white;'><tr><td>TABLE
LINE 1</td></tr><tr><td align='right'>TABLE LINE
2</td></tr></table></body></html>"

MAILSERVER = "smtp.mynetwork.com"

* Send two attachments. A CSV file which will automatically open in excel
* and an RTF file.

DIM ATTACHMENTS(2)
MAT ATTACHMENTS = ""

ATTACHPARAMS<1, 1> = "application/vnd-msexcel"
ATTACHPARAMS<2, 1> = MIME_ENCODE_QUOTED_PRINTABLE
ATTACHPARAMS<3, 1> = "example.csv"
ATTACHPARAMS<6, 1> = MIME_DISPOSITION_ATTACHMENT

ATTACHPARAMS<1, 2> = "application/rtf"
ATTACHPARAMS<2, 2> = MIME_ENCODE_BASE64
ATTACHPARAMS<3, 2> = "example.rtf"
ATTACHPARAMS<6, 2> = MIME_DISPOSITION_ATTACHMENT

ATTACHMENTS(1) = '"EXAMPLE, CSV FILE", "ONE LINE ONLY":CRLF
ATTACHMENTS(2) = '{\rtf1\ansi\deff0 {\fonttbl {\f0 Times New
Roman;}}\f0\fs60 Example RTF }'

OPTIONS = ""
* Western European Summer Time is GMT +1
OPTIONS<4> = "+0100"

CALL CROSS.SENDEMAIL.SUB(ERR, SLRESP, ALRESP, RESPDATA, MESSAGE.FROM,
MESSAGE.TO, SUBJECT, BODYPARAMS, BODY, ATTACHPARAMS, MAT ATTACHMENTS,
MAILSERVER, OPTIONS, MAT RECIPEERS)
IF ERR THEN
    * Error Handling Code
END
```

Retrieving Mail

The POP3 Protocol

The CROSS Mail Suite uses the POP3 protocol to retrieve email. The POP3 protocol works around three states for a connection. After initial connection to the service, the session is in the Authentication state. In this state the client can authenticate to the server (using the USER, APOP or NONE commands) or QUIT the connection and disconnect.

Once a client has successfully authenticated with the server, the session enters the transaction state. In this state the client can perform mailbox operations (list, retrieve, delete etc).

Once a client in the transaction state issues the QUIT command, the server enters the update state and session operations are actually performed. If the client disconnects before issuing the QUIT command no preceding operations from the transaction state are committed and the users mailbox remains as it was prior to the client connection.

Using POP3

The dimensioned array `POP3.SESSION` holds information about the current POP3 connection including the socket handle for the connection and must be passed into all calls to the POP3 routines. To prepare `POP3.SESSION` before calling the suite routines use code similar to the following:

```
DIM POP3.SESSION(POP3_SESSION_DIMLEN)
MATPARSE POP3.SESSION FROM POP3_SESSION_DEFAULT
```

This will create the `POP3.SESSION` array with the correct dimension and initialise it with default parameters. If your flavour of U2 does not support runtime dimensioned arrays use the appropriate compiler switch (eg: `$OPTIONS -STATIC.DIM`) or copy the array dimension from the equates file:

```
DIM POP3.SESSION(25)
```

In order to use POP3 service you must first make a TCP socket connection to the service. Once this is done, your session is implicitly in the Authorisation state and you must authenticate with the service.

Connecting/Disconnecting from a POP3 Server

The Mail suite allows both unencrypted and encrypted connections to a POP3 server. Encrypted POP3 services are sometimes referred to as POP3S (POP3 Secure) connections.

The call signature for the connection subroutine, `CROSS.POP3.CONNECT`, is:

```
SUBROUTINE CROSS.POP3.CONNECT(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, SERVER, PORT, TLS, TIMEOUT, SECURITYCONTEXT)
```

In addition to the standard four described above, the following parameters may be passed into the subroutine:

| | |
|------------------------|---|
| SERVER | The IP address or hostname of the POP3 service you wish to connect to. If this is not specified it defaults to <code>127.0.0.1</code> which is the loop back address of the local machine (<code>localhost</code>). |
| PORT | The port number you wish to connect to. For unencrypted connections this defaults to <code>110</code> , for encrypted connections this defaults to <code>995</code> . |
| TLS | If this variable evaluates to true the suite will attempt to make an encrypted connection. |
| TIMEOUT | This is a timeout value for socket operations in milliseconds. The default value is <code>2000</code> . |
| SECURITYCONTEXT | In order to make an encrypted connection you must pass in a security context previously created with the U2 security routines. |

After a successful call to this routine, `POP3.SESSION` will hold a socket handle to the server in the `POP3_SESSION_SKT` element.

The call signature for the disconnection subroutine, `CROSS.POP3.DISCONNECT`, is:

```
CROSS.POP3.DISCONNECT(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION)
```

The following is example code to make an encrypted connection to a server and then disconnect.

```
* These are the U2 routines for creating a security context
IF CREATESECURITYCONTEXT(CONT, "TLSv1") THEN
PRINT "createSecurityContext Failed"
STOP
END

* In production you may wish to use more restrictive authentication
* rules however since you are probably using TLS to prevent password
* snooping rather than to verify the identity of your LDAP server
* this may not be necessary.
IF ADDAUTHENTICATIONRULE(CONT, 2, "VerificationStrength", "generous") THEN
PRINT "addAuthenticationRule Failed"
STOP
END

SLRESP = 0
ALRESP = 0
RESPDATA = ""

SERVER = "mypop3server"
TLS=@TRUE

CALL CROSS.POP3.CONNECT(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION,
SERVER, PORT, TLS, TIMEOUT, CONT)
IF ERR THEN
PRINT "CONNECT FAILED"
PRINT "System Error      ":SLRESP
PRINT "Application Error ":ALRESP
PRINT "Description       ":RESPDATA
STOP
END

* At this point we are connected to the POP3 server

SLRESP = 0
ALRESP = 0
RESPDATA = ""

CALL CROSS.POP3.DISCONNECT(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION)
IF ERR THEN
PRINT "DISCONNECT FAILED"
PRINT "System Error      ":SLRESP
PRINT "Application Error ":ALRESP
PRINT "Description       ":RESPDATA
STOP
END

* At this point we are disconnected from the POP3 server
* and the socket handle in POP3.SESSION is closed
```

Authenticating with using USER Authentication

The commonest form of authenticating with a POP3 service requires a user name and password. Although this is sufficient to authenticate a user it should be noted that unless an encrypted connection to the server is used, the username and password will be passed in plain text across the network which may leave this information vulnerable to network sniffing. Nevertheless, this is no more insecure than logging into U2 using telnet. The call signature for `CROSS.POP3.AUTH.USER` is:

```
SUBROUTINE CROSS.POP3.AUTH.USER(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, USER, PASS)
```

The parameters unique to this subroutine are:

| | |
|-----------------|---|
| USER | This is the user name to use for authenticating <code>jsmith</code> or <code>jsmith@somewhere.com</code> |
| PASSWORD | This is the password to authenticate the user name |

If **ERR** evaluates to true then the user is not authenticated, otherwise the user is authenticated and we are implicitly in the transaction state. Sample code for user authentication is shown below:

```
USER="username"
PASS="password"
CALL CROSS.POP3.AUTH.USER(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION,
USER, PASS)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
  IF ALRESP EQ POP3_ERROR_AUTHENTICATION_FAILED THEN
    * Perhaps the password was wrong and you should try again?
  END ELSE
    * Other error handling code
  END
END
```

Authenticating with using APOP Authentication

APOP Authentication method does not pass the user password across the network in plain text, instead it passes a digest based on the password and a challenge phrase from the server. It is not supported by all POP3 services and is not as strong as an appropriate SSL connection it is however superior to plaintext user authentication. If POP3.SESSION element POP3_SESSION_APOP is true then the server claims to support APOP authentication.

The call signature for `CROSS.POP3.AUTH.APOP` is:

```
SUBROUTINE CROSS.POP3.AUTH.APOP(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, USER, SECRET)
```

The parameters unique to this subroutine are:

| | |
|---------------|---|
| USER | This is the user name to use for authenticating <code>jsmith</code> or <code>jsmith@somewhere.com</code> |
| SECRET | This is the password to authenticate the user name |

If **ERR** evaluates to true then the user is not authenticated, otherwise the user is authenticated and we are implicitly in the transaction state. Sample code for user authentication is shown below:

```
USER="username"
SECRET="password"
CALL CROSS.POP3.AUTH.APOP(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION,
USER, SECRET)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
  BEGIN CASE
    CASE ALRESP EQ POP3_ERROR_AUTHENTICATION_FAILED
      * Perhaps the password was wrong and you should try again?

    CASE ALRESP EQ POP3_ERROR_SERVER_DOES_NOT_SUPPORT_APOP
      * The server does not support APOP

    CASE @TRUE
      * Other error handling code

  END CASE
END
```

Authenticating with using NONE Authentication

The POP3 protocol also supports authentication using the NONE method. This implies that the session has been authenticated at some other layer (perhaps during SSL negotiation).

The call signature for `CROSS.POP3.AUTH.NONE` is:

```
SUBROUTINE CROSS.POP3.AUTH.NONE(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION)
```

If `ERR` evaluates to true then the user is not authenticated (or the server will not accept the NONE operation), otherwise the user is authenticated and we are implicitly in the transaction state. Sample code for user authentication is shown below:

```
CALL CROSS.POP3.AUTH.NONE(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
  IF ALRESP EQ POP3_ERROR_AUTHNONE_DID_NOT_AUTHENTICATE THEN
    * Operation failed
  END ELSE
    * Other error handling code
  END
END
```

QUIT from a session

The QUIT command can be issued at any time to end a POP3 session. When issued in the transaction state it has special meaning as it indicates that the server should update the users mailbox with all the commands issued during the session. If a session wishes to abandon all the operations thus far it should either use the RSET command (which will unmark any messages marked for deletion) or drop the connection.

The call signature for `CROSS.POP3.QUIT` is:

```
SUBROUTINE CROSS.POP3.QUIT(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION)
```

If `ERR` evaluates to true then the operation failed. The client should drop the connection but the mailbox will be left in an unknown state.

```
CALL CROSS.POP3.QUIT(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END
```

Retrieving Mailbox status (STAT)

The STAT command can only be issued in the transaction state. It returns the number of messages in a mailbox and the overall mailbox size in bytes.

The call signature for `CROSS.POP3.STAT` is:

```
SUBROUTINE CROSS.POP3.STAT(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, MCNT, MBXSIZE)
```

The parameters unique to this subroutine are:

| | |
|----------------|--|
| MCNT | Returns the number of messages in the mailbox. |
| MBXSIZE | Returns the mailbox size in bytes. |

If `ERR` evaluates to true then the operation failed.

```
CALL CROSS.POP3.STAT(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION, MCNT,
MBXSIZE)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END

PRINT "MAILBOX CONTAINS ":"MCNT:" MESSAGES"
PRINT "MAILBOX SIZE IS ":"MBXSIZE:" BYTES"
```

Retrieving a Mailbox Listing (LIST)

The LIST command can only be issued in the transaction state. It returns a message listing for a specific message or a listing of all the messages in the mailbox.

The call signature for `CROSS.POP3.LIST` is:

```
SUBROUTINE CROSS.POP3.LIST(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, MNUM, MCNT, MNUMS, MSIZES)
```

The parameters unique to this subroutine are:

| | |
|---------------|---|
| MNUM | The message number to list. If this is empty all messages are listed. |
| MCNT | The number of messages returned in the list. |
| MNUMS | The message numbers of the listed messages. |
| MSIZES | The message sizes in bytes of the listed messages. |

If **ERR** evaluates to true then the operation failed.

```
* List all messages in the mailbox
MNUM = ""
CALL CROSS.POP3.LIST(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION, MNUM,
MCNT, MNUMS, MSIZES)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END

PRINT "MAILBOX CONTENTS: "
FOR I = 1 TO MCNT
  PRINT MNUMS<I>:" ":MSIZES<I>
NEXT I
```

Retrieving a Message (RETR)

The RETR command can only be issued in the transaction state. It retrieves a message identified by message number.

The call signature for `CROSS.POP3.RETR` is:

```
SUBROUTINE CROSS.POP3.RETR(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, MNUM, MESSAGE)
```

The parameters unique to this subroutine are:

| | |
|----------------|---|
| MNUM | The message number to retrieve. |
| MESSAGE | The message contents retrieved. Use <code>CROSS.MIME.DECODE</code> to parse the message contents and headers. |

If **ERR** evaluates to true then the operation failed.

```
* Retrieve the first message in the mailbox
MNUM = "1"
CALL CROSS.POP3.RETR(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION, MNUM,
MESSAGE)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END
PRINT "RETRIEVED MESSAGE NUMBER ":MNUM
PRINT MESSAGE
```

Marking a message for deletion (DELETE)

The DELETE command can only be issued in the transaction state. It marks a message identified by message number for deletion. Deletion does not occur until the session enters the update state. The RSET command can be used to clear the deletion mark.

The call signature for `CROSS.POP3.DELETE` is:

```
SUBROUTINE CROSS.POP3.DELETE(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, MNUM)
```

The parameters unique to this subroutine are:

| | |
|------|--|
| MNUM | The message number to mark for deletion. |
|------|--|

If `ERR` evaluates to true then the operation failed.

```
* Retrieve the first message in the mailbox
MNUM = "1"
CALL CROSS.POP3.DELETE(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION, MNUM)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END
PRINT "MARKED MESSAGE NUMBER ":MNUM:" FOR DELETION"
```

The NOOP command

The NOOP command can only be issued in the transaction state. It performs no actions but confirms that the server is responding to commands.

The call signature for `CROSS.POP3.NOOP` is:

```
SUBROUTINE CROSS.POP3.NOOP(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION)
```

If `ERR` evaluates to true then the operation failed.

```
* Retrieve the first message in the mailbox
CALL CROSS.POP3.NOOP(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END
```

Unmarking messages for deletion (RSET)

The RETR command can only be issued in the transaction state. If any messages have been marked for deletion they are unmarked.

The call signature for `CROSS.POP3.RSET` is:

```
SUBROUTINE CROSS.POP3.RSET(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION)
```

If `ERR` evaluates to true then the operation failed.

```
* Retrieve the first message in the mailbox
CALL CROSS.POP3.RSET(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END
```

Retrieving part of a message (TOP)

The TOP command can only be issued in the transaction state. It retrieves the top `LCNT` lines of a message identified by message number.

The call signature for `CROSS.POP3.TOP` is:

```
SUBROUTINE CROSS.POP3.TOP(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, MNUM, LCNT, MESSAGE)
```

The parameters unique to this subroutine are:

| | |
|----------------|--|
| MNUM | The message number to retrieve. |
| LCNT | The number of lines to retrieve from the top of the message (this includes the header lines of a message). |
| MESSAGE | The lines retrieved. |

If `ERR` evaluates to true then the operation failed.

```
* Retrieve the first message in the mailbox
MNUM = "1"
LCNT = "10"
CALL CROSS.POP3.TOP(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION, MNUM,
LCNT, MESSAGE)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END
PRINT "RETRIEVED ":LCNT:" LINES FROM MESSAGE NUMBER ":MNUM
PRINT MESSAGE
```

Message Unique ID Listing (UIDL)

The UIDL command can only be issued in the transaction state. It returns a unique-id listing for a nominated message or for all messages in the mailbox. This unique-id can be used to identify a message between POP3 sessions.

The call signature for `CROSS.POP3.UIDL` is:

```
SUBROUTINE CROSS.POP3.UIDL(ERR, SLRESP, ALRESP, RESPDATA, MAT
POP3.SESSION, MNUM, MCNT, MNUMS, MUIDS)
```

The parameters unique to this subroutine are:

| | |
|--------------|--|
| MNUM | The message number to retrieve. If this is empty a listing of all messages in the mailbox is retrived. |
| MCNT | The number of messages listed. |
| MNUMS | The message numbers listed. |
| MUIDS | The unique ids for each message listed. |

If **ERR** evaluates to true then the operation failed.

```
* Retrieve a unique id list for all messaged in the mailbox
MNUM = ""
CALL CROSS.POP3.UIDL(ERR, SLRESP, ALRESP, RESPDATA, MAT POP3.SESSION, MNUM,
MCNT, MNUMS, MUIDS)
IF ERR EQ MAIL_RETURN_FAIL THEN
  * Error Handling Code
END
PRINT "UNIQUE ID LISTING FOR MAILBOX"
FOR I = 1 TO MCNT
  PRINT MNUMS<I>:" " :MUIDS<I>
NEXT I
```

Decoding a MIME Message

The mail suite allows you to decode a mime message or mime encoded body . This is done using the `CROSS.MIME.DECODE` routine. RFC 2822 and the associated MIME standards describe an internet message as a series of headers followed by a message body (which may then be further structured using MIME). This routine will decode any RFC 2822 compliant message (treating the body as a plain text body part) and will decode multipart MIME messages placing the contents of each MIME part in the `CONTENTS` array.

These routines refer to both RFC 822 and RFC 2822. The first is the original Internet Message Format standard for which the routines were written. RFC 2822 updates this standard. Although the Mail suite is compliant with RFC 2822, references to the earlier standard remain in constants and subroutine names.

The call signature for this routine is:

```
CROSS.MIME.DECODE(ERR, SLRESP, ALRESP, RESPDATA, MESSAGE, MAT
MESSAGE.HEADERS, MAT MIME.META, MAT CONTENTS)
```

In addition to the standard four described above, the following parameters may be passed into the subroutine:

| | |
|----------------------------|--|
| MESSAGE | An email message or mime encoded body. This will be parsed and its contents returned in <code>MESSAGE.HEADERS</code> , <code>MIME.META</code> and <code>CONTENTS</code> . |
| MAT MESSAGE.HEADERS | <p>A dimensioned array of size <code>RFC822_HEADER_DIM_LEN</code> (5)</p> <p>This contains the headers associated with the top level of <code>MESSAGE</code>. The elements of <code>MESSAGE.HEADERS</code> are:</p> <p>RFC822_HEADER_NAME (1) A list of header names, one per attribute.</p> <p>RFC822_HEADER_VALUE (2) A list of header values. A header name may have multiple values in which case these will be multi-valued. These have white space stripped but are not further processed.</p> <p>RFC822_HEADER_COMMS (3) If the header is listed in RFC 2822 as structured, it is parsed and any extracted comments are multi-valued in this element.</p> <p>RFC822_HEADER_MCNT (4) A count of the number of multi-values associated with each header name.</p> <p>RFC822_HEADER_PARSED (5) If the header is listed in RFC 2822 as structured, this is the parsed value of the header. Otherwise it contains a copy of the raw header value from <code>RFC822_HEADER_VALUE</code>.</p> |

| | |
|---|--|
| MAT MIME.META | A dimensioned array of size MIME_META_DIM_LEN (64) |
| | This contains the meta data for the MIME parts of MESSAGE each element contains an attribute for each MIME part extracted. |
| | MIME_META_CNT (1) |
| | The number of MIME body parts extracted from MESSAGE . This indicates the number of attributes in the remaining parts of MIME.META and the number of elements used in the CONTENTS array. |
| | MIME_META_PARENT (2) |
| | The index of the multipart parent of the current part. For the top part in a message this is empty. |
| | MIME_META_CHILD_CNT (3) |
| | A count of the number of child message parts extracted from this mime part. This will only be set for parts with MIME_META_MULTI set to true. |
| | MIME_META_CHILDREN (4) |
| | A multi-valued list of the indices of child MIME parts extracted from this MIME part. |
| | MIME_META_MULTIPART (5) |
| | If this MIME part is a multipart MIME part then this field will be true. |
| | MIME_META_MULTIPART_BOUNDARY (7) |
| | The boundary used to decode this multipart MIME part. |
| MIME_META_CONTENT_TYPE_MAJOR (9) | |
| The major content type defined for the MIME part. A list of MIME content types is maintained by IANA. Currently this will be one of application, audio, image, message, model, multipart, text or video . | |
| MIME_META_CONTENT_TYPE_MINOR (10) | |
| This is the minor content type defined for the MIME part. For example plain for a text/plain content type or octet-stream for an application/octet-stream content type. | |
| MIME_META_CONTENT_TYPE_NAME (11) | |
| An optional name associated with the MIME part. This is typically a file name. | |
| MIME_META_CONTENT_ID (12) | |
| An optional unique ID associated with the content. | |
| MIME_META_CONTENT_DISPOSITION (13) | |
| An optional indicator as to how the MIME part should be displayed to the user. Typical values are inline or attachment . | |
| MIME_META_CONTENT_DISPOSITION_FILENAME (14) | |

| | |
|--|--|
| | <p>An optional filename which may be supplied with the content-disposition header.</p> |
| | <p>MIME_META_CONTENT_DISPOSITION_CREATION_DATE (15) An optional creation date which may be supplied with the content-disposition header. This date will be in RFC822 format.</p> |
| | <p>MIME_META_CONTENT_DISPOSITION_MODIFICATION_DATE (16) An optional modification date which may be supplied with the content-disposition header. This date will be in RFC822 format.</p> |
| | <p>MIME_META_CONTENT_DISPOSITION_READ_DATE (17) An optional read date which may be supplied with the content-disposition header. This date will be in RFC822 format.</p> |
| | <p>MIME_META_CONTENT_DISPOSITION_SIZE (18) An optional file size which may be supplied with the content-disposition header.</p> |
| | <p>MIME_META_TEXT_CHARSET (19) Text MIME parts may nominate a character set to be used when displaying the part's contents. This defaults to US-ASCII if it is not supplied.</p> |
| | <p>MIME_META_TEXT_FORMAT (20) Text MIME parts may nominate a format parameter (see RFC 2646).</p> |
| | <p>MIME_META_APPLICATION_OCTET_STREAM_TYPE (21) MIME parts of type application/octet-stream may supply a type parameter. This is indicative data for the part which may be displayed to a user.</p> |
| | <p>MIME_META_APPLICATION_OCTET_STREAM_PADDING (22) MIME parts of type application/octet-stream may supply a padding parameter. If the source bit stream was not an even multiple of eight bits, this is the number of padding bits applied at the end of the stream to pad to a full byte.</p> |
| | <p>MIME_META_MESSAGE_PARTIAL_ID (23) MIME parts of type message/partial must supply a unique id so that the parts may be reassembled.</p> |
| | <p>MIME_META_MESSAGE_PARTIAL_NUMBER (24) MIME parts of type message/partial must supply a part number so that the parts may be reassembled in order.</p> |
| | <p>MIME_META_MESSAGE_PARTIAL_TOTAL (25) MIME parts of type message/partial must supply the total number of parts so that the application can determine when all parts have been assembled.</p> |
| | <p>MIME_META_PREAMBLE (26) MIME messages may contain plain text before the first MIME part. If present this is included here.</p> |

MAT CONTENTS

This a dimensioned array which must be large enough to hold every MIME part in MESSAGE. This will include multipart type MIME parts and multipart type sub-parts.

Sample code to decode and display the contents of a MIME message and all its MIME parts is shown below:

```

MESSAGE = a previously retrieved MIME message

DIM CONTENTS(3)
DIM META(MIME_META_DIM_LEN)

CALL CROSS.MIME.DECODE(ERR, SLRESP, ALRESP, RESPDATA, MESSAGE, MAT META, MAT
CONTENTS)
IF ERR THEN
  * Error handling code
END

PRINT "MIME PARTS ":META(MIME_META_CNT)
FOR I = 1 TO META(MIME_META_CNT)
  PRINT "PARENT      ":META(MIME_META_PARENT)<I>
  PRINT "CHILD CNT   ":META(MIME_META_CHILD_CNT)<I>
  PRINT "CHILDREN    ":CONVERT(@VM, ", ", META(MIME_META_CHILDREN)<I>)
  IF META(MIME_META_MULTIPART)<I> THEN
    PRINT "MULTIPART/ ":META(MIME_META_CONTENT_TYPE_MINOR)<I>
    PRINT "  DECODED      ":META(MIME_META_MULTIPART_DECODED)<I>
    PRINT "  BOUNDARY      ":META(MIME_META_MULTIPART_BOUNDARY)<I>
    PRINT "  BOUNDARY LEN   ":META(MIME_META_MULTIPART_BOUNDARY_LEN)<I>
  END ELSE
    PRINT "SINGLE - ":
    PRINT META(MIME_META_CONTENT_TYPE_MAJOR)<I>:
    PRINT "/":
    PRINT META(MIME_META_CONTENT_TYPE_MINOR)<I>

    PRINT "  CONTENT ID    ":META(MIME_META_CONTENT_ID)<I>
    PRINT "  DISPOSITION   ":META(MIME_META_CONTENT_DISPOSITION)<I>
    PRINT "  FILENAME      ":META(MIME_META_CONTENT_DISPOSITION_FILENAME)<I>
    BEGIN CASE
      CASE META(MIME_META_CONTENT_TYPE_MAJOR)<I> EQ "TEXT"
        PRINT "    TEXT PARAMETERS"
        PRINT "    CHARSET      ":META(MIME_META_TEXT_CHARSET)<I>
        PRINT "    FORMAT       ":META(MIME_META_TEXT_FORMAT)<I>

        CASE META(MIME_META_CONTENT_TYPE_MAJOR)<I> EQ "APPLICATION" AND
META(MIME_META_CONTENT_TYPE_MINOR)<I> EQ "OCTET-STREAM"
          PRINT "    APPLICATION PARAMETERS"
          PRINT "    TYPE         ":
          PRINT META(MIME_META_APPLICATION_OCTET_STREAM_TYPE)<I>
          PRINT "    PADDING      ":
          PRINT META(MIME_META_APPLICATION_OCTET_STREAM_PADDING)<I>

          CASE META(MIME_META_CONTENT_TYPE_MAJOR)<I> EQ "MESSAGE" AND
META(MIME_META_CONTENT_TYPE_MINOR)<I> EQ "PARTIAL"
            PRINT "    MESSAGE/PARTIAL PARAMETERS"
            PRINT "    ID           ":META(MIME_META_TEXT_ID)<I>
            PRINT "    NUMBER      ":META(MIME_META_TEXT_NUMBER)<I>
            PRINT "    TOTAL       ":META(MIME_META_TEXT_TOTAL)<I>

          END CASE
        PRINT "CONTENTS:"
        PRINT CONTENTS(I)
        PRINT "--EOF--"
        PRINT
      END
    END
  NEXT I

```

Validating an Email Address

The mail suite allows email addresses to be validated to conform with RFC 2822. The subroutine `CROSS.EMAILADDRESS.VALIDATE` will validate an email address and return just the email address with any comments or display name removed.

The call signature for this routine is:

```
CROSS.EMAILADDRESS.VALIDATE(ERR, SLRESP, ALRESP, RESPDATA, ADDRESS,
USEDNS)
```

In addition to the standard four described above, the following parameters may be passed into the subroutine:

| | |
|----------------|--|
| ADDRESS | An email address to be validated. If the address is valid, a canonical address without comments or display name will be returned in this parameter. |
| USEDNS | If true the hostname part of ADDRESS will be validated by a DNS query. The query will accept a hostname as valid if there is a Mail Exchanger (MX) record associated with the domain. |

Performing a DNS query

The mail suite allows DNS queries to be performed from TCL. The file `CROSS.RESOLV.CONF` may be created to hold name server information. If this file contains a record called `NS`, its contents will be used as a list of name server IP addresses. If the file or the `NS` record do not exist, the engine attempts to read `/etc/resolv.conf` to determine the name servers to contact.

Performing a DNS query from TCL

The routine `CROSS.DNS.QUERY` uses the following command format:

```
CROSS.DNS.QUERY NAME [TYPE]
```

The parameters to this program are

| | | | | | | | | | | | | | |
|--------------|--|----------|--------------|-----------|----------------|-----------|---------------------------|------------|---------------------|--------------|-----------------------------|------------|------------------------------|
| NAME | Domain name or IP address to query. | | | | | | | | | | | | |
| TYPE | Optional record type to request. By default for a domain name the routine attempts to find an A type record; for an IP address the routine attempts to find a PTR type record. Supported record types are as per RFC 1035, include: <table style="margin-left: 40px;"> <tr><td>A</td><td>Host Address</td></tr> <tr><td>MX</td><td>Mail exchanger</td></tr> <tr><td>NS</td><td>Authoritative name server</td></tr> <tr><td>PTR</td><td>Domain name pointer</td></tr> <tr><td>CNAME</td><td>Canonical name for an alias</td></tr> <tr><td>SOA</td><td>Start of a zone of authority</td></tr> </table> | A | Host Address | MX | Mail exchanger | NS | Authoritative name server | PTR | Domain name pointer | CNAME | Canonical name for an alias | SOA | Start of a zone of authority |
| A | Host Address | | | | | | | | | | | | |
| MX | Mail exchanger | | | | | | | | | | | | |
| NS | Authoritative name server | | | | | | | | | | | | |
| PTR | Domain name pointer | | | | | | | | | | | | |
| CNAME | Canonical name for an alias | | | | | | | | | | | | |
| SOA | Start of a zone of authority | | | | | | | | | | | | |

Sample output from this program is shown below. Each record type returned is printed. Record formats are specific to each record.

```
CROSS.DNS.QUERY CROSS.NET.AU ALL
Queried CROSS.NET.AU type ALL
Answers:
1 : CROSS.NET.AU
  Type MX
  Pref. 10
  Server librarian2.internal.CROSS.NET.AU
2 : CROSS.NET.AU
  Type A
  203.42.18.130
3 : CROSS.NET.AU
  Type SOA
  Server :librarian2.internal.CROSS.NET.AU
  Contact :craig.CROSS.NET.AU
  Serial :2001092001
  Refresh :10800 3h
  Retry :3600 1h
  Expire :604800 1w
  Min. TTL:86400 1d
4 : CROSS.NET.AU
  Type NS
  librarian2.internal.CROSS.NET.AU
Additional:
1 : librarian2.internal.CROSS.NET.AU
  Type A
  10.0.1.5
```