Micriμm

Empowering Embedded Systems

μC/POP3c

V1.01

User's Manual

www.Micrium.com

Registration
Please register the software via email. This way we can make sure you will receive updates or notifications of updates as soon as they become available. For registration please provide the following information:

- Your full name and the name of your supervisor
- Your company name
- Your job title
- Your email address and telephone number
- Company name and address
- Your company's main phone number
- Your company's web site address
- Name and version of the product

Please send this information to: **licensing@micrium.com**

Contact address

Micrium
949 Crestview Circle
Weston, FL 33327-1848
U.S.A.
Phone : +1 954 217 2036
FAX : +1 954 217 2037
WEB : **www.micrium.com**
Email : **support@micrium.com**

Manual versions

If you find any errors in this document, please inform us and we will make the appropriate corrections for future releases.

Manual Version	Date	By	Description
V1.00	2006/02/01	SR	First version.
V1.01	2006/04/25	SR	Code modifications

Table Of Contents

Introduction

POP3 (Post Office Protocol version 3) is a protocol designed to allow smaller nodes in the Internet not able to maintain a message transport system to manage mail in a useful fashion. The POP3 protocol is used to allow a workstation to retrieve mail that the server is holding for it.

µC/POP3c is an add-on product to µC/TCP-IP that implements the client POP3 protocol. µC/POP3c implements part of the following RFC:

RFC #1939	**ftp://ftp.rfc-editor.org/in-notes/rfc1939.txt**
RFC #2822	**ftp://ftp.rfc-editor.org/in-notes/rfc2822.txt**

This document describes how to configure and use the µC/POP3c module on top of µC/TCP-IP in a µC/OS-II environment. A Cogent CSB337 (ARM9) development platform with IAR compiler is used to demonstrate the typical application of this module, but other platforms and tool chains may be used as well.

Required modules version

The current version of the µC/POP3c module has been developed and tested using version **1.84** of µC/TCP-IP.

Chapter 2

Directories and Files

The code and documentation of the µC/POP3c module are organized in a directory structure according to "AN-2002, µC/OS-II Directory Structure". Specifically, the files are found in the following directories:

`\Micrium\Software\uC-POP3c`

This is the main directory for µC/POP3c.

`\Micrium\Software\uC-POP3c\Doc`

This directory contains the µC/POP3c documentation files, including this one.

`\Micrium\Software\uC-POP3c\Cfg\Template`

This directory contains a template of µC/POP3c configuration.

`\Micrium\Software\uC-POP3c\Source`

This directory contains the µC/POP3c source code. This protocol is implemented in two files:

```
pop3-c.c
pop3-c.h
```

Note that the '-c' at the end of `pop3` stands for client and thus contains 'client' side code. `pop3-c.h` is a header file containing client declarations for POP3.

Chapter 3

Using µC/POP3c

This chapter provides examples on how to use µC/POP3c. A Cogent CSB337 with a AT91RM9200 CPU (ARM9) running µC/OS-II and µC/TCP-IP was used to demonstrate its application, as illustrated in figure 3-1.

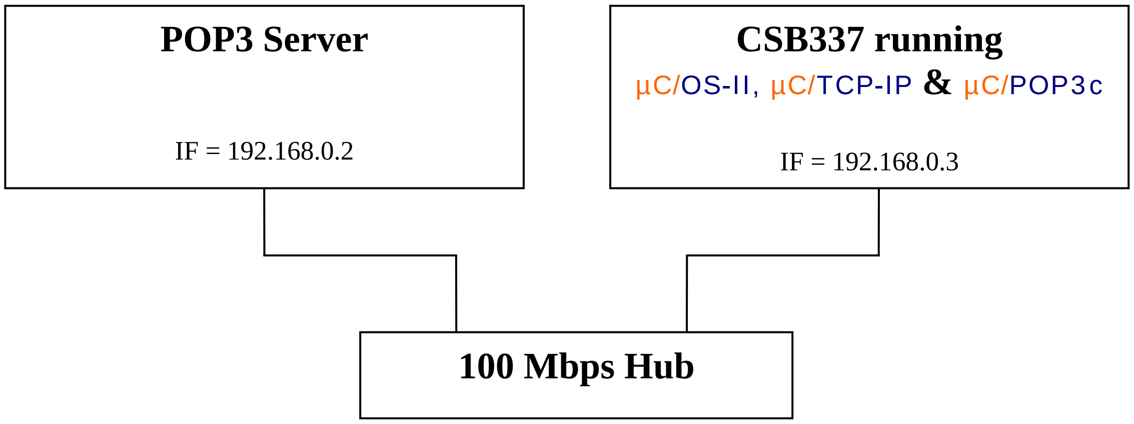

Figure 3-1, Test setup

3.01 µC/POP3c test code

The code in the next section assumes knowledge of µC/OS-II and µC/TCP-IP. Indeed, this section of code only concerns µC/POP3c and you need to be able to configure the real-time OS and the TCP stack in order to run it.

3.02 µC/POP3c test code, POP3c_Test()

Listing 3-1 is shown to demonstrate the µC/POP3c module capabilities. This code sequentially calls every exported functions of this module is order to illustrate its typical use.

Listing 3-1

```
void  POP3c_Test (void)
{
    NET_IP_ADDR  ip_server;
    NET_ERR      errMsg;
    NET_SOCK_ID  sock;

    CPU_INT16U   msg_qty;
    CPU_INT32U   mbox_size;
    CPU_INT16U   i;
    CPU_INT32U   msg_size;

    CPU_CHAR     msg_buf [MSG_BUF_SIZE];
    POP3c_MSG    msg;


    ip_server = NetASCII_Str_to_IP("192.168.0.2", &errMsg);                        (1)
    if (errMsg != NET_ASCII_ERR_NONE) {
        APP_DEBUG_TRACE("Error - NetASCII_Str_to_IP: %d\n", errMsg);
        return;
    }

    sock = POP3c_Connect(ip_server, 110, &errMsg);                                  (2)
    if (errMsg != POP3c_ERR_NONE ) {
        APP_DEBUG_TRACE("Error - POP3c_Connect: %d\n", errMsg);
        return;
    }

    POP3c_Authenticate(sock, "auser", "apassword", &errMsg);                       (3)
    if (errMsg != POP3c_ERR_NONE) {
        APP_DEBUG_TRACE("Error - POP3c_Authenticate: %d\n", &errMsg);
        return;
    }

    POP3c_MboxStat(sock, &msg_qty, &mbox_size, &errMsg);                           (4)
    if (errMsg != POP3c_ERR_NONE) {
        return;
    }
    APP_DEBUG_TRACE("POP3c_MboxStat: %d msg, %d bytes\n",
                     msg_qty,
                     mbox_size);

    for (i = 1; i <= msg_qty; i++) {
        POP3c_MsgStat(sock, i, &msg_size, &errMsg);                                (5)
        if (errMsg == POP3c_ERR_NONE) {
            APP_DEBUG_TRACE("Message %d: %d bytes\n", i, msg_size);

            POP3c_MsgRetrieve(sock,                                                (6)
                              i,
                              msg_buf,
                              MSG_BUF_SIZE,
                              DEF_NO,
                              &errMsg);

            APP_DEBUG_TRACE("Original message (buffer):\n");
            APP_DEBUG_TRACE("%s\n", msg_buf);


            POP3c_MsgRead(msg_buf,                                                 (7)
                          MSG_BUF_SIZE,
                          &msg,
                          &errMsg);

            APP_DEBUG_TRACE("Content of message %d\n\n", i);



            APP_DEBUG_TRACE("From     : %s\n",   msg.From);
            APP_DEBUG_TRACE("Sender   : %s\n",   msg.Sender);
            APP_DEBUG_TRACE("Date     : %s\n",   msg.Date);
            APP_DEBUG_TRACE("Reply to : %s\n",   msg.Reply_to);
            APP_DEBUG_TRACE("Subject  : %s\n\n", msg.Subject);
            APP_DEBUG_TRACE("%s\n\n", msg.Body);
        }
    }
    if (msg_qty >= 1) {
        POP3c_MsgDel(sock, 1, %errMsg);                                    (8)
    }

    POP3c_Disconnect(sock, &errMsg);                                       (9)
    APP_DEBUG_TRACE("Disconnected\n");

    return;
}
```

L3-1(1)	Convert the ASCII dotted-decimal notation to a network protocol IPv4 address.
L3-1(2)	Establish a TCP connection to the POP3 server (192.168.0.2) on port 110.
L3-1(3)	Authenticate on the POP3 server. This function relies on the USER and PASS commands (see `pop3-c.h` for details). As demonstrated in the above listing, the password is sent in clear text.
L3-1(4)	Get information on logged in maildrop (message quantity, except for deleted ones, and total maildrop size).
L3-1(5)	Get information on particular message (total message size, in octets).
L3-1(6)	Retrieve "raw" message from the POP3 server. Header and body are put into `msg_buf` (`MSG_BUF_SIZE` has to be set to a large enough value). Use the function `POP3c_MsgStat()` introduced at point L3-1(5) to determine a message's size. Note that providing a buffer that is too small for the whole message won't lead to runtime errors; however, it might prevent the application from running in a predictable way if the complete message was necessary for execution of the program.
L3-1(7)	Read (i.e. parse) message and fill in `POP3c_MSG` structure. This function takes in parameter the `msg_buf` previously filled by `POP3c_MsgRetrieve()` and extract a few pieces of information from its header. However, the addresses fields are not parsed (see section 3.04 for more information). Also, the `Message` field of this structure is in fact a pointer to the first bytes of the message body. Finally, note that `message_buffer` is not modified by this function.
L3-1(8)	Delete a message from the maildrop.
L3-1(9)	Disconnect from the server. The update mechanism takes place here (permanently removing messages set for deletion).

3.03 µC/POP3c module configuration

The µC/POP3c module has to be configured according to your specific needs. A template configuration file (pop3-c_cfg.h) is included in the module package (see Chapter 2, Directories and Files), and this configuration should be copied into your app_cfg.h file.

Here are the customizable variables:

```
#define  POP3c_CFG_IPPORT                110
```

This value sets the default port to use when calling POP3c_Connect() without specifying any particular port. Standard listening port for POP3 servers is 110.

3.04 µC/POP3c module limitations

This POP3c client implements a part of RFC 1939; not all commands have been implemented (see pop3-c.h for more details).

The function POP3c_MsgRead() does not parse addresses; they are set in the structure as received from the server. For instance, the structure returned (POP3c_MSG) might contain the address as well as the name associated with it. Application that needs only the address would have to provide its own parsing algorithm. Also, this structure only provide room for one address, so only the first address of an address list will be copied.

The µC/POP3c module only implements a basic message retrieval mechanism. Accordingly, no provision is made for attachments or message decoding. The application is hence responsible for taking appropriate actions should this sort of behavior be desirable.

3.05 µC/POP3c memory requirements

µC/POP3c only has one global variable in RAM: the response buffer (POP3c_Resp_Buf) The amount of memory needed by this variable corresponds to the value of RESP_BUF_LEN. See file pop3-c.h for more details.

Concerning the calling task's stack size, it should be larger than 128 bytes; a smaller stack could introduce run-time problems.

Chapter 4

µC/POP3c API Reference

This chapter provides a reference to the µC/POP3c API. Each of the user-accessible services is presented in alphabetical order. The following information is provided for each of those services:

- A brief description
- The function prototype
- The filename of the source code
- A description of the arguments passed to the function
- A description of the returned value(s)
- Specific notes and warnings on using the service

POP3c_Authenticate()

```
void POP3c_Authenticate(NET_SOCK_ID  sock,
                        CPU_CHAR    *username,
                        CPU_CHAR    *pswd,
                        NET_ERR     *perr);
```

File	Called from
POP3-C.C	Application

`POP3c_Authenticate()` logs the user into the system (the POP3 server).

Arguments

`sock`	Socket ID returned by `POP3c_Connect()`.
`username`	Username on the POP3 server, or NULL if not needed.
`pswd`	Password associated with the previous username, or NULL if not needed.
`perr`	Pointer to a variable that will hold the return error code from this function, which can be any of the following:

`POP3c_ERR_NONE`	No error, authentication successful.
`POP3c_ERR_ARG_TOO_LONG`	See note/warning #1.
`POP3c_ERR_NEG_RESP`	Negatve response from server.
`POP3c_ERR_RX_FAILED`	Error receiving data from socket connection.

Returned Values

void.

Notes/Warnings

1. As stated in RFC #1939, Section 'Basic Operation' "Each argument may be up to 40 characters long".
2. Passing a NULL argument for either the username of the password result in the associate command not to be send to the server.

POP3c_Connect()

```
NET_SOCK_ID POP3c_Connect(NET_IP_ADDR  ip_server,
                          CPU_INT16U   port,
                          NET_ERR     *perr);
```

File	Called from
POP3-C.C	Application

`POP3c_Connect()` establishes a TCP connection with the POP3 server.

Arguments

`ip_server`	IP address of the POP3 server to contact.
`port`	TCP port to use. If "0", `POP3c_CFG_IPPORT` is used.
`perr`	Pointer to a variable that will hold the return error code from this function, which can be any of the following:

`POP3c_ERR_NONE`	No error.
`POP3c_ERR_SOCK_OPEN_FAILED`	Error opening socket.
`POP3c_ERR_SOCK_CONN_FAILED`	Error connecting to server.
`POP3c_ERR_NEG_RESP`	Negatve response from server.
`POP3c_ERR_RX_FAILED`	Error receiving data from socket connection.

Returned Values

Socket descriptor if no error; -1 otherwise.

Notes/Warnings

1. If anything goes wrong while trying to connect to the server, the socket is closed by calling `NetSock_Close()`. Hence, all data structures are returned to their original state in case of a failure.

POP3c_Disconnect()

```
void POP3c_Disconnect(NET_SOCK_ID  sock,
                      NET_ERR     *perr);
```

File	Called from
`POP3-C.C`	Application

`POP3c_Disconnect()` closes the connection between the client and the server.

Arguments

`sock`	Socket ID returned by `POP3c_Connect()`.
`perr`	Pointer to a variable that will hold the return error code from this function, which can be any of the following:

`POP3c_ERR_NONE`	No error.

Returned Values

void.

Notes/Warnings

1. The possible error code returned by `NetSock_Close()` is not taken into account.

POP3c_MboxStat()

```
void POP3c_MboxStat(NET_SOCK_ID  sock,
                    CPU_INT32U  *msg_qty,
                    CPU_INT32U  *mbox_size,
                    NET_ERR     *perr);
```

File	Called from
POP3-C.C	Application

`POP3c_MboxStat()` gets the number of message(s) in the mailbox, as well as its total size.

Arguments

`sock`	Socket ID returned by `POP3c_Connect()`.
`msg_qty`	Number of message(s) on the server for this account.
`mbox_size`	Size of this mailbox (in bytes).
`perr`	Pointer to a variable that will hold the return error code from this function, which can be any of the following:

`POP3c_ERR_NONE`	No error.
`POP3c_ERR_NEG_RESP`	Negatve response from server.
`POP3c_ERR_RX_FAILED`	Error receiving data from socket connection.

Returned Values

void.

Notes/Warnings

1. If successful, the response for this command is in the format: "+OK nn mm" where "nn" correspond to the number of messages and "mm" is the total size of the maildrop in octets.
2. The values returned via the pointers (number of message(s) and mailbox size) does not include message set for deletion, as stated in RFC #1939, Section 'The TRANSACTION State : STAT Command' "Note that messages marked as deleted are note counted in either total".

POP3c_MsgDel()

```
void POP3c_MsgDel(NET_SOCK_ID  sock,
                  CPU_INT32U   msg_nbr,
                  NET_ERR     *perr);
```

File	Called from
POP3-C.C	Application

`POP3c_MsgDel()` deletes a specific message from the server.

Arguments

`sock`	Socket ID returned by `POP3c_Connect()`.
`msg_nbr`	Index of message to delete.
`perr`	Pointer to a variable that will hold the return error code from this function, which can be any of the following:

`POP3c_ERR_NONE`	No error.
`POP3c_ERR_NEG_RESP`	Negatve response from server.
`POP3c_ERR_RX_FAILED`	Error receiving data from socket connection.

Returned Values

void.

Notes/Warnings

1. The `msg_nbr` argument starts at the value "1". Message "0" does not exist.

POP3c_MsgRead()

```
void POP3c_MsgRead(CPU_CHAR   *msg_buf,
                   CPU_INT32U  buf_size,
                   POP3c_MSG  *msg,
                   NET_ERR    *perr);
```

File	Called from
POP3-C.C	Application

`POP3c_MsgRead()` reads and structures a message previously retrieved from a POP3 server.

Arguments

`msg_buf`	Buffer conteining a message received with `POP3c_MsgRetrieve()`.
`buf_size`	Size of `msg_buf`.
`msg`	Pointer to `POP3c_MSG` structure being filled by this function.
`perr`	Pointer to a variable that will hold the return error code from this function, which can be any of the following:

`POP3c_ERR_NONE`	No error.
`POP3c_ERR_INCOMPLETE_MSG`	Incomplete message in `msg_buf`.

Returned Values

void.

Notes/Warnings

1. From RFC 2822 (Internet Message Format), Section 'Lexical Analysis of Messages : General Description', "The body is simply a sequence of characters that follows the header and is separated from the header by an empty line (i.i., a line with nothing preceding the CRLF)". Hence, `GetLine()` will return a length of 2 when it encounters that delimitation.
2. Characters `<CRLF>` are not copied into the various structure's header fields. Instead, a '\0' character is appended.
3. The leading WSP (after the comma) is(are) not copied into the various structure's header fields.
4. If the character '\0' is found in the msg_buf before the end of the header, it means that buffer, and hence the message, is incomplete. However, there is no way to know if the message body was complete. One should always check the return value of `POP3c_MsgRetrieve()` for confirmation.

POP3c_MsgRetrieve()

```
void POP3c_MsgRetrieve(NET_SOCK_ID  sock,
                       CPU_INT32U   msg_nbr,
                       CPU_CHAR    *dest_buf,
                       CPU_INT32U   buf_size,
                       CPU_BOOLEAN  del_msg,
                       NET_ERR     *perr);
```

File	Called from
`POP3-C.C`	Application

`POP3c_MsgRetrieve()` gets a specific essage from the POP3 server.

Arguments

`sock`	Socket ID returned by `POP3c_Connect()`.
`msg_nbr`	Index of message of interest.
`dest_buf`	Pointer to allocated buffer used to copy the message.
`buf_size`	Size of `dest_buf`.
`del_msg`	Indicate if the message should be deleted from the server.
`perr`	Pointer to a variable that will hold the return error code from this function, which can be any of the following:

`POP3c_ERR_NONE`	No error.
`POP3c_ERR_DEST_BUF_TOO_SMALL`	`dest_buf` too small.
`POP3c_ERR_NEG_RESP`	Negatve response from server.
`POP3c_ERR_RX_FAILED`	Error receiving data from socket connection.

Returned Values

void.

Notes/Warnings

1. The `msg_nbr` argument starts at the value "1". Message "0" does not exist.
2. The client software is responsible for providing a large enough buffer in order to contain the whole message. Failure to do so might lead to runtime problems. If the buffer passed is too small for a given message, only the first buf_size - 1 bytes will be copied, followed by a '\0'. See POP3c_RespServerMulti() for more information.

POP3c_MsgStat()

```
void POP3c_MsgStat(NET_SOCK_ID  sock,
                   CPU_INT32U   msg_nbr,
                   CPU_INT32U  *msg_size,
                   NET_ERR     *perr);
```

File	Called from
`POP3-C.C`	Application

`POP3c_MsgStat()` the size of a specific message from the POP3 server.

Arguments

`sock`	Socket ID returned by `POP3c_Connect()`.
`msg_nbr`	Index of message of interest.
`msg_size`	Size of the message in bytes.
`perr`	Pointer to a variable that will hold the return error code from this function, which can be any of the following:

`POP3c_ERR_NONE`	No error.
`POP3c_ERR_ARG_TOO_LONG`	See note/warning #1.
`POP3c_ERR_NEG_RESP`	Negatve response from server.
`POP3c_ERR_RX_FAILED`	Error receiving data from socket connection.

Returned Values

void.

Notes/Warnings

1. The `msg_nbr` argument starts at the value "1". Message "0" does not exist.

Appendix A

µC/POP3c Licensing Policy

You need to obtain an 'Object Code Distribution License' to embed µC/POP3c in a product that is sold with the intent to make a profit. Each 'different' product (i.e. your product) requires its own license but, the license allows you to distribute an unlimited number of units for the life of your product. Please indicate the processor type(s) (i.e. ARM7, ARM9, MCF5272, MicroBlaze, Nios II, PPC, etc.) that you intend to use.

For licensing details, contact us at:

Micrium
949 Crestview Circle
Weston, FL 33327-1848
U.S.A.

Phone	: +1 954 217 2036
FAX	: +1 954 217 2037
WEB	: **www.micrium.com**
Email	: **licensing@micrium.com**

Appendix B

References

µC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
CMP Books, 2002
ISBN 1-57820-103-9