Introduction

µTasker is an operating system designed especially for embedded applications where a tight control over resources is desired along with a high level of user comfort to produce efficient and highly deterministic code.

The operating system is integrated with TCP/IP stack and important embedded Internet services along side device drivers and system specific project resources.

µTasker and its environment are essentially not hardware specific and can thus be moved between processor platforms with great ease and efficiency.

However the µTasker project setups are very hardware specific since they offer an optimal pre-defined (or a choice of pre-defined) configurations, taking it out of the league of "board support packages (BSP)" to a complete "project support package (PSP)", a feature enabling projects to be greatly accelerated.

This document discusses the implementation and use of the SMTP interface in the V1.3 release.

Introduction to the µTasker SMTP implementation

SMPT – Simple Mail Transfer Protocol – is used to send Emails over the internet and as such is a very widely used protocol.

µTasker supports sending textual emails without attachments, either with no authentication or optionally with LOGIN authentication. Combined with DNS support it can perform the transmission to an SMTP server at a know address such as ***smtp.provide.com***. If no DNS support is needed or wanted, it performs the same function to the fixed IP address of the SMTP server.

Note that it is generally recommended to use DNS since the SMTP servers do sometimes change IP addresses (much less common is a change of its web address) and this would require reconfiguring devices before Email transmission can function properly again.

The demo project includes optional support of browser based SMTP configuration and saving to FLASH. Alternatively such settings can be hard coded into the project. The demo project also includes a test of email transmission.

What use are Emails in an embedded project?

Of course an embedded device is not going to compose lengthy emails to keep its friends informed of what it has been up to, but it may have reason to inform someone of certain events that have taken place.

Image that an embedded device is used to monitor the wear of a component in a piece of machinery – say using a value obtained by periodically sampling a voltage. Perhaps the voltage indicates that the wear is getting close to a limiting value which will require part replacement or general service work to be performed. It may set an output to light up a warning LED to draw attention to the fact but what if no one looks at it or the machinery is generally not monitored by an operator? What might you do if you were employed to keep an eye on the machine? Well what is typical today is to write an email and send it to the service department of the company responsible for the maintenance of the equipment, suggesting that a service visit be planned within the next couple of weeks.

This is of course something that the embedded device could also do itself. It could also repeat the request say every day just to ensure that it is not forgotten. There are of course numerous other examples where such services can be very useful.

Configuring µTasker to use SMTP

The define `USE_SMTP` is used to activate the SMTP support in the project. This includes the contents of the file `smtp.c` as well as adding the `fnSmtp()` task. One TCP socket is reserved for use by the SMTP client.

There are a few configurations which are performed in `config.h` which can be adjusted to suit the project.

```
#define SMTP_MESSAGE_LEN   300  // Frame length of SMTP transmissions
   Transmitted TCP frames are built on stack so are represent only short
   occupation of memory. If only short emails are to be sent this value can
   be set to a value allowing the complete mail to be transmitted in one
   frame. If the processor used has tight memory restrictions small frame
   lengths ensure that the stack use is minimised but may result in the
   email transmission being performed in a number of small frames. In fact
   the speed of email transmission is generally not of great importance and
   so even quite small values will not cause problems.

#define USE_SMTP_AUTHENTICATION // support login when sending Email
   This define enables code to support login authentication. Setting the
   define doesn’t force login since it is optionally performed when the
   connection is established. If you need to be able to do authentication
   it will be necessary to activate this however.

#define SMTP_PARAMETERS    // support SMTP parameters in parameter system
   This define adds all important SMTP configuration settings to the
   parameter system in the demo project. These allow the user to configure
   the SMTP, save it and later change it if required. Without this support
   all configuration settings are hard coded and can therefore only be
   changed by recompiling the project.
```

Sending an Email

This is in fact very easy to do but does require a little bit of preparation…

Since we are SMTP client it is not necessary to previously configure the protocol. The email transmission is invoked by the following call.

```
fnConnectSMTP(ucSMTP_server, SMTP_LOGIN, fnEmailTest); // use login
fnConnectSMTP(ucSMTP_server, 0, fnEmailTest);          // no login
```

```
extern int  fnConnectSMTP
    (unsigned char *ucIP,
     unsigned char ucMode,
     const CHAR *(*fnCallback)(unsigned char, unsigned short *));
```

`ucSMTP_server` is the IP address of the SMTP server which is connected to. It is either known or resolved using DNS from the web address of the server. See the demo project (`application.c` for an example of how DNS can be started and the transmission takes place subsequently after the IP address has been resolved.

The mode is either SMTP_LOGIN or nothing. When login is specified, the SMTP client will first request that the server performs the login authentication sequence before sending the email. *Many SMTP servers require that this takes place and will refuse emails if this option is not used – and the number is growing since this is a method which helps avoid spam.*

`fnEmailTest` is a call back function which the application must handle during the process. It then supplies various details and also the email contents itself. The following is an extract from the demo project which will be used in the following discussion. It has been simplified slightly and supports SMTP options from the parameter system. See the demo project (`application.c`) for the original code.

```
// Email call back handler
//
static const CHAR *fnEmailTest
            unsigned char ucEvent, unsigned short *usData)
{
switch (ucEvent) {
case SMTP_GET_DOMAIN:  return cUserDomain;
case SMTP_USER_NAME: return temp_pars->temp_parameters.ucSMTP_user_name;
case SMTP_USER_PASS: return temp_pars->temp_parameters.ucSMTP_password;
case SMTP_GET_SENDER: return temp_pars->temp_parameters.ucSMTP_user_email;
case SMTP_GET_DESTINATION: return cEmailAdd;
case SMTP_GET_SUBJECT: return cSubject;
case SMTP_SEND_MESSAGE:
  {                              // here we can send our Email message
  unsigned short usEmailPosition = *usData;
    if (usEmailPosition >= (sizeof(cEmailText) - 1)) {
      *usData = 0;
      return 0;                   // Email has been completely sent
    }
    else {                        // remaining length
      *usData = (sizeof(cEmailText) - 1) - usEmailPosition;
    }                  // return a pointer to the next part of message
    return (const CHAR *)&cEmailText[usEmailPosition];
  }

case SMTP_MAIL_SUCCESSFULLY_SENT: break  // email successfully sent!!

// various possible errors - we could repeat or try with other settings
case ERROR_SMTP_LOGIN_FAILED: // user name and password incorrect
case ERROR_SMTP_LOGIN_NOT_SUPPORTED: // we are trying to use login but the
                                        server doesn't support it
case ERROR_SMTP_POLICY_REJECT:       // we are not using login but the
                                        server insists on it
        break;
}
return 0;
}
```

When the transmission process is started, the client doesn't know anything about the message to be sent, it needs only to know the IP address of the server to be contacted.

After successfully establishing a TCP connection on port 69 with the SMTP server there is a standard procedure to prepare for the actual email transmission which requires the application passing some details. As can be seen from the call back function code this involves various strings, such as email addresses, possibly user name and password, and a subject. After the initialisation procedure and possibly login it is up to the application to pass also the email content in chunks until it has completely been transmitted.

To see the sequence it is simplest to record the Ethernet activity (using Ethereal) when a test email is sent, which shows the following sequence [notice that commands and responses are terminated using CR + LF (0x0d 0x0a) or "\r\n"].

1. TCP Connection
2. The SMTP server response with a ready message (Code 220)
3. The SMTP client sends EHLO code with its domain information. Here the call back function is activated with event `SMTP_GET_DOMAIN` to delivers the string. In the demo project the string is defined in `config.h`
   ```
   #define OUR_USER_DOMAIN              "my_domain.com";
   ```
   The actual domain information supplied tends to be unimportant.
4. The SMTP server then sends the welcome message (Code 250) with some text like "please to meet you" followed by some details about what types of login are accepted.
5. If we are to do login this is where it is requested, if not we jump to step 11
   The SMTP client sends the AUTH command with the parameter LOGIN.
6. Should the SMTP server not support this method it will answer with a fatal error code 501 and the attempt is terminated. In this case it will be necessary to remove the LOGIN option and send without it.
   Assuming that the SMTP server does support LOGIN, it replies with the request code (334) and base64 coded text – which reads "VXNlcm5hbWU6" and is coded "Username:"
7. The SMTP client call back is now activated with the event `SMTP_USER_NAME` and is responsible to pass back a pointer to the user name. In the demo project this is either a fixed user name, defined in the project hardware header (eg: `app_hw_ne64.h`)
   ```
   #define SMTP_ACCOUNT_NAME         "smtp_user"
   ```
   or else it is a parameter string in the parameter system (see below for more details of configuring using a web browser).
   The string is coded to base64 format and sent to the SMTP server
8. The SMTP server then requests the password in a similar fashion using the request code 334 followed by "UGFzc3dvcmQ6", meaning "Password:".
   Note that this authentication is not very strong since it is in fact very easy to transfer between the coded and decoded strings – just search for base64 on the internet to find one of a variety of online encoding/decoding tools…
9. The SMTP client call back now receives the event `SMTP_USER_PASS` and can deliver a pointer to the password in a manor equivalent to the user name. This is either a fixed password in the project hardware header or a user parameter.
   ```
   #define SMTP_PASSWORD          "smtp_password"
   ```
10. The reaction of the SMTP server depends on whether it can find a matching user name and password combination or not.
    A code 535 indicates a failed login attempt and a code 235 indicates a successful

login. In the case of a failure the call back function receives the event `ERROR_SMTP_LOGIN_FAILED` or else the procedure continues.

11. Assuming either successful login or else no login (we have just jumped from step 5) the SMTP client call back is activated with the event `SMTP_GET_SENDER` and must return a pointer to the sender's email address. This is either a fixed string or a user parameter.
    `#define SENDERS_EMAIL_ADDRESS  "NE64@uTasker.com"` (eg. in `app_hw_ne64.h`).
    The user's email address is generally not important to being accepted and can therefore even be fictive.
12. The SMTP server responds with the OK (Code 250) which causes the SMTP client call back to be activated with the event `SMTP_GET_DESTINATION` and must now return a pointer to the destination email address.
    In the demo project this is a string initialised with a default address from `config.h`
    `#define DEFAULT_DESTINATION_EMAIL_ADDRESS "test_name@test.com";`
    It is then modified in the email test on the start page of the web server – see below. An application could of course also use a list of email addresses to be sent to depending on which actual message is to be transferred.
13. The SMTP server responds with OK (Code 250) which causes the SMTP client to send the command DATA, meaning that it now wants to send an email.
14. The SMTP server responds with the code 354 and usually a remark that the email can begin and should be terminated by a line with only a single '.' in it. This causes the SMTP client call back to be activated with the event `SMTP_GET_SUBJECT` since the first part of the email body contains also the email subject.
    `#define EMAIL_SUBJECT  "NE64 Test"  (eg. in app_hw_ne64.h).`
    The SMTP client now sends a first TCP frame to the SMTP server with a lead-in constructed from:
    ```
    To : test_name@test.com[CR + LF]
    Subject: NE64 Test[CR + LF]
    From: NE64@uTasker.com[CR + LF]
    [CR + LF]
    ```
15. The SMTP client call back routine is now responsible for passing the email contents. It is called with the event `SMTP_SEND_MESSAGE` and can return a pointer to the textual contents to be transferred.
    It is important to understand that the email transmission itself may require a number of TCP frames to be sent and so the event may have to be handled a number of times before the complete text has been transmitted. In case of a frame loss, the repeated portion of the email text also has to be regenerated in the call back routine.

    The project demo code illustrates how simple this however can be performed. The call back routine also receives a pointer to an offset in the email text, which is initially zero. It uses this to calculate the pointer to the next part of the message and also modifies it with the remaining text length after the frame has been transmitted. Repetitions are thus automatically performed if the SMTP client modifies the length back to the position or the previous frame. The end of transmission is detected when the transmitted length becomes equal to the email message itself.

    The email text is defined as a fixed string in the demo project:
    ```
    #define EMAIL_CONTENT   "Hello!!\r\nThis is an email message
    from the DEMO9S12NE64.\r\nI hope that you have received this
    test and have fun using the uTasker operating system with
    integrated TCP/IP stack.\r\r\nRegards your DEMO9S12NE64!!";
    ```

(eg. in app_hw_ne64.h).

16. The SMTP client inserts the EOM (end of message) sequence of a single line with just a '.' When the call back indicates that the complete message has been processed (when it returns a null pointer as response to the `SMTP_SEND_MESSAGE` event.
17. The SMTP server acknowledges the EOM with the code 250, after which the SMTP client sends the `QUIT` command.
18. The SMTP server initiates closing the connection after code 221, indicating that it is going to close.
19. The TCP connection is terminated by the server and the SMTP client call back routine receives a final confirmation event `SMTP_MAIL_SUCCESSFULLY_SENT`.

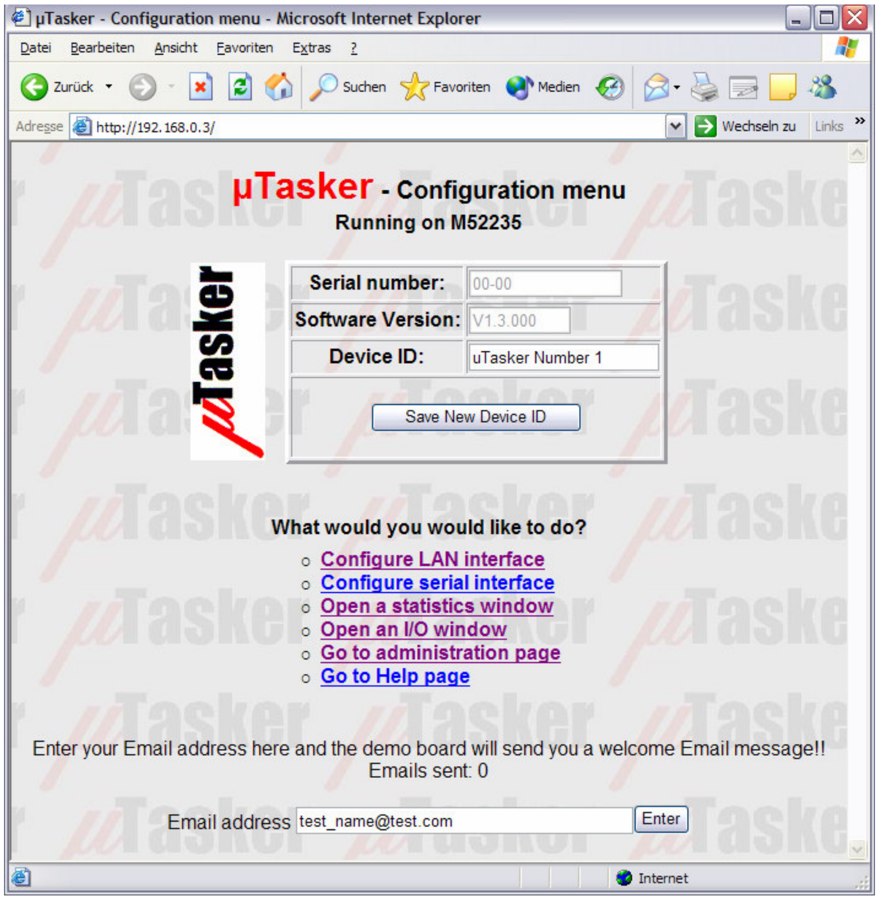
    This means that the email was successfully send to the SMTP server but not necessarily that it will actually reach its final destination!! The rest of the process does however not lie in the hands of the SMTP client…

The SMTP process is indeed quite straight forward and the SMTP code in the µTasker can be easily followed in smtp.c. It is also practical to test it with the simulator and step through any routines of interest. The only invisible extra function which it performs is to monitor the connection with a timer of 30s. If there is no SMTP activity after a TCP connection has been established, it will timeout to avoid the SMTP connection being blocked. In this case the SMTP client call back will receive the event `ERROR_SMTP_TIMEOUT`.

µTasker demo project support

The µTasker demo project includes an email test on the first page of its web server. It also includes optional configuration via web browser.

The SMTP support can be activated and configured as described earlier on in this document. The following is a typical screen shot of the start page of the demo project web server.


µTasker - Configuration menu - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Medien
Adresse http://192.168.0.3/
Wechseln zu Links
Internet

µTasker - Configuration menu
Running on M52235

Serial number:	00-00
Software Version:	V1.3.000
Device ID:	uTasker Number 1

Save New Device ID

What would you would like to do?

- Configure LAN interface
- Configure serial interface
- Open a statistics window
- Open an I/O window
- Go to administration page
- Go to Help page

Enter your Email address here and the demo board will send you a welcome Email message!!
Emails sent: 0

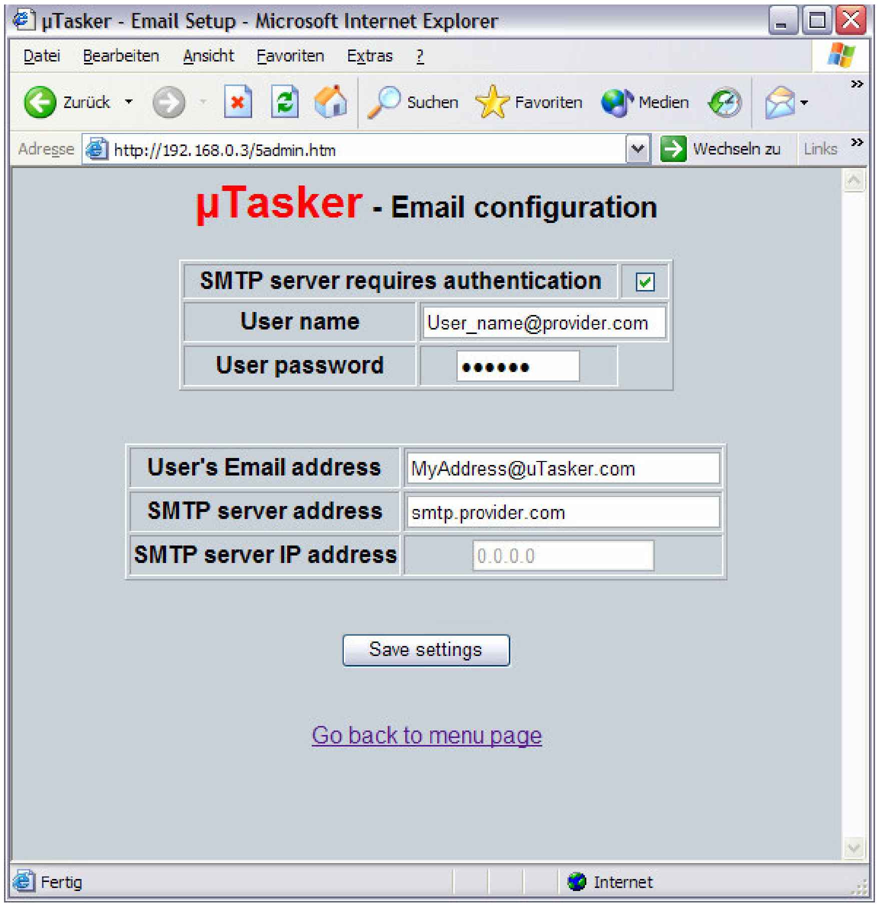
Email address test_name@test.com Enter

If the SMTP support is not activated in the project the Email address entry field will be disabled.

If the project is not configured with parameter support, all SMTP settings are fixed and can only be changed by recompiling the project.

Even when the project supports SMTP parameters the standard web pages do not support configuration of these, but I will explain here how it is easily changed to allow this. This also illustrates quite a powerful feature of the µTasker design since it allows such tasks to be performed on a temporary basis, which relieves the project of having to have full time space reserved for the functions.

- In the web pages folder to the project open the folder “AlternativePages” where you will find a HTML file specifically for the set up of emails (eg. MEmail.htm or a similar name which suits the processor project you are using).
- Copy this file via FTP and it will overwrite the administration page.
- Browse to the administration page to see the following:


µTasker - Email Setup - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten Medien
Adresse http://192.168.0.3/5admin.htm
Wechseln zu Links
Fertig
Internet

µTasker - Email configuration

SMTP server requires authentication	✓
User name	User_name@provider.com
User password	••••••

User's Email address	MyAddress@uTasker.com
SMTP server address	smtp.provider.com
SMTP server IP address	0.0.0.0

Save settings

Go back to menu page

Using this temporary web side you can comfortably set up the SMTP client to suit your requirements, either with or without LOGIN authentication. Once the parameters have been set to suit, save the settings and send a test email on the start side to be sure that all is working well.

> *Note that the SMTP IP address is disabled if DNS is active in the project (in this case the SMTP server address is important). However the resolved IP address will be displayed.*
>
> *If DNS is not active in the project the SMTP server address filed is disabled and the IP address of the server must be entered and save.*

Once you have finished the set up, simply reload the original administration page via FTP and all will be as before – apart from the fact that your SMTP setting are now as required.

Conclusion

The SMTP support, including LOGIN authentication, enables embedded devices to make use of Email. Practical and comfortable set up is supported, which has also shown how such functions can be temporarily loaded to a project so that they do not occupy valuable resources when no longer needed.