ESOS Mail – Example 1


```
void user_init(void) {
  esos_RegisterTask( sender_A );
  esos_RegisterTask( recipient_A );
}

ESOS_USER_TASK( sender_A ) {
  uint32_t                    u32_rnd;
  static   uint8_t            u8_cnt=0;
  static   ESOS_TASK_HANDLE   hTask;
  static   MAILMESSAGE        st_Message;

  ESOS_TASK_BEGIN();
  hTask = esos_GetTaskHandle( recipient_A );
  while (TRUE) {
    ESOS_TASK_MAKE_MSG_UINT8(st_Message, u8_cnt);
    ESOS_TASK_WAIT_ON_TASKS_MAILBOX_HAS_AT_LEAST(hTask,
           sizeof(uint8_t));
    printf("T0 sending MESSAGE %d\n", u8_cnt);
    ESOS_TASK_SEND_MESSAGE(hTask, &st_Message);

    u32_rnd = 1+(0x0F & esos_GetRandomUint32());
    ESOS_TASK_WAIT_TICKS( u32_rnd<<8);

    u8_cnt++;  if (u8_cnt>50) u8_cnt=0;
  } // endof while(TRUE)
  ESOS_TASK_END();
} // end sender_A()
```

get/store recipient task handle

create message in local storage

ESOS mailboxes are local to each task. Wait for recipient to have room for incoming message, then send.

wait random delay

ESOS Mail – Example 1


```
ESOS_USER_TASK( recipient_A ) {
  uint32_t                u32_rnd;
  uint8_t                 u8_x;
  static uint8_t          u8_cnt=0;
  static MAILMESSAGE      stMsg;

  ESOS_TASK_BEGIN();
  while (TRUE) {
    ESOS_TASK_WAIT_FOR_MAIL();
    while ( ESOS_TASK_IVE_GOT_MAIL() ) {
        ESOS_TASK_GET_NEXT_MESSAGE( stMsg );
        printf("Got a message from ");
        if (ESOS_DOES_TASK_HAVE_ID(sender_A,stMsg.u16_FromTaskID)) {
            printf("sender_A");
        } else {
            printf("UNKNOWN");
        }
        printf(" containing %d enroute time = %d ms\n",
             stMsg.au8_Contents[0],
             esos_GetSystemTick()-stMsg.u32_Postmark );
    } //endof while()
  } // endof while(TRUE)
  ESOS_TASK_END();
} // end recipient_A()
```

wait until mail arrives

consume messages until they are all gone

make local copy of message in `stMsg` allowing task mailbox to reclaim space

read contents to determine sender and postmark

ESOS Mail Service

ESOS_TASK_MAKE_MSG_EMPTY(st_Msg)
ESOS_TASK_MAKE_MSG_UINT8(st_Msg, u8_x)
ESOS_TASK_MAKE_MSG_UINT8_X2(st_Msg, u8_x1, u8_x2)
ESOS_TASK_MAKE_MSG_UINT16(st_Msg, u16_x)
ESOS_TASK_MAKE_MSG_UINT32(st_Msg, u32_x)
ESOS_TASK_MAKE_MSG_STRING(st_Msg, psz_x)

Creates a local copy of a mail message in the provided **st_Msg** mail message structure. Must be called in a task context as the created local message populates a field in **st_Msg** that encodes the current task as the sending task. Additional message constructor functions exist in the **esos_mail.h** file.

ESOS_TASK_MAILBOX_GOT_AT_LEAST_DATA_BYTES(pst_Task, u8_DataLen)

Is **TRUE** if the task denoted by **ESOS_TASK_HANDLE pst_Task** has room in its mailbox for a mail message with a payload of **u8_DataLen** bytes. Else, evaluates to **FALSE**.

ESOS_TASK_WAIT_ON_TASKS_MAILBOX_HAS_ROOM_MESSAGE(pst_Task, pst_Msg)

Blocks the current task until the task denoted by **ESOS_TASK_HANDLE pst_Task** has room in its mailbox for the mail message **pst_Msg**.

ESOS_TASK_SEND_MESSAGE(pst_ToTask, pst_Msg)

Immediately places local message **pst_Msg** in the mailbox of the task denoted by **pst_ToTask** which is an **ESOS_TASK_HANDLE**. This function assumes that the destination task **pst_ToTask** has sufficient room in its mailbox for the message. Therefore, this function is non-blocking.

ESOS_TASK_WAIT_ON_DELIVERY(pst_ToTask, pst_Msg)

Sends the local message **pst_Msg** to the task denoted by **ESOS_TASK_HANDLE pst_ToTask**, and blocks the current task until the receiving task **pst_ToTask** has read the mail message. This function assumes that the destination task **pst_ToTask** has sufficient room in its mailbox for the message.

ESOS_TASK_GET_NEXT_MESSAGE(pst_Msg)

Reads the next (oldest) message from the current task's mailbox and copies information into the **pst_Msg** mail message structure. If the sending task is blocked on message delivery of this mail message, calling this function will unblock that task.

ESOS_TASK_GET_LAST_MESSAGE(pst_Msg)

Reads the last (newest) message from the current task's mailbox and copies information into the **pst_Msg** mail message structure. All older messages in the current task's mailbox are flushed. If any sending tasks are blocked on message delivery of any messages in the current task's mailbox, calling this function will unblock those task(s).

ESOS Mail – Example 2

user_init() is not shown.

Task `recipient_B` is structured like task in previous example, except only checks for incoming mail periodically.

Once we have incoming message(s), process them until they are gone.

```
ESOS_USER_TASK( recipient_B ) {
  uint32_t                  u32_rnd;
  uint8_t                   u8_x;
  static uint8_t            u8_cnt=0;
  static MAILMESSAGE        stMsg;

  ESOS_TASK_BEGIN();
  while (TRUE) {
        u32_rnd = 1+(0x0F & esos_GetRandomUint32());
       ESOS_TASK_WAIT_TICKS( u32_rnd << 10);
       ESOS_TASK_WAIT_FOR_MAIL();
       while ( ESOS_TASK_IVE_GOT_MAIL() ) {
           ESOS_TASK_GET_NEXT_MESSAGE( stMsg );
           printf("Got a message from ");
           if (ESOS_DOES_TASK_HAVE_ID( sender_B0,
                        stMsg.u16_FromTaskID)) {
               printf("sender_B0");
           }
           else {
               printf("UNKNOWN");
           }
           printf (" containing %d enroute time = %d ms\n",
              stMsg.au8_Contents[0],
              esos_GetSystemTick()-stMsg.u32_Postmark );
    } //endof while()
  } // endof while(TRUE)
  ESOS_TASK_END();
} // end recipient_B()
```

Read each message (freeing mailbox space), decode message, and act upon it.

ESOS Mail – Example 2


```
ESOS_USER_TASK( sender_B0 ) {
  uint32_t                          u32_rnd;
  static   uint8_t                  u8_cnt;
  static   ESOS_TASK_HANDLE         hTask;
  static   MAILMESSAGE              st_Message;

  ESOS_TASK_BEGIN();
  hTask = esos_GetTaskHandle( recipient_B );
  while (TRUE) {
    if (ESOS_TASK_MAILBOX_GOT_AT_LEAST_DATA_BYTES( hTask,
                  sizeof(uint8_t) ) ) {
       ESOS_TASK_MAKE_MSG_UINT8(st_Message, u8_cnt);
       printf("B0 sending MESSAGE %d\n", u8_cnt);
       ESOS_TASK_SEND_MESSAGE(hTask, &st_Message);
    } else {
       printf("B0 doing useful work instead of mailing.
                  Discarding MESSAGE %d.\n", u8_cnt );
    }
    u8_cnt++;  if (u8_cnt>50) u8_cnt=0;
    u32_rnd = 1+(0x0F & esos_GetRandomUint32());
    ESOS_TASK_WAIT_TICKS( u32_rnd<<6 );
  } // endof while(TRUE)
  ESOS_TASK_END();
} // end sender_B0()
```

Store handle to recipient task.

Determine if recipient has mailbox room. If so send message. If not, this task can do something else.

create outgoing message in local storage

send mail message to recipient

If recipient does NOT have room in its mailbox, then this task can do something else and try sending mail again later.

ESOS Mail – Example 3

user_init() is not shown. Task sender_C0 (not shown) is identical to sender_B0 in previous example.

```
ESOS_USER_TASK( sender_C1 ) {
  uint32_t                          u32_rnd;
  static   uint8_t                  u8_cnt=100;
  static   ESOS_TASK_HANDLE         hTask;
  static   MAILMESSAGE              st_Message;

  ESOS_TASK_BEGIN();
  hTask = esos_GetTaskHandle( recipient_C );
  while (TRUE) {
    ESOS_TASK_MAKE_MSG_UINT8(st_Message, u8_cnt);
    ESOS_TASK_WAIT_ON_TASKS_MAILBOX_HAS_AT_LEAST(hTask,
              sizeof(uint8_t));
    u32_rnd = 1+(0x0F & esos_GetRandomUint32());
    if ( (u32_rnd % 4) == 0 ) {
        st_Message.u8_flags |= ESOS_MAILMESSAGE_REQUEST_ACK;
        printf("C1 sending ACKREQ MESSAGE %d\n", u8_cnt);
        ESOS_TASK_SEND_MESSAGE_WAIT_DELIVERY(hTask, &st_Message);
    } else {
        printf("C1 sending MESSAGE %d\n", u8_cnt );
        ESOS_TASK_SEND_MESSAGE(hTask, &st_Message);
    }
    u8_cnt++;  if (u8_cnt>150) u8_cnt=100;
    ESOS_TASK_WAIT_TICKS( u32_rnd<<7);
  } // endof while(TRUE)
  ESOS_TASK_END();
} // end sender_C1()
```

Task `recipient_C` is structured identically to recipient task in previous example.

Determine if recipient has mailbox room. If so send message. If not, this task can do something else.

On average, 1 in 4 messages in this task will request “delivery confirmation”.

This task is blocked until the recipient task “read” the mail message

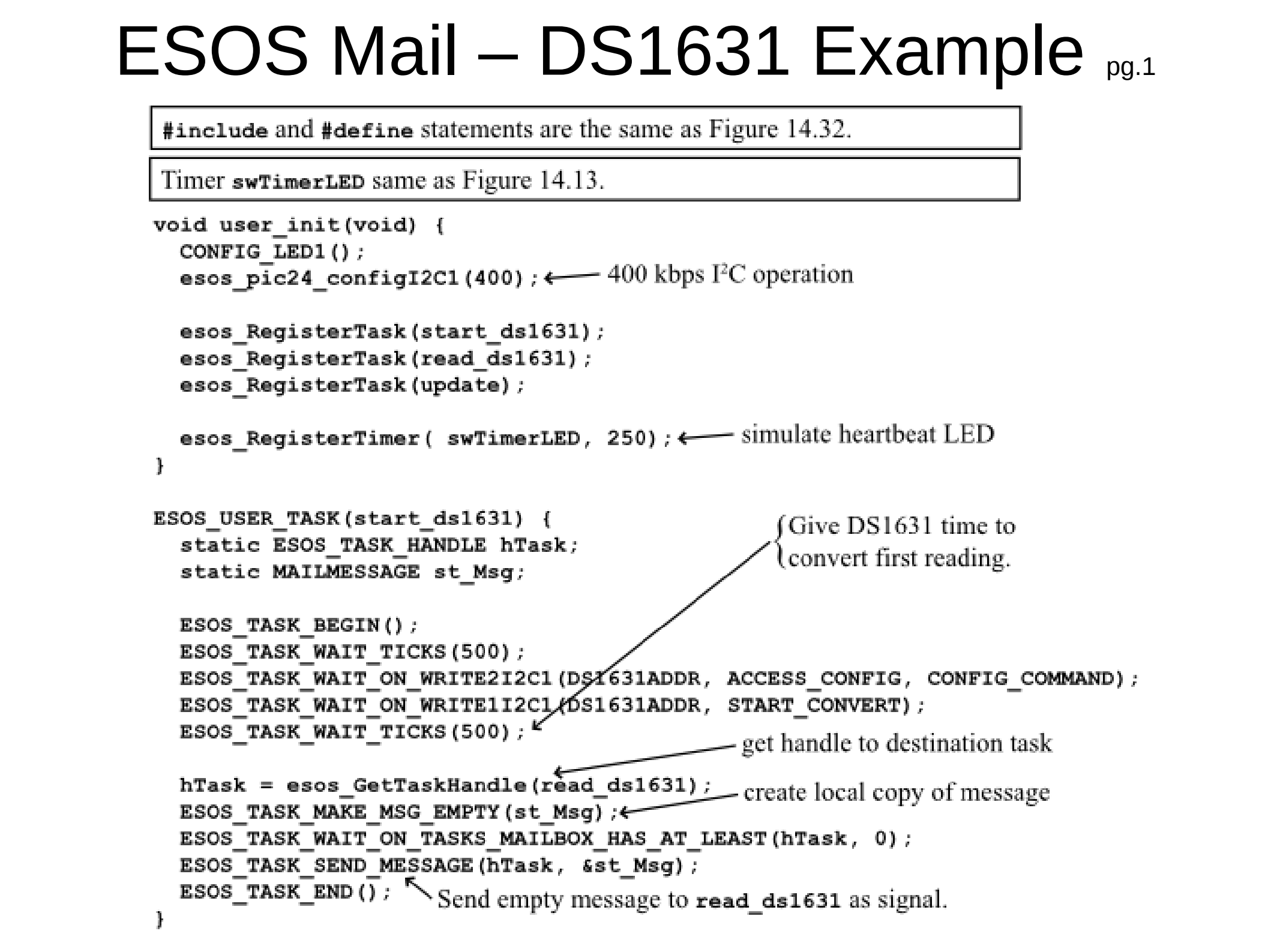
send a “normal” mail message (non-blocking)

ESOS Mail – DS1631 Example

`#include` and `#define` statements are the same as Figure 14.32.

Timer `swTimerLED` same as Figure 14.13.

```
void user_init(void) {
  CONFIG_LED1();
  esos_pic24_configI2C1(400);

  esos_RegisterTask(start_ds1631);
  esos_RegisterTask(read_ds1631);
  esos_RegisterTask(update);

  esos_RegisterTimer( swTimerLED, 250);
}

ESOS_USER_TASK(start_ds1631) {
  static ESOS_TASK_HANDLE hTask;
  static MAILMESSAGE st_Msg;

  ESOS_TASK_BEGIN();
  ESOS_TASK_WAIT_TICKS(500);
  ESOS_TASK_WAIT_ON_WRITE2I2C1(DS1631ADDR, ACCESS_CONFIG, CONFIG_COMMAND);
  ESOS_TASK_WAIT_ON_WRITE1I2C1(DS1631ADDR, START_CONVERT);
  ESOS_TASK_WAIT_TICKS(500);

  hTask = esos_GetTaskHandle(read_ds1631);
  ESOS_TASK_MAKE_MSG_EMPTY(st_Msg);
  ESOS_TASK_WAIT_ON_TASKS_MAILBOX_HAS_AT_LEAST(hTask, 0);
  ESOS_TASK_SEND_MESSAGE(hTask, &st_Msg);
  ESOS_TASK_END();
}
```



ESOS Mail – DS1631 Example


```
ESOS_USER_TASK(read_ds1631) {
  static uint8_t u8_lo, u8_hi;
  static MAILMESSAGE st_Msg;
  static ESOS_TASK_HANDLE h_Update;

  ESOS_TASK_BEGIN();
  ESOS_TASK_WAIT_FOR_MAIL();
  ESOS_TASK_GET_LAST_MESSAGE( &st_Msg);

  h_Update = esos_GetTaskHandle( update );
  while (TRUE) {
    ESOS_TASK_WAIT_ON_AVAILABLE_I2C();
    ESOS_TASK_WAIT_ON_WRITE1I2C1(DS1631ADDR, READ_TEMP);
    ESOS_TASK_WAIT_ON_READ2I2C1(DS1631ADDR, u8_hi, u8_lo);
    ESOS_TASK_SIGNAL_AVAILABLE_I2C();
    ESOS_TASK_MAKE_MSG_UINT8_X2(st_Msg, u8_hi, u8_lo);
    ESOS_TASK_WAIT_ON_TASKS_MAILBOX_HAS_AT_LEAST(
                h_TaskUpdate,2*sizeof(uint8_t));
    ESOS_TASK_WAIT_ON_DELIVERY(h_TaskUpdate, &st_Msg);
    ESOS_TASK_WAIT_TICKS(750);
  }
  ESOS_TASK_END();
}

ESOS_USER_TASK(update) {
  float f_tempC, f_tempF;
  static MAILMESSAGE st_Msg;
  int16_t i16_temp;

  ESOS_TASK_BEGIN();
  while (TRUE) {
    ESOS_TASK_WAIT_FOR_MAIL();
    ESOS_TASK_GET_LAST_MESSAGE(&st_Msg);
    i16_temp = st_Msg.au8_Contents[0];
    i16_temp = ((i16_temp<<8)|st_Msg.au8_Contents[1]);

    f_tempC = (float) i16_temp;
    f_tempC = f_tempC/256;
    f_tempF = f_tempC*9/5 + 32;
    printf("Temp is: 0x%0X, %4.4f (C), %4.4f (F)\n",
        i16_temp, (double) f_tempC, (double) f_tempF);
  }
  ESOS_TASK_END();
}
```

First message (an empty message) received must be from **start_ds1631** task. It signifies that DS1631 device is initialized and ready for use.

Read temperature from DS1631 device via I2C

Send message with DS1631 data payload to **update** task.

Block until sure that task **update** read the sent mail message.

Message has DS1631 data. Message arrival is signal to proceed.

Get the mail message containing the most recent temperature value. Reading message will signal task **read_ds1631** to continue.

Application uses **printf** for convenience.

user_init() is not shown.

swTimer LED() is identical to that in previous examples.

Task start_ds1631() is not shown.