


```

/*****\
*   Initialize LME library   *
\*****/
if (LM_Init() != LM_OK)
    return EXIT_FAILURE;

/*****\
*   Initialize Communication Port(s)   *
\*****/
com_port1 = LMX_ComPort_Open(LMX_COMPOR_T_NUMBER,
                            (char *)LMX_COMPOR_T_PATH, LM_B115200);

if(com_port1==LM_NULL)
    return EXIT_FAILURE; /* exit this program */

/*****\
*   Choose the modem (device)   *
*   (see LM_Enums.h for more)   *
\*****/
device = LM_DEVICE_XX_GenericGSM;

/*****\
*   Register the Device(s) to a (command) communications port   *
\*****/
hDevice = LM_Device_Register(device, com_port1);

if (hDevice == LM_NULL)
    return EXIT_FAILURE; /* exit this program */

/*****\
*   Register to receive Device events   *
\*****/
if (LM_Event_Register(hDevice, DEVICE_MSG_ANY) != LM_OK)
    return EXIT_FAILURE;

/*****\
*   Startup Device   *
\*****/
if(LM_Device_Startup(hDevice) != LM_OK)
    return EXIT_FAILURE; /* exit this program */

/*****\
*                               *
*   MAIN LOOP                               *
*                               *
*   This is an example of a typical main processing loop   *
*   used in many embedded applications.   *
\*****/
main_loop=0; /* used to exit the while(1) (for example purposes only) */
while(1)
{
    /* check if modem is in READY state */
    if(LM_State_getCurState(hDevice)==LM_DEVICE_STATE_READY)
    {
        /*----- Send an SMS -----*/
        if(!SMS_sent) /* ensure this runs once, otherwise we'll get a lot of SMS's...*/
        {

            /* tell modem which character set we are using
            ("American","European","Chinese etc") */

```

```

strcpy((char *)command_parm.SMS.char_set, "IRA");
LM_Device_SendCommand(hDevice, DEVICE_CMD_SMS_SET_CHAR_SET,
                      &command_parm);

/* tell modem to send an SMS message */
strcpy((char *)command_parm.SMS.phone_number, "8015129966");
strcpy((char *)command_parm.SMS.text_msg, "Hello from Linkmaxx");
if(LM_Device_SendCommand(hDevice, DEVICE_CMD_SMS_SEND_TEXT,
                        &command_parm)==LM_OK)
    printf("\nSMS queued to be delivered to modem.\n");
else
    printf("!!!ERROR trying to send SMS!!!\n");

SMS_sent=1; /* set flag so that no more SMS's are sent */
}
}

/*****\
* Let Linkmaxx do its Processing *
\*****/

LM_Process(100); /* time in milliseconds */

/* LME ideal delay is 100mS, but MUST pass (as close
* to) ACTUAL value as possible */

/* If a system timer/clock is not available, choose a
* 'best guess'. Best guess here is 100mS == 100000uS
* which is used in usleep(100000)/Sleep(100) below */

/* for example purposes, we break out of the loop after a while;
* don't want a PC to 'hang' here forever */
if (main_loop > 500) break; /* run for x loops. x*100mS=total runtime */
main_loop++;

/* Let the CPU sleep for 100mS */
#if LM_PLATFORM_UNIX
usleep(100*MICROSECONDS); /* FYI: this is NOT an ANSI-C function */
#elif LM_PLATFORM_WIN32
Sleep(100);
#elif LM_PLATFORM_EMBEDDED
#endif

} /* end while(1) */

/*****\
* Shut down Devices & Cleanup *
\*****/

/* close down device - prevents Device memory corruption */
LM_Device_Shutdown(hDevice);

/* free up registered ports */
LM_Device_DeRegister(hDevice);

return EXIT_SUCCESS;
} /* end main() */

/*****//**
* @brief Response FROM commands sent to the device are passed

```

using this function to the application using LME.

```
*
*
* @param[in] hDevice      device device handle
* @param[in] success     success of the command sent (OK, Error, Timed-out)
* @param[in] xerror      eXtended error code (if any)
* @param[in] dev_port    communications port that the device is assigned to
* @param[in] cmd         command that was sent
* @param[in] cmd_string  text string version of command
* @param[in] data        structure containing data - This must be used or copied
*                        before the function returns, as LME does not keep a
*                        copy of this data
*
* @return                OK/Error
*****/
void LMX_Command_CB(LM_Device_h hDevice, LM_Error_e success, LM_XError xerror,
                  LMX_ComPort_h dev_port, LM_UINT16 cmd, char *cmd_string,
                  LM_CommandData_u *data)
{
    char success_str[80];

    /*-----*\
    *      Check if parm(s) are valid
    \*-----*/
    /* check if device is valid */
    if (hDevice==LM_NULL)
    {
        printf("Invalid device handle (hDevice) in LMX_Command_CB().\n");
        return;
    }
    /* check if port is valid */
    if (dev_port==LM_NULL)
    {
        printf("Com port has not been initialized!\n");
        return;
    }

    /*-----*\
    *      Check the command's success state
    \*-----*/
    switch (success)
    {
        case(LM_OK):                strcpy(success_str, "OK");          break;
        case(LM_ERROR):             strcpy(success_str, "ERROR");      break;
        case(LM_TIMEDOUT):          strcpy(success_str, "TIMED OUT");  break;
        case(LM_NOT_SUPPORTED):     strcpy(success_str, "TIMED OUT");  break;
        case(LM_XERROR):
            sprintf(success_str, "ERROR #%d (%s)", xerror,
                LM_Debug_XERRORtoSTR(xerror));
            break;
        default:
            printf("Unknown case in LMX_Command_CB()\n");
            break;
    }

    /*-----*\
    *      Process the commands result
    \*-----*/
    switch(cmd)
    {
        /* commands that don't return a value (other than OK/ERROR) */
        case(DEVICE_CMD_CHECK_IF_ALIVE):
        case(DEVICE_CMD_PBOOK_WRITE):
        case(DEVICE_CMD_REG_STATUS_REPORTING):
        case(DEVICE_CMD_SMS_SEND_TEXT):
        case(DEVICE_CMD_ECHO_OFF):
        case(DEVICE_CMD_SET_BAUDRATE):
        case(DEVICE_CMD_SMS_SET_CHAR_SET):
            printf("%s returned %s.\n", LM_Debug_CMDtoSTR(
                (LM_DEVICE_FUNC_NAME_e)cmd), success_str);
            break;
    }
}
```

```

case(DEVICE_CMD_GET_SOFTWARE_VERSION):
    if ((success==LM_OK) && (data!=LM_NULL))
        printf("DEVICE_CMD_GET_SOFTWARE_VERSION returned %s\n",
                data->version.swVer);
    else
        printf("DEVICE_CMD_GET_SOFTWARE_VERSION returned %s.\n",
                success_str);
    break;
case(DEVICE_CMD_GET_SIGNAL_QUALITY):
    if ((success==LM_OK) && (data!=LM_NULL))
        printf("DEVICE_CMD_GET_SIGNAL_QUALITY returned SSI=%d BER=%d\n",
                data->SignalQuality.strength,data->SignalQuality.BER);
    else
        printf("DEVICE_CMD_GET_SIGNAL_QUALITY returned %s.\n", success_str);
    break;
case(DEVICE_CMD_PBOOK_READ):
    if ((success==LM_OK) && (data!=LM_NULL))
    {
        printf("DEVICE_CMD_PBOOK_READ returned %s\n",
                data->PBook.phone_number);
        printf("DEVICE_CMD_PBOOK_READ returned %s\n",
                (char *)data->PBook.name);
    }
    else
        printf("DEVICE_CMD_PBOOK_READ returned %s.\n", success_str);

    break;
case(DEVICE_CMD_SMS_READ):
    if ((success==LM_OK) && (data!=LM_NULL))
    {
        printf("DEVICE_CMD_SMS_READ returned...\n");
        printf(" From: %s\n",data->SMS.phone_number);
        printf(" Message: %s\n\n",data->SMS.text_msg);
    }
    else
        printf("DEVICE_CMD_SMS_READ returned %s.\n", success_str);

    break;
case(DEVICE_CMD_REG_STATUS):
    if ((success==LM_OK) && (data!=LM_NULL))
        printf("DEVICE_CMD_REG_STATUS returned %d\n",
                data->NetworkInfo.operatorRegStatus);
    else
        printf("DEVICE_CMD_REG_STATUS returned %s\n", success_str);
    break;
case(DEVICE_CMD_USER):
    if ((success==LM_OK) && (data!=LM_NULL))
    {
        printf("DEVICE_CMD_USER(%s) returned %s.\n",
                cmd_string, success_str);

        if(data->ATuser.num_params>0)
            printf(" response 1 = [%s]\n",data->ATuser.parm1);

        if(data->ATuser.num_params>1)
            printf(" response 2 = [%s]\n",data->ATuser.parm2);

        if(data->ATuser.num_params>2)
            printf(" response 3 = [%s]\n",data->ATuser.parm3);

        if(data->ATuser.num_params>3)
            printf(" response 4 = [%s]\n",data->ATuser.parm4);

        if(data->ATuser.num_params>4)
            printf(" response 5 = [%s]\n",data->ATuser.parm5);

        if(data->ATuser.num_params>5)
            printf(" response 6 = [%s]\n",data->ATuser.parm6);
    }
    else

```

```

        printf("DEVICE_CMD_USER returned %s.\n", success_str);
break;
default:
    printf("Command (%s) complete, but NO callback LMX_Command_CB()\n",
        LM_Debug_CMDtoSTR((LM_DEVICE_FUNC_NAME_e)cmd));
break;

} /* end switch cmd */

printf("\n"); /* print out formatting */

return;
} /* LMX_Command_CB() */

/*****
* @brief          Event messages coming FROM the device are passed back
*                using this function.
*
* @param[in] hDevice  device device handle
* @param[in] msg      message type enum (see LM_Device.h)
* @param[in] data     structure containing data - This must be used or copied
*                    before the function returns, as LME does not keep a
*                    copy of this data
*
* @return         OK/Error
*****/
void LMX_Event_CB(LM_Device_h hDevice, LM_DeviceMsgName_e msg,
    LM_MessageParm_u *data)
{

    /* Check if parm(s) are valid */
    if (hDevice==LM_NULL)
    {
        printf("Invalid device handle (hDevice) in LMX_Event_CB().\n");
        return;
    }
    if (data==LM_NULL)
        return;

    /* process the device event message */
    switch(msg)
    {
        case(DEVICE_MSG_REG_STATUS):
            printf(">DEVICE_MSG_REG_STATUS event message: %d\n",
                data->regStatus.operatorRegStatus);
            break;
        case(DEVICE_MSG_SMS_REC'D):
            printf(">DEVICE_MSG_SMS_REC'D event message number = %d\n",
                data->smsLocNum);
            break;
        case(DEVICE_MSG_SIGNAL_QUALITY):
            printf(">DEVICE_MSG_SIGNAL_QUALITY: SSI=%d BER=%d\n",
                data->SignalQuality.strength,data->SignalQuality.BER);
            break;
        /* device STATE changes */
        case(DEVICE_MSG_DEVICE_STATUS):
            switch(data->deviceState)
            {
                case(LM_DEVICE_STATE_NONE):
                    printf("< Device STATE: none >\n"); break;
                case(LM_DEVICE_STATE_START):
                    printf("< Device STATE: Starting >\n"); break;
                case(LM_DEVICE_STATE_INITIALIZING):
                    printf("< Device STATE: Initializing >\n"); break;
                case(LM_DEVICE_STATE_NO_SERVICE):
                    printf("< Device STATE: Searching for Service >\n"); break;
                case(LM_DEVICE_STATE_NO_RESPONSE):

```

```

        printf("< Device STATE: NOT Responding >\n"); break;
    case(LM_DEVICE_STATE_READY):
        printf("< Device STATE: READY >\n"); break;
    case(LM_DEVICE_STATE_STARTING_DATA):
        printf("< Device STATE: Starting Data mode >\n"); break;
    case(LM_DEVICE_STATE_DATA_ONLY):
        printf("< Device STATE: In DATA ONLY mode >\n"); break;
    case(LM_DEVICE_STATE_MUX_MODE):
        printf("< Device STATE: In CMD/Data MUX mode >\n"); break;
    case(LM_DEVICE_STATE_SHUT_DOWN):
        printf("< Device STATE: Shutting down >\n"); break;
    default: ;
}
break;
default:
    printf("Message (%d) Received, but NO callback LMX_Event_CB()\n",msg);
    break;
} /* end switch event */

return;
} /* LMX_Event_CB() */

```