

# kucomms userspace programmers guide

## Table of Contents

Introduction.....	1
Defining and registering callbacks.....	1
Sending a message.....	2

## Introduction

In order to create a userspace application that can communicate with a kernel module, it is necessary to define three callback functions and then run the main loop of the message endpoint.

## Defining and registering callbacks

The first step is to define 3 classes, the user callbacks will be members of these classes.

```
#include "MessageManager.h"

class MyMessageHandler : public MessageHandler
{
public:
    bool hlr(const struct Message * message,
             MessageQueueWriter & tx_msgq,
             std::vector<MessageQueueWriter> & tx_msgq_list);
};

class MyWorkHandler : public WorkHandler
{
public:
    bool hlr(std::vector<MessageQueueWriter> & tx_msgq_list);
};

class MyTimerHandler : public TimerHandler
{
public:
    void hlr(const __u64 time,
             std::vector<MessageQueueWriter> & tx_msgq_list);
};
```

The next step is to declare the handler methods. The methods shown below have no implementation and are examples only.

```

bool
MyMessageHandler::hlr(const struct Message * message,
                     MessageQueueWriter & tx_msgq,
                     std::vector<MessageQueueWriter> & tx_msgq_list)
{
    return true;
}

bool
MyWorkHandler::hlr(std::vector<MessageQueueWriter> & tx_msgq_list)
{
    return false;
}

void
MyTimerHandler::hlr(const __u64 time,
                    std::vector<MessageQueueWriter> & tx_msgq_list)
{
}

```

The last step is to run the main loop of the message endpoint.

```

static bool g_stopped = false;

void
terminate_signal_handler(int sig)
{
    g_stopped = true;
}

int main(int argc, char ** argv)
{
    signal(SIGTERM, terminate_signal_handler);

    MyMessageHandler msghler;
    MyWorkHandler workhler;
    MyTimerHandler timerhler;

    bool ok = MessageManager::run(
        "/dev/kucomms_myname"
        1024*1024,
        g_stopped,
        msghler,
        workhler,
        timerhler);

    return 0;
}

```

## Sending a message

A message is sent by calling the add() method of the object passed to the caller.

Below are two examples of sending a message.

```

bool
MyMessageHandler::hlr(const struct Message * message,
                      MessageQueueWriter & tx_msgq,
                      std::vector<MessageQueueWriter> & tx_msgq_list)
{
    // Send the message received back to the sender
    bool ok = tx_msgq_list[0].add(message);
    return true;
}

bool
MyWorkHandler::hlr(std::vector<MessageQueueWriter> & tx_msgq_list)
{
    DataMessage msg(10); // Build a message
    Message * message = msg.get();
    __u8 * data = msg.get_data();
    message->m_type = 0;
    message->m_id = id;
    message->m_userValue = 0;
    for (__u32 u0=0; u0<msg.get_data_length(); u0++) data[u0] = 0;

    bool ok = tx_msgq_list[0].add(msg.get()); // Send message

    return false;
}

```