Host Interface Protocol

# Introduction

The Host Interface Protocol (HIO) is a frame-based protocol between a host and the Talaria TWO device. This protocol can be used over UART or SPI transports. This application note covers the fundamentals of developing HIO based application on both host and the Talaria TWO EVK.

# HIO Protocol Overview

There are three types of messages:

1. A request sent from host to device.
2. A response message sent form device to host.
3. An indication message sent from device to host for asynchronous information such as Wi-Fi scan results or device wake-up notifications.

All requests are preceded by a header:

|  |
| --- |
| struct hio\_msghdr {  uint8\_t group;  uint8\_t msgid;  uint16\_t trxid;  }; |

where,

1. group: group identifier
2. msgid: message identifier
3. trxid: a number that can be used by the host to sequence requests (it is just returned in response messages)

The message header and body are preceded by a two-byte (little endian) length of the header and body (excluding the length).

Graphical user interface

Description automatically generated with low confidence

Figure 1: HIO message framing

# Sample Host application: hio\_query

This basic application sends a hio\_query\_req to Talaria TWO and prints the output of the response of Talaria TWO.

## Programming Talaria TWO

For this sample application any existing Talaria TWO app with HIO enabled such as the STW application is used:

|  |
| --- |
| ./script/boot.py --device /dev/ttyUSB2 --reset=evk42\_bl --speed 921600 ./apps/stw/bin/stw.elf |

## Building the host application

|  |
| --- |
| cd sdk/examples/using\_hio/hio\_host  make |

This generated the hio\_query among other sample host applications.

## Run hio\_query

|  |
| --- |
| ./hio\_query /dev/ttyUSB2 |

## Expected host application output

|  |
| --- |
| Status: 0  Version: 0  Ngroups: 4  maxsize: 1020  fw\_version: $Id: git-  patch\_rev: $Patch: g  nmsg: 7 528 1288 2310 |

## Sending an HIO message from host to Talaria TWO

To send a HIO message, send\_command which is defined in hio\_transport.c is used:

|  |
| --- |
| int send\_command(int group\_id, int msg\_id, void \*data, int size, int transid)  {  int len;  char actual\_pay\_load[512] = {0,};  struct hio\_header\_info header = { 0, };  header.groupid = group\_id;  header.msgid = msg\_id;  header.trandid = transid;  len = sizeof(header) + size;  header.len = len - 2;  if (data)  memcpy(actual\_pay\_load + sizeof(header), data, size);  memcpy(actual\_pay\_load, &header, sizeof(header));  return write\_hio\_message(actual\_pay\_load, len);  }  int write\_hio\_message(char \*message, int msg\_len) {  int bytes\_written = 0;  while (bytes\_written < msg\_len) {  bytes\_written += write(gfd, message + bytes\_written, 1);  }  return bytes\_written;  } |

hio\_query.c

|  |
| --- |
| …  struct hio\_query\_req query\_req;  send\_command(HIO\_GROUP\_HIO, HIO\_QUERY\_REQ, &query\_req, sizeof(query\_req), 0);  ... |

Here, send\_command is used and the HIO group, msgid, size of HIO req structure, and trxid are specified.

After receiving this HIO message, Talaria TWO responds with another HIO message to the host which has the same header as above. This header can be used to identify the next action with the HIO message.

To handle the returning HIO message, a thread that reads the serial port waiting for a valid HIO message is created.

hio\_query.c

|  |
| --- |
| static void\* hio\_response\_reader(void \*p) {  uint32\_t msg\_count = 0;  while (!gExit) {  int read\_count = 0;  uint32\_t res\_len;  char msg\_len[4] = {0,};  while(read\_count < 2)  {  read\_count += read\_hio\_message(msg\_len+read\_count, 2-read\_count);  }  msg\_len[2] = 0x00;  msg\_len[3] = 0x00;  res\_len = msg\_len[0] & 0x000000FF;  if (msg\_len[1] != 0) {  res\_len = ((uint8\_t)msg\_len[0]&0xFF) + (msg\_len[1]<<8) & 0xFFFF;  }  sleep(0.2);  if(res\_len > 0)  {  char \*res\_payload = malloc(res\_len+1);  read\_count = 0;  while(read\_count < res\_len)  {  read\_count += read\_hio\_message((res\_payload)+read\_count, res\_len-read\_count);  }  process\_response(res\_len, res\_payload);  }  }  return NULL;  } |

There is also a function called process\_response() that parses the received response by its groupid and msgid.

|  |
| --- |
| static void process\_response(uint32\_t res\_len, char\* res\_payload)  {  struct hio\_res\_header res\_header = {0,};  memcpy(&res\_header, res\_payload, sizeof(struct hio\_res\_header));  uint32\_t payloadLength = res\_len - sizeof(struct hio\_res\_header);  char\* payload = malloc(payloadLength);  if(res\_header.group\_id == HIO\_GROUP\_HIO)  {  switch(res\_header.res\_code)  {  case 0x80:  {  struct hio\_query\_rsp \*rsp = malloc(sizeof(struct hio\_query\_rsp));  memcpy(rsp, res\_payload+sizeof(struct hio\_res\_header), sizeof(struct hio\_query\_rsp));  printf("Status: %d\n", rsp->status);  printf("Version: %d\n", rsp->version);  printf("Ngroups: %d\n", rsp->ngroups);  printf("maxsize: %d\n", rsp->maxsize);  printf("fw\_version: %s\n", rsp->fw\_rev);  printf("patch\_rev: %s\n", rsp->patch\_rev);  uint16\_t nmsg[rsp->ngroups];  memcpy(nmsg, (res\_payload+sizeof(struct hio\_res\_header)+sizeof(struct hio\_query\_rsp)), (rsp->ngroups)\*2);  printf("nmsg: ");  for(int i = 0; i < rsp->ngroups; i++)  {  printf("%d ", nmsg[i]);  }  printf("\n");  break;  }  default:  break;  }  }  else  {  printf("Group Code: 0x%X\n", res\_header.group\_id);  printf("Response Code: 0x%X\n", res\_header.res\_code);  printf("payload length: %d\n", payloadLength);  printf("Payload: \n" );  for(int i = 0; i < payloadLength; i ++)  {  printf("%X", \*(payload+i));  }  printf("\n");  }  } |

# Creating custom HIO group and commands

## Host side application

To create custom set of commands, select an available groupid. The highest possible groupid value is 0xFF. Each HIO message request, response, and indication will need its own msgid. An appropriate reference for this is: hio\_custom\_def.h

For this sample application, custom HIO command group with a single request, response, and indication structure is created.

The sample application sends a ping message from the host and Talaria TWO will send a HIO message response pong back to the host. An indication to the host is sent every 5 seconds.

hio\_custom\_def.h

|  |
| --- |
| #define HIO\_GROUP\_CUSTOM 0xFF  #define CUSTOM\_PING\_REQ 0x00  #define CUSTOM\_PING\_RSP 0x80  #define CUSTOM\_PING\_IND 0xC0  struct custom\_ping\_req {  uint32\_t msg\_length;  char msg[0]; };  struct custom\_ping\_rsp {  uint32\_t cpr\_status;  uint32\_t msg\_length;  char msg[0]; };  struct custom\_ping\_ind {  uint32\_t msg\_len;  char msg[0]; }; |

Here, custom HIO group with a single req, rsp, and ind structure is created.

hio\_custom.c

|  |
| --- |
| …  int main(int argc, char \*argv[])  {  int ping\_req\_count = 0;  if (init\_hio\_transport(argv[1]) < 0) {  printf("Failed to open serial device %s\n", argv[1]);  return -1;  }  pthread\_t thread\_id;  pthread\_create(&thread\_id, NULL, hio\_response\_reader, NULL);  uint16\_t trxid = 0;  char\* msg = "ping";  uint32\_t index = 0;  for(ping\_req\_count = 0;ping\_req\_count < 10;ping\_req\_count++)  {  struct custom\_ping\_req \*ping\_req = malloc(sizeof(struct custom\_ping\_req) + strlen(msg) + 1);  ping\_req->msg\_length = strlen(msg);  memset(ping\_req->msg,0, ping\_req->msg\_length+1);  memcpy(ping\_req->msg, msg, ping\_req->msg\_length);  send\_command(HIO\_GROUP\_CUSTOM, CUSTOM\_PING\_REQ, ping\_req, sizeof(struct custom\_ping\_req) + strlen(msg), trxid++);  printf("\*\*\*Sending ping request from host:%u\*\*\*\*\*\*\*\r\n",index);  index++;  sleep(1);  free(ping\_req);  sleep(2);  … |

Like the hio\_query example, send\_command()is used to send a HIO message to Talaria TWO.

To process the HIO message response from Talaria TWO, the groupid and msgid are added to process\_response method which processes the HIO messages.

|  |
| --- |
| …  else if(res\_header.group\_id == HIO\_GROUP\_CUSTOM)  {  switch(res\_header.res\_code)  {  case 0x80:  {  struct custom\_ping\_rsp \*ping\_rsp = malloc(payloadLength);  memcpy(ping\_rsp, res\_payload+sizeof(struct hio\_res\_header), payloadLength);  printf("Status: %u\n", ping\_rsp->cpr\_status);  printf("Message: %s\n\n", ping\_rsp->msg);  free(ping\_rsp);  break;  }  case 0xC0:  {  struct custom\_ping\_ind \*ping\_ind = malloc(payloadLength);  memcpy(ping\_ind, res\_payload+sizeof(struct hio\_res\_header), payloadLength);  printf("\*\*\*Received custom\_ping\_ind\*\*\*\n");  printf("Message: %s\n\n", ping\_ind->msg);  free(ping\_ind);  break;  }  case 0xC2:  {  struct custom\_datafromt2\_ind \*t2data\_ind = malloc(payloadLength);  memcpy(t2data\_ind, res\_payload+sizeof(struct hio\_res\_header), payloadLength);  printf("\*\*\*Received t2data\_ind\*\*\*\n");  printf("Message: %s\n\n", t2data\_ind->msg);  free(t2data\_ind);  break;  }  default:  break;  }  }  … |

## Talaria TWO Application

As with the host side application, there is also a need to specify the grpid and msgid to create custom set of commands and the req, rsp, ind structures. They are defined in hio\_custom\_def.h.

Before using any HIO commands, first register them using hio\_api\_init and create custom hio\_api stuct.

|  |
| --- |
| …  static const struct hio\_api custom\_api = {  .group = HIO\_GROUP\_CUSTOM,  .num\_handlers = 2,  .handler = {  custom\_ping,  custom\_datafromhost,  NULL,  }  };  …  int  int main(void)  {  os\_printf("Custom HIO api\n");  if(hio\_api\_register(&custom\_api, NULL) == 0){  os\_printf("Successfully registered HIO message group\r\n");  }else{  os\_printf("Failed to register HIO message group\r\n");  }  while (true) {  os\_msleep(5000);  os\_printf("Available heap:%d\r\n",os\_avail\_heap());  }  return 0;  }  … |

In this code, the struct custom\_api contains the handlers for each of the HIO message request to be received. The first handler, custom\_ping, coincides with HIO message req with msgid = 0x00 and if there is a second one it will be msgid= 0x01.

Since each handler coincides with a HIO message request, this is also where HIO message response would be sent back. This can be done with hio\_write\_msg or hio\_response\_status if status value needs to be returned.

|  |
| --- |
| static struct packet\* custom\_ping(void \*ctx, struct packet \*msg)  { uint32\_t status = 0;  struct custom\_ping\_req\* ping\_req = packet\_data(msg);  uint32\_t msg\_length = ping\_req->msg\_length;  struct pfrag \*frag = NULL;  os\_printf("Got ping req: %.\*s \n",msg\_length, ping\_req->msg);  char\* pong = "pong";  uint32\_t pong\_length = strlen(pong);  struct packet \*rsp;  frag = pfrag\_alloc(pong\_length);  memcpy(pfrag\_insert\_tail(frag,pong\_length), pong, pong\_length);  rsp = create\_custom\_ping\_rsp(pong\_length, status);  packet\_add\_frag(rsp, frag);  hio\_write\_msg(rsp, HIO\_GROUP\_CUSTOM , CUSTOM\_PING\_RSP, 0);  return NULL;  } |

## Run custom HIO Sample applications

1. Build Talaria TWO sample application

|  |
| --- |
| cd sdk/examples/using\_hio  make |

This should generate t2\_hio\_custom.elf.

1. Programming the Talaria TWO

|  |
| --- |
| cd sdk  ./script/boot.py --device /dev/ttyUSB2 --reset=evk42\_bl --speed 921600 ./examples/using\_hio/bin/t2\_hio\_custom.elf |

1. Build the Host application

|  |
| --- |
| cd sdk/examples/using\_hio/hio\_host  make |

1. Run the Host application

|  |
| --- |
| ./hio\_custom /dev/ttyUSB2 |

1. Expected Output
2. Talaria TWO output:

|  |
| --- |
| Got ping req: ping  Sent ping Response  Got data: InnoPhase is a fabless semiconductor company specializing in extreme low power wireless solutions. The company s developed the industry’s first digital PolaRFusion radio architecture to shatter the low power barrier of wireless communications. Combining this groundbreaking RF signal processing Technology with embedded processing will enable our company to revolutionize the IoT edge-computing industry. … |

1. Host application output:

|  |
| --- |
| Received custom\_ping\_rsp  Status: 0  Message: pong  Received custom\_ping\_rsp  Status: 0  Message: pong  Received custom\_ping\_rsp  Status: 0  Message: pong  Received custom\_ping\_rsp  Status: 0  Message: pong |