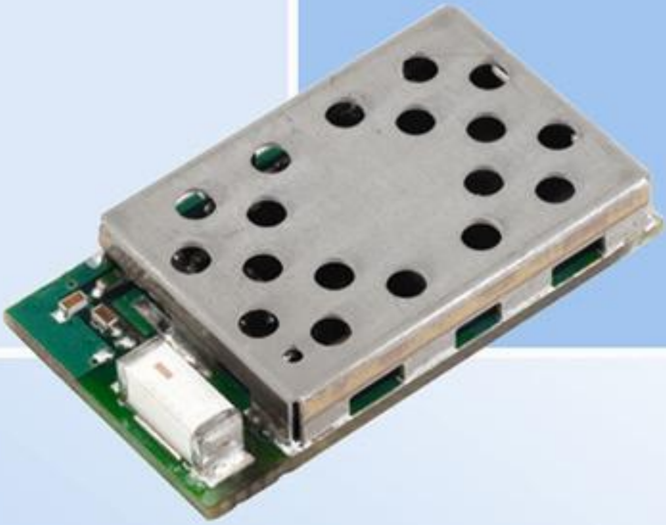
# BTM44x

## USER MANUAL



www.lairdtech.com

Laird TECHNOLOGIES

Innovative Technology
for a Connected World

Laird Technologies is the world leader in the design and manufacture of customized, performance-critical products for wireless and other advanced electronics applications.

Laird Technologies partners with its customers to find solutions for applications in various industries such as:

Network Equipment

Telecommunications

Data Communications

Automotive Electronics

Computers

Aerospace

Military

Medical Equipment

Consumer Electronics

Laird Technologies offers its customers unique product solutions, dedication to research and development, as well as a seamless network of manufacturing and customer support facilities across the globe.

LWS-UM-BTM44x 1111

This material is preliminary

Information furnished by Laird Technologies in this specification is believed to be accurate. Devices sold by Laird Technologies are covered by the warranty and patent indemnification provisions appearing in its Terms of Sale only. Laird Technologies makes no warranty, express, statutory, and implied or by description, regarding the information set forth herein. Laird Technologies reserves the right to change specifications at any time and without notice. Laird Technologies' products are intended for use in normal commercial and industrial applications. Applications requiring unusual environmental requirements such as military, medical life-support or life-sustaining equipment are specifically not recommended without additional testing for such application.

Limited Warranty, Disclaimer, Limitation of Liability

## Revision History

| Revision | Description |
|---|---|
| Version 1.0 | 9/10/2011 – Initial Release |

# TABLE OF CONTENTS

# Chapter 1
# Product Description

The BTM44x Bluetooth® modules from Laird Technologies have been designed to meet the needs of developers who wish to add robust, short range Bluetooth data connectivity to their products. They are based on the market leading Cambridge Silicon Radio BC04 chipset, providing exceptionally low power consumption with outstanding range. They support the latest Bluetooth® Version 2.1 Specification, providing the important advantage of Secure Simple Pairing, which improves security and enhances the ease of use for end customers.

With physical sizes as small as 12.5 x 18.0mm and best of class, low-power operation, these modules are the ideal choice for applications where designers need both performance and minimum size. For maximum flexibility in systems integration, the modules are designed to support a separate power supply for I/O.

To aid product development and integration, Laird Technologies has integrated a complete Bluetooth protocol stack within the modules, including support for multiple Bluetooth Profiles. The modules are fully qualified as Bluetooth End Products, allowing designers to integrate them within their own products with no further Bluetooth Qualification. They can then list and promote their products on the Bluetooth website free of charge.

Support for Serial Port Profile (SPP), Human Interface Device (HID) profile and Health Device Profile (HDP) are included in the module. The support of the Bluetooth Sig's Health Device Profile makes this the ideal module for development of Continua compliant medical and wellness devices. By default the Health Device Profile supports the ISO/IEEE 11073-10415 device specialization for weigh scales, but additional specializations can be made available upon request.

Communication is available to the module over a serial UART utilizing either a custom Multi-point Packet Protocol API or comprehensive AT commands. Combined with a low cost developer's kit, this ensures that the choice of Laird Technologies Bluetooth modules guarantees the fastest route to market.

## FEATURES AND BENEFITS

- Bluetooth® v2.1+EDR
- Adaptive Frequency Hopping to cope with interference from other wireless devices
- Secure Simple Pairing support
- External or internal antenna options
- Comprehensive AT interface for simple programming
- Alternate Packet based interface for complex programming
- Bluetooth® END Product Qualified
- Compact size
- Class 2 output – 4dBm
- Low power operation
- UART interface

## APPLICATIONS

- Embedded Devices
- Phone Accessories
- Security Devices
- Medical and Wellness Devices
- Automotive Applications

Bluetooth® Advertising

## BLUETOOTH® PROFILES SUPPORTED

- Serial Port Profile (SPP)
- Human Interface Device (HID) Profile Host and device supported
- Health Device Profile (HDP): Agent supported
- IEEE Device Specialization 11073-10415 (Weight Scale)
- Bluetooth® Advertising
- ePOS

# Chapter 2
# Hardware Specifications

| Pin | Signal | Description | Voltage Specification |
|---|---|---|---|
| 1 | Unused | | |
| 2 | GND | | |
| 3 | UART_CTS | Clear to Send I/P | VUSB |
| 4 | UART_RXD | Receive data I/P | VUSB |
| 5 | UART_RTS | Request to Send O/P | VUSB |
| 6 | UART_TXD | Transmit data O/P | VUSB |
| 7 | GND | | |
| 8 | SPI_CSB | SPI bus chip select I/P | VIO |
| 9 | SPI_MISO | SPI bus serial O/P | VIO |
| 10 | SPI_MOSI | SPI bus serial I/P | VIO |
| 11 | SPI_CLK | SPI bus clock I/P | VIO |
| 12 | VDD_USB | USB & UART supply voltage | |
| 13 | VDD_IO | I/O supply voltage | |
| 14 | VDD_IN | Main supply voltage | |
| 15 | GND | | |
| 16 | PCM_IN | PCM clock I/P | VIO |
| 17 | PCM_SYNC | PCM sync I/P | VIO |
| 18 | PCM_CLK | PCM clock I/P | VIO |
| 19 | PCM_OUT | PCM Data O/P | VIO |
| 20 | RESET | Module reset I/P | See note 2 |
| 21 | GPIO5 | I/O for host | VIO |
| 22 | GPIO2 / UART_DCD | I/O for host | VIO |
| 23 | GND | | |
| 24 | Unused | | |
| 25 | Unused | See note 3 | |
| 26 | Unused | See note 3 | |
| 27 | Unused | See note 3 | |
| 28 | GND | See note 3 | |
| 29 | ANT (BTM44X) | Antenna connection (50 ohm | See note 3 |
| 30 | GND | See note 3 | |
| 31 | Unused | See note 3 | |
| 32 | Unused | See note 3 | |
| 33 | Unused | See note 3 | |
| 34 | Unused | See note 3 | |
| 35 | Unused | See note 3 | |
| 36 | Unused | See note 3 | |
| 37 | Unused | See note 3 | |
| 38 | Unused | | |
| 39 | Unused | | |
| 40 | Unused | | |
| 41 | GND | | |
| 42 | GPIO1 / UART_RI | I/O for host | VIO |
| 43 | GPIO9 / UART_DTR | I/O for host | VIO |
| 44 | GPIO10 / UART_DSR | I/O for host | VIO |
| 45 | GND | | |
| 46 | D- | Not used for AT module | VUSB |
| 47 | D+ | Not used for AT module | VUSB |
| 48 | GPIO7 | I/O for host | VIO |
| 49 | GPIO6 | I/O for host | VIO |
| 50 | GPIO4 | I/O for host | VIO |

## Pin Definitions

Notes:

1 Unused pins may have internal connections and must not be connected.

2 Reset input is active low. Input is pulled up to VDD_IN via 22k. Minimum reset pulse width is 5ms.

3 Pins 25-37 should be left not connected on modules with integrated antenna (BTM441/3)

## Physical Specifications

| Categories | Feature | Implementation |
|---|---|---|
| Physical | Dimensions | 12.5mm x 18.0 x 3.4mm (BTM440/2) |
| | Weight | 3 grams |
| Environmental | Operating Temperature | -30°C to +85°C |
| | Storage Temperature | -40°C to +85°C |
| Miscellaneous | Lead free | Lead-free and RoHS compliant |

## Electrical Specifications

| Categories | Feature | Implementation |
|---|---|---|
| Wireless Specification | Bluetooth® | Version 2.1+EDR |
| | Transmit Class | Class 2 |
| | Frequency | 2.402 – 2.480 GHz |
| | Channels | 79 channels Frequency Hopping<br>Adaptive Frequency Hopping |
| | Max Transmit Power | +4 dBm at antenna pad (BTM440/2)<br>+4 dBmi from integrated antenna (BTM441/3) |
| | Min Transmit Power | -27 dBm at antenna pad (BTM440/2)<br>-27 dBmi from integrated antenna (BTM441/3) |
| | Receive Sensitivity | -84 dBm |
| | Range | 30 m |
| | Data Transfer Rate | Up to 350 kbps |

## Operational Specifications

**Recommended Operating Conditions**

| OPERATING CONDITIONS | MIN | MAX |
|---|---|---|
| VDD_USB (USB compatibility not required) | 1.7 | 3.6 |
| VDD_USB (USB compatibility required) | 3.1 | 3.6 |
| VDD_IO | 1.7 | 3.3 |
| VDD_IN | 3.0 | 3.3 |

## Voltage Parameters

| **Logic Levels (VUSB)** | | | |
|---|---|---|---|
| INPUT VOLTAGE LEVELS | MIN | TYP | MAX |
| $V_{ih}$ | 0.7VDD_USB | | |
| $V_{il}$ 2.7<VDD_USB<3.0 | -0.4 | | +0.8 |
| 1.7<VDD_USB<1.9 | -0.4 | | +0.4 |
| **OUTPUT VOLTAGE LEVELS (1.7<VDD_USB<1.9)** | | | |
| $V_{oh}$ (Iout = -4mA) | VDD_USB – 0.4 | | |
| $V_{ol}$ (Iout = 4mA) | | | 0.4 |
| **OUTPUT VOLTAGE LEVELS (2.7<VDD_USB<3.0)** | | | |
| $V_{oh}$ (Iout = -4mA) | VDD_USB – 0.2 | | |
| $V_{ol}$ (Iout = 4mA) | | | 0.2 |

Note: VDD_USB must be connected to power the USB and UART interfaces.

| **Logic Levels (VIO)** | | | |
|---|---|---|---|
| INPUT VOLTAGE LEVELS | MIN | TYP | MAX |
| $V_{ih}$ | 0.7VDD_IO | | |
| $V_{il}$ 2.7<VDD_IO<3.0 | -0.4 | | +0.8 |
| 1.7<VDD_IO<1.9 | -0.4 | | +0.4 |
| **OUTPUT VOLTAGE LEVELS (1.7 < VDD_IO < 1.9)** | | | |
| $V_{oh}$ (Iout = -4mA) | VDD_IO – 0.4 | | |
| $V_{ol}$ (Iout = 4mA) | | | 0.4 |
| **OUTPUT VOLTAGE LEVELS (2.7 < VDD_IO < 3.0)** | | | |
| $V_{oh}$ (Iout = -4mA) | VDD_IO – 0.2 | | |
| $V_{ol}$ (Iout = 4mA) | | | 0.2 |

Table 2-1. Voltage Parameters

## I/O Details

| Categories | Feature | Implementation |
|---|---|---|
| Command Interface | AT Instructions set | Comprehensive control of connection and module operation<br>S Registers for non-volatile storage of parameters |
| UART Interface | Baud Rate | AT Mode Default Baud Rate: 9600 bps<br>MP Mode Default Baud Rate: 115,200 bps |
| Supply Voltage | Supply | 3.0V – 3.3V DC |
| | I/O | 1.7V – 3.3V DC (independent of Supply) |
| | USB & UART | 1.7V – 3.6V DC (independent of Supply) |
| Coexistence / Compatibility | WLAN (802.11) | 2-wire and 3-wire hardware coexistence schemes supported |
| Connections | Interface | Surface Mount Pads |
| | External Antenna (BTM440/2) | Pad for 50 Ohm antenna |
| Approvals | Bluetooth | Qualified as an END product |
| | FCC | Limited Modular Approval (BTM440/2)<br>Full Modular Approval (BTM441/3) |
| | Industry Canada (IC) | Limited Modular Approval (BTM440/2)<br>Full Modular Approval (BTM441/3) |
| | CE & R&TTE | Meets CE and R&TTE requirements |
| Miscellaneous | Lead free | Lead-free and RoHS compliant |
| | Warranty | 12 Months |
| Development Tools | Development Kit | Development board and software tools<br>DVK-BTM440/2 Dev Kit with BTM440/2 module fitted<br>DVK-BTM441/3 Dev Kit with BTM441/3 module fitted |

Table 2-2. I/O Details

# Chapter 3
# AT Command Set Reference

## Introduction to AT Commands

This chapter describes the ‘AT’ protocol used to control and configure the BTM44x Bluetooth module after it has been configured to present an ‘AT’ protocol instead of the alternate multipoint packet based interface. The multipoint packet protocol is described in section 6 of this document.

The protocol is similar to the industry standard Hayes AT protocol used in telephony modems, as both types of devices are connection oriented. The AT command set has been extended to make the Laird device perform the three core actions of a Bluetooth device, which is, establish Bluetooth connections, pairing and Inquiry. Many other AT commands are also provided to perform ancillary functions, such as trusted device database management and S Register maintenance.

Just like telephony modems, the Laird device powers up in an unconnected state and will only respond via the serial interface. In this state the Laird device can respond to Bluetooth Inquiries. Then, just like controlling a modem, the host can issue AT commands which map to various Bluetooth activities. These AT commands have appropriate counterparts in the alternate multipoint packet based protocol which also achieve the same goal.

The nature of ‘AT’ protocol allows it to control and manage only one connection at a time, this is in contrast to the multipoint packet protocol which can simultaneously control many connections. The main advantage ‘AT’ protocol offers is simplicity.

The module has a serial interface, through which the ‘AT’ protocol is channeled, which can be configured for baud rates from 1200 up to 921600, and has an RF communications end point. The default baud rate for AT command mode modules is 9600bps.

The RF communications endpoint has a concept of connected and unconnected modes and the ‘AT’ protocol at the serial interface has a concept of command and data modes. This leads to the matrix of states shown below.

| | **RF Unconnected** | **RF Connected** |
|---|---|---|
| Command Mode | Allowed | Allowed |
| Data Mode | ILLEGAL | Allowed |

The combination ‘Data + RF Unconnected Mode’ does not make sense and will be ignored.

Navigation between these states is done using the AT command/responses which are described in detail in subsequent sections.

There will be many references to the term ‘S Register’ in the rest of this document. These are basically an array of integer values stored in non-volatile memory which are used to configure the module so that it behaves in a certain way after being powered. These ‘S Register’ have two attributes; a value and an ID. The ‘ID’ is a positive integer number which is used in appropriate commands to read/write the values.

# AT Protocol Mode

## AT Protocol Assumptions

The CSR (Cambridge Silicon Radio) bluetooth chipset in Laird devices has limited memory resource. Therefore it is NOT proposed that there be full implementation of the AT protocol as seen in modems. The claim made for this device is that it will have a protocol ***similar*** to an AT modem. In fact, the protocol is similar enough so that existing source code written for modems, can be used with very little modification with a Laird device.

Therefore the following assumptions are made:-

- All commands are terminated by the carriage return character 0x0D, which is represented by the string <cr> in subsequence sections and cannot be changed at runtime.
- All responses from the Laird device have carriage return and linefeed characters preceding and appending the response. These dual character sequences have the values 0x0D and 0x0A respectively and shall be represented by the string <cr,lf>.
- All Bluetooth addresses are represented by a fixed 12 digit case insensitive hexadecimal string.
- All Bluetooth Device Class codes are represented by a fixed 6 digit case insensitive hexadecimal string.
- Most new Bluetooth specific commands are identified by the string +BTx, where x is generally a mnemonic of the intended functionality.

## Protocol Activation

Depending on the variant of the module, the AT protocol will need to be activated so that on power up it presents this protocol interface instead of the alternate multipoint protocol.

The method that will always be available and work will be activation via S Register 255 in multipoint mode (and mapped to 9255 in AT mode), where setting a value of 1 selects multipoint packet protocol and a value of 2 selects AT protocol. Note, changes to this S register get stored in non-volatile memory at time of change and so does not require the AT&W command (or the equivalent in multipoint mode CMD_STORE_REG) to commit to non-volatile memory.

Optionally some firmware variants will allow a value of 0 in this S Register and in this case on power up the protocol selection will be dependent on the state of one of the GPIO pins (user settable) so that one state forces AT and the other multipoint.

## AT Commands and Responses

This section describes all available AT commands. Many commands require mandatory parameters and some take optional parameters. These parameters are integer values, strings, Bluetooth addresses or device classes. The following convention is used when describing the various AT commands and the response to a command will also be stated.

| | |
|---|---|
| **<bd_addr>** | A 12 character Bluetooth address consisting of ASCII characters '0' to '9', 'A' to 'F' and 'a' to 'f'. |
| **<devclass>** | A 6 character Bluetooth device class consisting of ASCII characters '0' to '9', 'A' to 'F' and 'a' to 'f'. |
| **N** | A positive integer value. |
| **M** | An integer value which could be positive or negative, which can be entered as a decimal value or in hexadecimal if preceded by the '$' character. E.g. the value 1234 can also be entered as $4D2 |
| **<string>** | A string delimited by double quotes. E.g. "Hello World". The " character MUST be supplied as delimiters. |
| **<uuid>** | A 4 character UUID number consisting of ASCII characters '0' to '9', 'A' to 'F' and 'a' to 'f'. |

## Enter Local Command Mode

Command: **^^^**

Response: *<cr,lf>OK<cr,lf>*

Description: When in data + connected mode, the host can force the device into a command + connected mode so that AT Commands can be issued to the device while a connection is established. The character in this escape sequence is specified in the S2 register, so can be changed. The escape sequence guard time is set at compile time to be 100 milliseconds. Please refer to the section Dropping Connections in chapter 8 for more related information.

In modems this escape sequence is typically "{delay}**+**{delay}**+**{delay}**+**{delay}".

"{delay}**^**{delay}**^**{delay}**^**{delay}" is configured by default to avoid confusion when the module is providing access to a modem.

## Command Mode Status Check

Command: **AT**

Response: *<cr,lf>OK<cr,lf>*

Description: Used to check the module is available and does not invoke any specific action.

## Accept Incoming Connection (Answer Call)

Command: **ATA**

Response: *<cr,lf>CONNECT 123456789012,<uuid>,<<cr,lf>*
Where <uuid> is the profile on which the connection has been established.

escription: Accept an incoming connection, which is indicated by the unsolicited string <cr,lf>RING 123456789012<cr,lf> where 123456789012 is the Bluetooth address of the connecting device.

## Make Outgoing Connection

Command: **ATD<bd_addr>,<uuid>**

Response: *<cr,lf>CONNECT 123456789012,<uuid>,><cr,lf>*

Or

*<cr,lf>NO CARRIER<cr,lf>*

Description: Make a connection to device with Bluetooth address <bd_addr> and profile <uuid>. The <uuid> is an optional parameter which specifies the UUID of the profile server to attach to, and if not supplied then the default UUID for SPP (1101) is used.

The UUIDs in the following table are allowed:

| Profile Name | UUID |
|---|---|
| Serial Port | 1101 |
| HID | 1124 |
| HDP | Use appropriate canned HDP commands instead |

## Enable/Disable Echo

Command: **ATEn**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command enables or disables the echo of characters to the host. The default echo condition is set via S Register 506. This command does not affect the S Register 506.

| | |
|---|---|
| 0 | Disable echo. |
| 1 | Enable echo. |

All other values of n will generate an error.

## Drop Connection

Command: **ATH**

Response: *<cr,lf>NO CARRIER<cr,lf>*

*Or*

*<cr,lf>OK<cr,lf>*

Description: Drop an existing connection or reject an incoming connection indicated by the unsolicited RING message. If a connection does not exist then the response will be OK.

## Information

Command: **ATIn**

Response: *For recognized values of n.*

*<cr,lf>As Appropriate<cr,lf>OK<cr,lf>*

*All other values of n will generate*

*<cr,lf> Laird Technologies Inc (c)2010 <cr,lf> OK<cr,lf>*

Description: This will return the information in the following table about the Laird device. The list is not exhaustive as there are some values of 'n' which generate information for use by Laird Support.

| Index | Description |
|---|---|
| 0 | The product name/variant. |
| 1 | The underlying CCL Stack version information. |
| 3 | The Laird firmware revision. |
| 333 | The full Laird firmware revision |
| 4 | A 12 digit hexadecimal number corresponding to the Bluetooth address of the Laird device. |
| 6 | The maximum size of trusted device database. |
| 9 | 0 if not in a connect state and 1 if in a connect state. |
| 11 | The reason why a "NO CARRIER" resulted in the most recent attempt at making an outgoing connection. Where the response values are as follows:<br>3 = Normal disconnection |
| 21 | Current discoverable mode:<br>0 = Not Discoverable<br>1 = Generic Discoverable mode<br>2 = Limited Discoverable mode |
| 22 | Current connectable mode:<br>0 = Not Connectable<br>1 = Connectable |
| 23 | Same as (9) above.<br>0 if not in a connect state and 1 if in a connect state. |
| 42 | Current state of the module<br>14 = Not discoverable and not connectable and not in connection<br>18 = Connected mode<br>174 = Connectable and Discoverable<br>173 = Connectable only<br>172 = Discoverable only |
| 56 | The number of devices in the trusted device database in format a,b where 'a' is the number of devices in the 'rolling' database and 'b' in the 'persistant' database. |
| 100 | Returns the hardware ID (100 for BTM4xx platform) |

## Enter Data Mode When Connected and in Command Mode

Command: **ATO**

Response: *<cr,lf>CONNECT<cr,lf>*

*or*

*<cr,lf>ERROR nn<cr,lf>*

Description: Return to data mode. Assume that the module is in data mode after OK is received. Responds with an error if there is no Bluetooth connection.

## Set S Register

Command: **ATSn=m**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: As with modems, the Laird Bluetooth module employs a concept of registers which are used to store parameters, such as escape sequence character, inquiry delay time etc.

The value part ‘m’ can be entered as decimal or hexadecimal. A hexadecimal value is specified via a ‘$’ leading character. For example $1234 is a hexadecimal number.

When S register values are changed, the changes are **not** normally stored in non-volatile memory UNTIL the AT&W command is used (unless specifically stated otherwise). Note that AT&W does not affect some S registers, for example 520 to 525 or 9240 to 9255 as they are updated in non-volatile memory when the command is processed.

## Read S Register Value In Decimal Or Hex

Command: **ATSn?<$>**

Response: *For recognised values of n:*

*<cr,lf>As Appropriate<cr,lf>**OK**<cr,lf>*

*For unrecognised values of n:*

*<cr,lf>ERROR nn<cr,lf>*

Description: This will return the current value of register n. If the optional $ character is supplied after the ? then the returned value will be in hexadecimal with a leading $. For example the value 1000 will be returned as $3E8

## Read S Register’s Valid Range

Command: **ATSn=?**

Response: For recognised values of n:

<cr,lf>nnnn..mmmm<cr,lf>OK<cr,lf>

For unrecognised values of n:

<cr,lf>ERROR nn<cr,lf>

Description: This will return the valid range of values for register n.

### Send Data To Peer When In Command Mode

Command: **ATX<string>**

Response: <cr,lf>OK<cr,lf>

Or if a connection does not exist

*<cr,lf>ERROR 56<cr,lf>*

Description: This command is used to send data to the remote device when in local command and connected mode.

If a non-printable ascii character is to be sent then insert the escape sequence \hh where hh are two hexadecimal digits. The 3 character sequence \hh will be converted into a single byte before transmission to the peer.

Note: For HID connections, the entire <string> is deemed to be a single hid report.

### Factory Default

Command: **AT&F***

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command erases all user parameters in non-volatile memory.
The new settings will become active after a reset.

### Factory Default

Command: **AT&F+**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command erases all user parameters in non-volatile memory except S Registers 520 to 525 and 9240 to 9255. This means that the trusted device database is cleared, but 'AT' protocol mode is retained and UART config (baudrate, stopbits etc) is preserved.

The new protocol and settings will become active after a reset.

### Factory Default

Command: **AT&F*AT***

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command erases all user parameters in non-volatile memory except S Register 9255. This means that the trusted device database is cleared, but 'AT' protocol mode is retained and UART parameters are reset to factory default settings.

The new protocol and settings will become active after a reset.

### Factory default to multipoint mode

Command: **AT&F*MP***

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command erases all user parameters in non-volatile memory including S Register 9255 and S Reg 9255 is set to 1 for MP mode. This means that the trusted device database is cleared, and protocol is set to MP mode and all UART parameters are reset to factory default settings.

The new protocol and settings will become active after a reset.

### Write S Registers To Non-Volatile Memory

Command: **AT&W**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: Writes current S Register values to non-volatile memory so that they are retained over a power cycle.

### Write <String> To Blob(0)

Command: **AT+BTB=<string>**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to clear blob(0) first and then <string> is appended to that blob after the string is de-escaped. This allows binary data to be loaded into the blob buffer for subsequent processing using the AT+BTBnnnn command syntax.

### Append <String> To Blob(0)

Command: **AT+BTB+<string>**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used append <string> blob(0) after the string is de-escaped. This allows binary data to be loaded into the blob buffer for subsequent processing using the AT+BTBnnnn command syntax.

## Action And Process Data In Blob(0)

Command: **AT+BTBnnnn**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to process blob(0) as per the action specified by 'nnnn'. The actions are described briefly as per the table below (more details in the MP protocol section):-

| **Index** | **Action** |
|---|---|
| 0 | Clear Blob(0) |
| 1 | Get byte count in Blob(0) |
| 2 | Destructively read Blob(0). Data is sent so that non-printable data bytes are escaped with \hh. |
| 3 | Save Blob(0) as Hid Descriptor(0) in non-volatile memory |
| 4 | Load Blob(0) as Hid Descriptor(0) from non-volatile memory |
| 5 | Save Blob(0) as Hid Service Name in non-volatile memory |
| 6 | Load Blob(0) as Hid Service name from non-volatile memory |
| 7 | Commit Blob(0) as Enhanced Inquiry Data |
| 8 | Save Blob(0) as Enhanced Inquiry Data in non-volatile memory, to be used automatically after subsequent resets |
| 9 | Load Blob(0) from the Enhanced Inquiry Data from non-volatile memory. |

## Remove Trusted Device

Command: **AT+BTD<bd_addr>**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to remove the specified device from the list of trusted devices in the non-volatile database. If the device is not in the database then the response will still be an OK. Error response is for when the address is not a 12 character hex string.

## Remove All Trusted Devices

Command: **AT+BTD***

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to remove all devices from the list of trusted devices in the non-volatile database. No confirmation will be asked for. So beware!!!

**WARNING**: If you make a connection, the link key gets cached in the underlying stack. So if you subsequently delete the key using AT+BTD* and immediately request an authenticated connection to the same device, then the connection may be established. To ensure this does not happen, send ATZ after the AT+BTD*.

### Get the remote friendly name

Command: **AT+BTF<bd_addr>**

Response: *<cr,lf>Friendly Name*

*<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command gets the remote friendly name of the address specified. If the friendly name has non printable characters (including the character “) then those characters will be escaped into a 3 character ‘\hh’ sequence.

### Enable Connectable Mode

Command: **AT+BTG**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: Enable page scanning only and wait for a connection from any device. Inquiry scans are disabled.
The page scan window and interval timing is derived from S Reg 9009 and 9010.
Use ATi21 and ATi22 to determine the discoverable and connectable modes at any time.

### Inquire

Command: **AT+BTI**

Response: *<cr,lf>12346789012*
*<cr,lf>12345678914*
*<cr,lf>OK<cr,lf>*

Description: This will make the device perform an inquiry for ‘duration’ milliseconds and ‘max’ number of unique responses, where ‘duration’ is specified by S register 517 and ‘max’ is specified by S register 518. Only the Bluetooth address of responding devices is listed.

### Inquire And Display Devclass Too

Command: **AT+BTIV**

Response: *<cr,lf>12346789012,123456*

*<cr,lf>12345678914,123456*

*<cr,lf>OK<cr,lf>*

Description: As per AT+BTI but the response includes the device class code for all inquiry responses.

## Inquire And Get Friendly Names Too

Command: **AT+BTIN**

Response: *<cr,lf>12346789012,123456,"Laird BT Module"*

*<cr,lf>12345678914,123456, “Nokia N70"*

*<cr,lf>OK<cr,lf>*

Description: As per AT+BTI but the response includes the device class code and friendly name for all inquiry responses. The friendly name strings are in UTF-8 format as per the Bluetooth specification.

## Inquire With Enhanced Inq Resp

Command: **AT+BTIE**

Response: *<cr,lf>12346789012,123456,””,-45,"\0A\08Laird FEF"*

*<cr,lf>12345678914,123456,“",-75,””*

*<cr,lf>OK<cr,lf>*

Description: As per AT+BTI but the response includes the device class code, rssi and the enhanced inquiry information. The friendly name is not acquired as it is a time expensive procedure and therefore an empty string is sent as a placeholder.

## Set Pincode Or Passcode

Command: **AT+BTK=<string>**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to provide a passkey when PIN? 12345678 or PASSKEY? 12345678 indications are received asynchronously.

The string length must be in the range 1 to 16, for PIN? otherwise an error will be returned.

The string length must be exactly 6 characters, for PASSKEY? otherwise an error will be returned and each character MUST be a decimal digit in the range 0 to 9.

If there is no ongoing pairing in progress, then the <string> will be stored in non-volatile memory and be used in subsequent legacy pairing attempts. To delete the pincode stored in non-volatile memory, submit the command with an empty string. A stored value is not used for a PASSKEY? Event.

## Reject Yes/No Simple Secure Pairing

Command: **AT+BTKN**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: When the module is configured for 'Display with Yes/No' security via S Register 9006 then this command is used to convey a 'NO' for the simple pairing procedure. This command will be sent as a result of having received a "PASSKEY? 2 <bd_addr>" asynchronous response.

## Accept Yes/No Simple Secure Pairing

Command: **AT+BTKY**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: When the module is configured for 'Display with Yes/No' security via S Register 9006 then this command is used to convey a 'YES' for the simple pairing procedure. This command will be sent as a result of having received a "PASSKEY? 2 <bd_addr>" asynchronous response.

## Set Friendly Name In Non-Vol Memory

Command: **AT+BTN=<string>**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This sets the default friendly name of this device as seen by other devices. It will be stored in non-volatile memory. Use AT+BTN? To read it back. An empty string ("") will delete the string from non-volatile memory which will force the default friendly name to be used.

## Read Friendly Name From Non-Vol Memory

Command: **AT+BTN?**

Response: *<cr,lf>My FriendlyName<cr,lf>*

*<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: Read the friendly name from non-volatile memory.

## Enable Connectable+Discoverable Mode

Command: **AT+BTP**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: Enable page and inquiry scanning and wait for a connection from any device.

The page scan window and interval timing is derived from S Reg 9009 and 9010

The inquiry scan window and interval timing is derived from S Reg 9007 and 9008

## Enable Discoverable Mode Only

Command: **AT+BTQ**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: Set discoverable mode only by enabling inquiry scanning.

The inquiry scan window and interval timing is derived from S Reg 9007 and 9008

Use ATi21 and ATi22 to determine the discoverable and connectable modes at any time

## List Trusted Device

Command: **AT+BTT?**

Response: *<cr,lf>12346789012*

*<cr,lf>12345678913*

*<cr,lf>12345678914*

*<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to list the contents of both the ‘rolling’ and the ‘persist’ trusted device database. The link key is NOT displayed so the response is as shown above. If the list is empty then just the OK response is sent otherwise an OK is used to terminate the list. Use the command ATI6 to read the maximum size of the trusted device database.

NOTE: All new successful pairings are automatically stored in the ‘rolling’ database. If the database is full, then the oldest is deleted to make room for the new one. To ensure a link key is newer deleted, transfer it to the ‘persist’ database using the command AT+BTT<bd_addr> described in detail later.

## List Trusted Device

Command: **AT+BTTn?**

Response: *<cr,lf>12346789012*

*<cr,lf>12345678913*

*<cr,lf>12345678914*

*<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to list the contents of either the ‘rolling’ or the ‘persist’ trusted device database. Where n=0 for the rolling database and 1 for the persist database. The link key is NOT displayed so the response is as shown below. If the list is empty then just the OK response is sent otherwise an OK is used to terminate the list. Use the command ATI6 to read the maximum size of the trusted device database.

## Transfer Device To ‘Persist’ List

Command: **AT+BTT<bd_addr>**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: When a successful pairing occurs, the new link key is automatically stored in the ‘rolling’ database where if the database is full, the oldest device is deleted. This poses a risk of a trusted device being automatically deleted especially when the module is in ‘just works’ simple pairing mode and so pairings will and can occur without the host being involved and so there is a definite risk of link key deletion.

This command is used to transfer a device specified via the address supplied to the ‘persist’ database so that a trusted device will never get automatically deleted.

## Initiate A Pairing

Command: **AT+BTW<bd_addr>**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This initiates pairing with a device whose Bluetooth address is <bd_addr>. An OK response is sent immediately and when the PIN or PASSCODE is required, asynchronous indications will be sent to the host in the form PIN? <bd_addr> or PASSKEY? <bd_addr> or PAIR ? <bd_addr> where the address confirms the device with which the pairing is to be performed. To supply a PIN or passcode, use the AT+BTK command and to respond with a YES or NO, use the command AT+BTKY or AT+BTKN respectively.

For a successful pairing, the link key is automatically stored in the ‘rolling’ database which can be queried using the AT+BTT0? Command.

NOTE: The “OK” response is sent immediately on receipt of the AT+BTW command. On pairing completion, an unsolicited message will be sent to the host which will be in the form PAIR n <bd_addr>, where n will be 0 for a successful pairing.

## Disable Connectable And Discoverable Mode

Command: **AT+BTX**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: Disable page/inquiry scanning. This means it will not accept incoming connections or inquiry requests. More specifically it negates the effect of AT+BTQ, AT+BTG and AT+BTP commands.

Use ATi21 and ATi22 to determine the discoverable and connectable modes at any time

## HDP: Associate The Agent With Manager

Command: **AT+HAAhhhh**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Agent related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 0.

'hhhh' is obtained as a response to the AT+HAB command

## HDP: Bind Manager to Agent

Command: **AT+HAB<bd_addr>,iiii**

Response: *<cr,lf>hhhh<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Agent related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 0.

'iiii' is the nominal code for the data specialization.

## HDP: Disassociate The Agent From Manager

Command: **AT+HADhhhh**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Agent related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 0.

## HDP: Endpoint Definition In SDP Record

Command: **AT+HAE,iiii,"endpointname"**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Agent related command. Refer to Chapter8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 0.

It is used to insert details in the sdp record.

## HDP: Read Attribute Value In Agent

Command: **AT+HAGhhhh,aaaa,ssss**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Agent related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 0.

## HDP: Activate SDP Record For Agent

Command: **AT+HAL**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Agent related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 0.

## HDP: Trigger Agent Scan Report

Command: **AT+HARhhhh,pppp[,aaaa[,aaaa[…]]]**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Agent related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 0.

## HDP: Write Attribute Value To Agent

Command: **AT+HAShhhh,aaaa,ssss,ddddd**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Agent related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 0.

## HDP: Endpoint Definition in SDP Record (Manager)

Command: **AT+HME,iiii,"endpointname"**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Manager related command. Refer to Chapter 8 Application Examples for details. Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 1.

## HDP: Endpoint Definitionin SDP Record (Manager)

Command: **AT+HME,iiii,"endpointname"**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Manager related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 1.

## HDP: Activate SDP Record For Agent (Manager)

Command: **AT+HML**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Manager related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 1.

### HDP: Read Attribute Value (Manager)

Command: **AT+HMGhhhh,oooo,aaaa**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Manager related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 1.

### HDP: Send Time To Agent (Manager)

Command: **AT+HMThhhh,ttttttt**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This is a Health Device Profile (HDP Manager related command. Refer to Chapter 8 Application Examples for details). Please note ERROR 59, implies that the profile has not been activated which means bit 2 in S Reg 9003 is not set AND S Reg 9070 is not 1.

### Add To Trusted Device Database (Rolling)

Command: **AT+KY<addr>,<link_key>**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR **nn**<cr,lf>*

Description: This command is used to add (or replace) the <addr> and<link_key> pair to the rolling trusted device database. <addr> is a 12 hex digit bluetooth address and <link_key> is a 32 hex digit random number. For more details see the multipoint command CMD_TRUSTED_DB_ADD.

### Read The Link Key For Address Specified

Command: **AT+KY<addr>?**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

*nn will be 49 if the device is not in the trusted device database*

*nn will be 45 is S Reg 47 is not set to 1*

Description: This command is used to read the link key from the trusted device database for device with address <addr>. The link key information is sent only if S Reg 47 ( 9047) is set to 1. This command is gated through S Reg 47 (9047) to get confirmation from the user that they acknowledge that security has been compromised by allowing link keys to be read.

## Unsolicited/Async Responses

The ‘AT’ Protocol is a command/response type of protocol. This means that the Laird device will normally only respond to AT commands and in addition will only respond to one AT command at a time.

Under special circumstances, unsolicited responses will be sent to the host. They are described in the following subsections. Each unsolicited response it prefixed and postfixed by a cr,lf two character sequence.

Command: **No Command. This is a status message.**

Response: *RING*

Description: This string is sent to the host every second repeatedly when a remote device is initiating a serial port connection. The fully qualified string is in the form RING 012345678901 where 012345678901 is a 12 digit hexadecimal number which corresponds to the remote device’s Bluetooth address.

The host shall respond with the ATA command to accept the connection or reject it using the ATH command.

If S Register 0 is set to a non-zero value then the incoming SPP connection will automatically be accepted after the number of RINGS specified in S Register 0 is sent to the host.

Only incoming SPP connections invoke a RING response. Connections on other profiles will automatically be accepted.

Command: **No Command. This is a status message.**

Response: *PIN ? <bd_addr>*

Description: This response is sent to the host during a legacy pairing negotiation (pre BT version 2.1 compliant devices).

The fully qualified string is PIN? 012345678901 where 012345678901 is the Bluetooth address of the peer device. In response, the host must supply a pin code which is entered using the AT+BTK command.

Command: **No Command. This is a status message.**

Response: *PASSKEY ? N <bd_addr>[,passcode]*

Description: This response is sent to the host during a simple secure pairing (SSP) negotiation and the module has been configured appropriately via S Register 9006.

Where N will be 1 for the host to display the passkey supplied, 2 for the host to respond with either the command AT+BTKY or AT+BTKN and 3 for the host to respond with AT+BTK=”nnnnnn”.

The fully qualified string is :-

PASSKEY? 1 012345678901,123456 where 012345678901 is the Bluetooth address of the peer device and 123456 is the passcode to display to the user.

PASSKEY? 2 012345678901,123456 where 012345678901 is the Bluetooth address of the peer device and 123456 is the passcode to display to the user.

PASSKEY? 3 012345678901 where 012345678901 is the Bluetooth address of the peer device and the user will echo the passcode displayed on the peer device, or agree with the other user to enter the same random 6 digit passcode at both ends.

Command: **No Command. This is a status message.**

Response: *PAIR N <bd_addr>*

Description: This response is sent to the host on completion (success or otherwise) of a pairing process. If pairing was successful then ‘n’ = 0, if a timeout occurred then ‘n’=1 and for all other unsuccessful outcomes the value will be 2.

The parameter <bd_addr> is the address of the peer device if available.

Command: **No Command. This is a status message.**

Response: *RX<string>*

Description: This response is sent to the host when the unit is in online-command mode and S Register 531 is set to 3 and data arrives from a peer. For profiles other than SPP (1101) the S Register 531 is used as a flag. If it is 0, then the profile will be serviced in ‘canned’ mode and in that case RX”” responses are not sent and neither is the ATX<string> command needed to send data.

If the data from the string contains non-printable characters (for example ASCII 0 to 31 and ASCII 128 to 255), then those characters are translated into a 3 character escape sequence starting with ‘\’. For example the embedded <cr><lf> sequence would be sent as the 6 character string \0D\0A.

If the data contains the character ‘"’ then it is sent as \22.

If the data contains the character ‘\’ then it is sent as \5C

Command: **No Command. This is a status message.**

Response: *HDA:ASSOCIATED hhhh,iiii,cccc,sssssss*

Description: This is a Health Device Profile (HDP Agent) related asynchronous response. Refer to Chapter 8 Application Examples for details.

Command: **No Command. This is a status message.**

Response: *HDA:DISASSOCIATED hhhh*

Description: This is a Health Device Profile (HDP Agent) related asynchronous response. Refer to Chapter 8 Application Examples for details.

Command: **No Command. This is a status message.**

Response: *HDA:TIME hhhh,ttttttt*

Description: This is a Health Device Profile (HDP Agent) related asynchronous response. Refer to Chapter 8 Application Examples for details.

Command: **No Command. This is a status message.**

Response: *HDM:ASSOCIATED hhhh,iiii,cccc,sssssss*

Description: This is a Health Device Profile (HDP Manager) related asynchronous response. Refer to Chapter 8 Application Examples for details.

Command: No Command. This is a status message.

Response: *HDM:DISASSOCIATED hhhh*

Description: This is a Health Device Profile (HDP Manager) related asynchronous response. Refer to Chapter 8 Application Examples for details.

Command: **No Command. This is a status message.**

Response: *HDM:ASSOCIATED hhhh,iiii,cccc,sssssss*

Description: This is a Health Device Profile (HDP Agent) related asynchronous response. Refer to Chapter 8 Application Examples for details.

Command: **No Command. This is a status message.**

Response: *HDM:SCANREPORT hhhh:pppp …..<more>…*

Description: This is a Health Device Profile (HDP Agent) related asynchronous response. Refer to Chapter.

8 Application Examples for details.

# Chapter 4
# S Registers

All S registers are accessible when operating in AT protocol mode, but in MP protocol mode only the S Registers listed as ‘Standard’ and ‘Special’ are visible.

You will note that ‘Standard’ and ‘AT’ S Registers share the same numbers in some cases. For this reason, the Standard and Special registers are accessed from AT mode by offsetting 9000. For example, the standard S register 3 for profiles, is read by using the command ATS9003? and set using ATS9003=n.

## Standard S Registers

This section details all the standard configuration ‘S’ registers. Min and Max values are given in decimal, unless the value is prefixed by 0x, in that case the value is in hexadecimal.

| RegNo Dec (Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| 3 (03) | 0 | 3 | Profiles | Server Profile record Mask<br>Bit 0 = SPP<br>Bit 1 = HID<br>Bit 2 = HDP<br>If HID is enabled then see S Reg 39 for further configuration options in terms of device or host implementation.<br>If HDP is enabled then see S Reg 70 for further configuration options in terms of Agent or Manager services.<br>Note: depending on the firmware build, some profiles are not available, in that case setting or clearing the appropriate bit in this register will have no effect as that bit is ignored. |
| 4 (04) | 0 | 1 | GAP | Default Connectable Mode on power up/reset<br>0: Disable<br>1: Enable |
| 5 (05) | 0 | 2 | GAP | Default Discoverable Mode on power up/reset<br>0 : Disable<br>1 : Enable General Discoverable mode (uses GIAC = 0x9E8B33)<br>2 : Enable Limited Discoverable mode (used LIAC = 0x9E8B00)) |
| 6 (06) | 12 | 15 | GAP Security IO Capability | Default Security Mode on power up/reset<br>12 = SSP + IO_CAP_NO_INPUT_NO_OUTPUT<br>13 = SSP + IO_CAP_DISPLAY_YES_NO<br>14 = SSP + IO_CAP_KEYBOARD_ONLY<br>15 = SSP + IO_CAP_DISPLAY_ONLY |
| 7 (07) | 12 | 2560 | GAP | Inquiry Scan Interval in units of msec |
| 8 (08) | 12 | 2560 | GAP | Inquiry Scan Window in units of msec |
| 9 (09) | 12 | 2560 | GAP | Page Scan Interval in units of msec |
| 10 (0A) | 12 | 2560 | GAP | Page Scan Window in units of msec |
| 11 (0B) | 23 | 4096 | SPP | Rfcomm frame size for all rfcomm based profiles.<br>23-512 range will have a granularity of 1 and 512 to 4096 will have a granularity of 4.<br>E.g Setting 4095 will set a value of 4092. That is, new values are rounded down.<br>Note: Testing and calculations by our stack vendor has shown that the incremental benefit above 990 is not worth the downside of handling larger payloads. |

| RegNo Dec (Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| 12 (0C) | 1 | 30 | General / GAP | Link supervision Timeout in seconds.<br>This value is only read on power up when the profiles are registers, so after changing the value, it needs to be committed to non-volatile memory using AT&W or CMD_SREG_STORE and then the module needs a power cycle. |
| 14 (0E) | 0 | 1 | SSP | Auto Accept Channel Setup. If this is 1, incoming connections will be automatically accepted.<br>If this is 1, EVT_CONNECTION_SETUP events are not sent to the host when an incoming connection arrives. |
| 32 (20) | 0 | 4 | SPP | Master/Slave role preference for SPP incoming connections.<br>0 = Don't Care<br>1 = Prefer Master<br>2 = Prefer Slave<br>3 = Must be Master<br>4 = Must be Slave |
| 33 (21) | 0 | 4 | SPP | Master/Slave role preference for SPP outgoing connections.<br>0 = Don't Care<br>1 = Prefer Master<br>2 = Prefer Slave<br>3 = Must be Master<br>4 = Must be Slave |
| 34 (22) | 0 | 7 | SPP | If Profiles Reg 3 bit 0 is set and this is not 0 then incoming SPP connections are allowed up to the number specified in this register. |
| 35 (23) | 0 | 7 | SPP | If Profiles Reg 3 bit 0 is set and this is not 0 then outgoing SPP connections are allowed up to the number specified in this register |
| 36 (24) | 0 | 1 | Profiles | Enable DeviceId Sdp record. And use the vid/pid as per registers 37 and 38 |
| 37 (25) | 0 | 0xFFFF | Profiles | USB Vendor ID to use in the DeviceId record |
| 38 (26) | 0 | 0xFFFF | Profiles | USB Product ID to use in the DeviceId record |
| 39 (27) | -2 | 2 | HID | This register is significant if HID Profile is enabled via Register 3.<br>Negative values imply that HID HOST Profile is registered.<br>Value 0 and above imply HID DEVICE Profile will be registered and<br>0 = standard KEYBOARD device (104 keys)<br>And all other positive values are associated with custom HID Device Descriptors which are preloaded using the CMD_BLOBMANAGE or AT+BTB command into non-volatile memory.<br>There is MIMIMAL validation of the HID Descriptor that a user uploads. It is extremely important that a properly constructed descriptor is uploaded for storage in nonvol memory<br>Note: If HID functionality is not included in the firmware build, then this register will not be available |
| 40 (28) | 0 | 0x1F | SPP | On SPP connection, this specifies the initial state of the |

| RegNo Dec (Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | | following modem control lines sent to the peer,<br>Bit 0 := RTR (RTS/CTS)<br>Bit 1 := RTC (DTR/DSR)<br>Bit 2 := DV (DCD)<br>Bit 3 := IC (Ring Indicate RI)<br>Bit 4 := FC (Reserved – Future use) |
| 41 (29) | 0 | 0xFF | HID | HID Device options : Bit MASK Values.<br>When the module is configured as HID Device (Sreg39>=0) and (Sreg3 has right value), then this and S Reg 42 are used to modify optional flags that are exposed in the service record that tell the host what capabilities are built into the device.<br>The capabilities are exposed as bit masks. If a bit is set in this register, then the corresponding bit in SReg42 is the value that is used for that capability.<br>The flag masks are :-<br>HID_SIF_BATTERYPOWER 0x01<br>HID_SIF_BOOTDEVICE 0x02<br>HID_SIF_NORMALLYCONNECTABLE 0x04<br>HID_SIF_RECONNECTINITIATE 0x08<br>HID_SIF_REMOTEWAKE 0x10<br>HID_SIF_SDPDISABLE 0x20<br>HID_SIF_VIRTUALCABLE 0x40<br>HID_SIF_SUPERVISIONTIMEOUT 0x80<br>For more details about what these flags do and mean, please see the HID Profile specification available on the Bluetooth Sig Website<br>Note: If HID functionality is not included in the firmware build, then this register will not be available |
| 42 (2A) | 0 | 0xFF | HID | HID Device options : Bit Values.<br>See description for SReg 41<br>Note: If HID functionality is not included in the firmware build, then this register will not be available |
| 43 (2B) | 0 | 63 | HID | Country Code exposed in HID Device Descriptor which have the following values :-<br>NotSupported 0 Netherlands_Dutch 18<br>Arabic 1 Norwegian 19<br>Belgian 2 Persian_Farsi 20<br>Canadian_Bilingual 3 Poland 21<br>Canadian_French 4 Portuguese 22<br>Czech Republic 5 Russia 23<br>Danish 6 Slovakia 24<br>Finnish 7 Spanish 25<br>French 8 Swedish 26<br>German 9 Swiss_French 27<br>Greek 10 Swiss_German 28<br>Hebrew 11 Switzerland 29<br>Hungary 12 Taiwan 30<br>International_ISO 13 Turkish_Q 31<br>Italian 14 UK 32<br>Japan_Katakana 15 US 33<br>Korean 16 Yugoslavia 34<br>Latin American 17 Turkish_F 35 |

| RegNo Dec (Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | | Note: If HID functionality is not included in the firmware build, then this register will not be available |
| 44 (2C) | 0x6161 | 0x7A7A | HID | Language Code exposed in the HID Device Descriptor which have the following values. Each value contains ascii values of two lower case characters. For example ‘en’== 0x656E |

| | | | | | |
|---|---|---|---|---|---|
| Afar | 'aa' | Interlingua | 'ia' | Quechua | 'qu' |
| Abkhazian | 'ab' | Interlingue | 'ie' | Rhaeto_Romance | 'rm' |
| Afrikaans | 'af' | Inupiak | 'ik' | Kirundi | 'rn' |
| Amharic | 'am' | Indonesian | 'in' | Romanian | 'ro' |
| Arabic | 'ar' | Icelandic | 'is' | Russian | 'ru' |
| Assamese | 'as' | Italian | 'it' | Kinyarwanda | 'rw' |
| Aymara | 'ay' | Hebrew | 'iW' | Sanskrit | 'sa' |
| Azerbaijani | 'az' | Japanese | 'ja' | Sindhi | 'sd' |
| Bashkir | 'ba' | Yiddish | 'ji' | Sangro | 'sg' |
| Byelorussian | 'be' | Javanese | 'jw' | Serbo_Croatian | 'sh' |
| Bulgarian | 'bg' | Georgian | 'ka' | Singhalese | 'si' |
| Bihari | 'bh' | Kazakh | 'kk' | Slovak | 'sk' |
| Bislama | 'bi' | Greenlandic | 'kl' | Slovenian | 'sl' |
| Bengali | 'bn' | Cambodian | 'km' | Samoan | 'sm' |
| Tibetan | 'bo' | Kannada | 'kn' | Shona | 'sn' |
| Breton | 'br' | Korean | 'ko' | Somali | 'so' |
| Catalan | 'ca' | Kashmiri | 'ks' | Albanian | 'sq' |
| Corsican | 'co' | Kurdish | 'ku' | Serbian | 'sr' |
| Czech | 'cs' | Kirghiz | 'ky' | Siswati | 'ss' |
| Welsh | 'cy' | Latin | 'la' | Sesotho | 'st' |
| Danish | 'da' | Lingala | 'ln' | Sudanese | 'su' |
| German | 'de' | Laothian | 'lo' | Swedish | 'sv' |
| Bhutani | 'dz' | Lithuanian | 'lt' | Swahili | 'sw' |
| Greek | 'el' | Latvian | 'lv' | Tamil | 'ta' |
| English | 'en' | Malagasy | 'mg' | Tegulu | 'te' |
| Esperanto | 'eo' | Maori | 'mi' | Tajik | 'tg' |
| Spanish | 'es' | Macedonian | 'mk' | Thai | 'th' |
| Estonian | 'et' | Malayalam | 'ml' | Tigrinya | 'ti' |
| Basque | 'eu' | Mongolian | 'mn' | Turkmen | 'tk' |
| Persian | 'fa' | Moldavian | 'mo' | Tagalog | 'tl' |
| Finnish | 'fi' | Marathi | 'mr' | Setswana | 'tn' |
| Fiji | 'fj' | Malay | 'ms' | Tonga | 'to' |
| Faeroese | 'fo' | Maltese | 'mt' | Turkish | 'tr' |
| French | 'fr' | Burmese | 'my' | Tsonga | 'ts' |
| Frisian | 'fy' | Nauru | 'na' | Tatar | 'tt' |
| Irish | 'ga' | Nepali | 'ne' | Twi | 'tw' |
| Gaelic | 'gd' | Dutch | 'nl' | Ukrainian | 'uk' |
| Galician | 'gl' | Norwegian | 'no' | Urdu | 'ur' |
| Guarani | 'gn' | Occitan | 'oc' | Uzbek | 'uz' |
| Gujarati | 'gu' | Oromo | 'om' | Vietnamese | 'vi' |
| Hausa | 'ha' | Oriya | 'or' | Volapuk | 'vo' |
| Hindi | 'hi' | Punjabi | 'pa' | Wolof | 'wo' |
| Croatian | 'hr' | Polish | 'pl' | Xhosa | 'xh' |
| Hungarian | 'hu' | Pashto | 'ps' | Yoruba | 'yo' |
| Armenian | 'hy' | Portuguese | 'pt' | Chinese | 'zh' |
| Zulu | 'zu' | | | | |

| RegNo Dec (Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | | Note: If HID functionality is not included in the firmware build, then this register will not be available |
| 45 (2D) | 0 | 0x3FF | HID | Primary Language exposed in the HID Device Descriptor and the value is specified as follows ( where the values are in hexadecimal), refer to the HID specification for a complete list:-<br><br>NEUTRAL 00 , RUSSIAN 19 , FAEROESE 38<br>ARABIC 01 , SERBIAN 1a , HINDI 39<br>BULGARIAN 02 , CROATIAN 1a , MALAY 3e<br>CATALAN 03 , SLOVAK 1b , KAZAK 3f<br>CHINESE 04 , ALBANIAN 1c , SWAHILI 41<br>CZECH 05 , SWEDISH 1d , UZBEK 43<br>DANISH 06 , THAI 1e , TATAR 44<br>GERMAN 07 , TURKISH 1f , BENGALI 45<br>GREEK 08 , URDU 20 , PUNJABI 46<br>ENGLISH 09 , INDONESIAN 21 , GUJARATI 47<br>SPANISH 0a , UKRAINIAN 22 , ORIYA 48<br>FINNISH 0b , BELARUSIAN 23 , TAMIL 49<br>FRENCH 0c , SLOVENIAN 24 , TELUGU 4a<br>HEBREW 0d , ESTONIAN 25 , KANNADA 4b<br>HUNGARIAN 0e , LATVIAN 26 , MALAYALAM 4c<br>ICELANDIC 0f , LITHUANIAN 27 , ASSAMESE 4d<br>ITALIAN 10 , FARSI 29 , MARATHI 4e<br>JAPANESE 11 , ARMENIAN 2b , SANSKRIT 4f<br>KOREAN 12 , AZERI 2c , KONKANI 57<br>DUTCH 13 , BASQUE 2d , MANIPURI 58<br>NORWEGIAN 14 , MACEDONIAN 2f , SINDHI 59<br>POLISH 15 , VIETNAMESE 2a , KASHMIRI 60<br>PORTUGUESE 16 , AFRIKAANS 36 , NEPALI 61<br>ROMANIAN 18 , GEORGIAN 37<br><br>Note: If HID functionality is not included in the firmware build, then this register will not be available |
| 46 (2E) | 0 | 0x3F | HID | Sub Language exposed in the HID Device Descriptor and the value is specified as follows for english ( where the values are in hexadecimal), refer to the HID specification for a complete list:-<br><br>ENGLISH_US 01<br>ENGLISH_UK 02<br>ENGLISH_AUSTRALIAN 03<br>ENGLISH_CANADIAN 04<br>ENGLISH_NEWZEALAND 05<br>ENGLISH_IRELAND 06<br>ENGLISH_SOUTHAFRICA 07<br>ENGLISH_JAMAICA 08<br>ENGLISH_CARIBBEAN 09<br>ENGLISH_BELIZE 0A<br>ENGLISH_TRINIDAD 0B<br>ENGLISH_ZIMBABWE 0C<br>ENGLISH_PHILIPPINES 0D |

| RegNo<br>Dec<br>(Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | | Note: If HID functionality is not included in the firmware build, then this register will not be available |
| 47 | 0 | 1 | Pairing | When this register is 1, in MP Mode it enables the EVT_LINK_KEY_EX event to be sent to the host when the module receives the CMD_TRUSTED_DB_IS_TRUSTED command and in AT mode it enables the AT+KY? command so that the link key information is sent to the host in the response.<br><br>By default this key is 0.<br>By setting this to 1, the customer acknowledges that security can be compromised. |
| 50..65<br>(32..41) | 0 | 0 or 15 | GPIO | This specifies the functionality to be attached to GPIO 0 to 15 appropriately.<br><br>50 for GPIO_0 onwards to 65 for GPIO_15.<br><br>Depending on the platform, if the GPIO does not physically exist then the max value will be 0.<br><br>The functionality is specified as follows:-<br><br>0 : None (the pio hardware will not be touched)<br>1 : Input<br>2 : Output (0 on reset)<br>3 : Output (1 on reset)<br>4 : Input Inverted<br>5 : Output Inverted (0 on reset)<br>6 : Output Inverted (1 on reset)<br>7 : UART RI Input (DTE) (Ring Indicate)<br>8 : UART DCD Input (DTE)<br>9 : UART DSR Input<br>10 : UART RI Output (DCE)<br>11 : UART DCD Output (DCE)<br>12 : UART DTR Output<br>13 : UART TX Buffer NOT Empty Output<br>14 : Input pin – Select Protocol 0=AT,1=MP<br>15 : Output pin – 0 for AT, 1 for MP<br><br>For all UART input/outputs (except 13,14 & 15) there is an implied inversion. Which means to assert the output a logical 1 is applied, but by the time it hits the output pin or after it is read, there is automatic inversion.<br><br>When writing to this location(s) validation will prevent a write if the new value is a UART functionality and the same value already exists in another register. Therefore the best approach to writing to this group of registers is to first write 0 to all 16 registers and then write new values to all.<br><br>New values for these registers only come into effect after a reset/power cycle. |

| RegNo<br>Dec<br>(Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | | For details of functionality 13, see section "Uart Host Power Saving Facility" which describes how this output can be used to wake a UART host<br><br>Contact the manufacturer for specific mapping and functionality allocations. |
| 70 (46) | 0 | 1 | HDP | If set to 0 then an HDP AGENT profile will be implemented, otherwise with 1, a HDP MANAGER will be implemented.<br>In addition, bit 2 of Profiles S Reg3 has to be set.<br><br>Note: If HDP functionality is not included in the firmware build, then this register will not be available |
| 71 (47) | 0 | 65000 | HDP | HDP AGENT profile related.<br>This is the default time in milliseconds an HDP Agent will remain in associated state while there is no activity with the HDP Manger.<br>If the value is 0, that is taken as infinite time.<br>This time is referenced whenever a new hdp agent ieee specialization binding with a hdp manager Bluetooth is performed. In AT mode, see command AT+HAB and in MP mode see CMD_HDP_BIND<br><br>Note: If HDP functionality is not included in the firmware build, then this register will not be available |
| 72 (48) | 48 | 1024 | HDP | HDP Profile related.<br>This is the max tx pdu size and is computed to the nearest multiple of 16.<br><br>Note: If HDP functionality is not included in the firmware build, then this register will not be available |
| 73 (49) | 0 | 20000 | Sniff Mode / Power Saving | Sniff Attempt Time in units of milliseconds.<br>0 means disable.<br>This value must be less than half the Sniff Minimum Interval (SReg 75).<br>The value is stored in an 8 bit number and so a non-linear algorithm is used to convert between the value to the 8 bit number. This means, writing and reading may yield differing values.<br>See section "Sniff Mode Explained" |
| 74 (4A) | 1 | 40000 | Sniff Mode / Power Saving | Sniff timeout Time in units of milliseconds. 0 means disable.<br>The value is stored in an 8 bit number and so a non-linear algorithm is used to convert between the value to the 8 bit number. This means, writing and reading may yield differing values.<br>See section "Sniff Mode Explained" |
| 75 (4B) | 1 | 40000 | Sniff Mode / Power Saving | Sniff Minimum Interval in units of milliseconds. The value is stored in an 8 bit number and so a non-linear algorithm is used to convert between the value to the 8 bit number. This means, writing and reading may yield differing values.<br>See section "Sniff Mode Explained" |
| 76 (4C) | 1 | 40000 | Sniff | Sniff Maximum Interval in units of milliseconds. The value is |

| RegNo Dec (Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | Mode / Power Saving | stored in an 8 bit number and so a non-linear algorithm is used to convert between the value to the 8 bit number. This means, writing and reading may yield differing values.<br>See section "Sniff Mode Explained" |
| 80 (50) | 6500 | 70000 | UART Comms | UART latency time in microseconds.<br>In a connection, the data pump waits for X bytes (specified by SReg 11) before transferring the data to the air-side. In the event that the host only sends less than X bytes this has the potential of those bytes never being transmitted. Therefore, to cater for that scenario a timer is used to detect how long they have sat in the buffer and if too long, send those bytes air-side anyway. How long to wait is specified by this S register. The timer is started once when a byte arrives in an empty buffer– not restarted everytime a byte arrives in the buffer. |
| 81 | 10 | 80 | UART Comms | Memory usage in percent for MP mode UART rx processing. Leave to default value. Only change on advice from manufacturer.<br>New values are rounded up to the nearest 10.<br>This register controls how many memory blocks are reserved when the module receives a flood of UART data packets with small payloads. |
| 128 (80) | 0 | 0xFFFFFF | GAP | Module's Class of Device.<br>If Profiles SReg3==2 (that is only HID Profile) and SReg39 is set for 0 (ie built in keyboard hid descriptor), then this class of device is overridden with a value which specifies a hid keyboard device. |

Notes: Most registers are read by the firmware at reset. Hence after setting a register a reset is required for it to be effective. This means the relevant S Register set MUST be committed to non-volatile memory before initiating a reset. The S Registers are stored to non-volatile memory using the command [CMD_STORE_SREG].

## Special S Registers (240 to 255)

Registers 240 to 255 inclusive are marked as special in the sense that when written the value is automatically committed to non-volatile memory

| RegNo Dec (Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| 240 (F0) | 0 | 921600 | UART Comms | UART Baudrate<br><br>Writing 0 implies default baudrate which may be different for all the different protocols as follows:-<br>MP mode = 115200<br>AT mode = 9600<br><br>Automatically saved to non-volatile memory on write |
| 241 (F1) | 1 | 1 | UART Comms | UART Handshaking.<br>1=CTS/RTS |

| RegNo Dec (Hex) | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | | Automatically saved to non- volatile memory on write |
| 242 (F2) | 1 | 2 | UART Comms | UART Stopbits<br>Automatically saved to non- volatile memory on write |
| 243 (F3) | 0 | 2 | UART Comms | UART Parity<br>0=None<br>1=Odd<br>2=Even<br><br>Automatically saved to non- volatile memory on write |
| 255 (FF) | 0 | 2 | Protocol Mode | Host Communications Protocol<br>0 = Select protocol based on the state of GPIO input pin.<br>1 = Multipoint Packet Protocol<br>2 = AT Protocol<br>If this register contains 0, then at least one S register in the range 50 to 65 should be set to the value (14 == PROTOCOL_MODE_IN) so that the GPIO pin corresponding to that S Register is automatically configured as an input and if on power up, the state of that pin is 0, then AT protocol will be activated, oitherwise MP.<br>If no GPIO pin is configured to specify protocol and the value in this register is 0, then MP Protocol is selected.<br>Automatically saved to non-volatile memory on write |

All these S Registers, unless specifically mentioned become effective after a power cycle or reset.

## ‘AT’ S Registers

These registers are specific to AT protocol operation **only** and hence are not accessible from MP protocol mode. Any S Register marked as ‘remapped’ will be detailed at the place where the mapped-to S Register is described.

| RegNo | Min | Max | Category | Description |
|---|---|---|---|---|
| 0 | 0 | 15 | SPP | Rings before auto answering an incoming SPP connection. Setting 0 means a connection will not be auto answered. |
| 2 | 0x21 | 0x7E | SPP | Escape character to be used to change from data mode to command parsing mode when in a data connection. Three of these character enveloped by delays will result in a transition from data into command mode |
| 506 | 0 | 1 | General | If this is set to 1, then AT commands are echoed back to the host. This becomes effective after a power cycle. To make immediate effect on echoes, use the command ATE0 or ATE1.<br>Note: This register only specifies the initial state of echoes on power up. |
| 507 | 0 | 1 | SPP | When set to 1, which is the default, the DSR modem status input line is used to enter command mode and/or drop a connection.<br>When set to 0 the DSR line is not checked for connection related functionality. This is so that the input can be used to convey a digital status to the peer using |

| RegNo | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | | S Registers 651/654 and 661/664 inclusive. |
| 508 | Remapped | | GAP | To S Reg 9009 (9 in MP Mode see section "Standard S registers" ) |
| 509 | Remapped | | GAP | To S Reg 9010 (10 in MP Mode see section "Standard S registers" ) |
| 510 | Remapped | | GAP | To S Reg 9007 (7 in MP Mode see section "Standard S registers" ) |
| 511 | Remapped | | GAP | To S Reg 9009 (8 in MP Mode see section "Standard S registers" ) |
| 514 | 2 | 60 | Pairing | Minimum time in seconds to wait for a pairing operation to conclude. |
| 517 | 2 | 60 | Inquiry | Maximum time in seconds to perform an inquiry |
| 518 | 1 | 255 | Inquiry | Maximum number of inquiry responses in an inquiry |
| 520 | Remapped | | UART Comms | To S Reg 9240 (240 in MP Mode see section "Special S registers" ) |
| 521 | Remapped | | UART Comms | To S Reg 9240 (240 in MP Mode see section "Special S registers" ) |
| 522 | Remapped | | UART Comms | To S Reg 9241 (241 in MP Mode see section "Special S registers" ) |
| 523 | Remapped | | UART Comms | To S Reg 9242 (242 in MP Mode see section "Special S registers" ) |
| 524 | Remapped | | UART Comms | To S Reg 9243 (243 in MP Mode see section "Special S registers" ) |
| 530 | 1 | 60 | SPP | Reconnect delay when configured as master in auto connection mode. |
| 531 | 0 | 3 | SPP/ AT Parser | Specifies the AT Parser mode on connection establishment.<br>0 = Normal, that data is exchanged between UART and RF<br>1 = LOCAL_COMMAND. UART input is parsed by the AT interpreter and RF data is discarded<br>2 = REMOTE_COMMAND. RF input is parsed by the AT interpreter and UART data is discarded. If S Reg 536 is not 1 then this register cannot be set to 2 and an ERROR will be returned<br>3=LOCAL_COMMAND. UART input is parsed by the AT interpreter and incoming RF data is sent to the host using the **RX<string>** asynchronous response. |
| 561 | Remapped | | Sniff Mode | To S Reg 9073 (73 in MP Mode see section "Standard S registers" ) |
| 562 | Remapped | | Sniff Mode | To S Reg 9074 (74 in MP Mode see section "Standard S registers" ) |
| 563 | Remapped | | Sniff Mode | To S Reg 9075 (75 in MP Mode see section "Standard S registers" ) |
| 564 | Remapped | | Sniff Mode | To S Reg 9076 (76 in MP Mode see section "Standard S registers" ) |
| 619 | 0 | 0xFFFF | GPIO | This specifies a write mask when ATS620=X is used to |

| RegNo | Min | Max | Category | Description |
|---|---|---|---|---|
| | | | | set multiple GPIO states |
| 620 | 0 | 0xFFFF | GPIO | ATS620? will read the all GPIO states in one go.<br>ATS620=X will write new GPIO states using the mask in S Register 619 |
| 651 | -1 | 8 | GPIO | -1 is taken to mean no GPIO pin allocation<br>See note (1 & 3) below.<br>Requires firmware build 185 or newer |
| 652 | -1 | 8 | GPIO | -1 is taken to mean no GPIO pin allocation<br>See note (1 & 3) below<br>Requires firmware build 185 or newer |
| 653 | -1 | 8 | GPIO | -1 is taken to mean no GPIO pin allocation<br>See note (1 & 3) below<br>Requires firmware build 185 or newer |
| 654 | -1 | 8 | GPIO | -1 is taken to mean no GPIO pin allocation<br>See note (1 & 3) below<br>Requires firmware build 185 or newer |
| 661 | -1 | 8 | GPIO | -1 is taken to mean no GPIO pin allocation<br>See note (2 & 3) below<br>Requires firmware build 185 or newer |
| 662 | -1 | 8 | GPIO | -1 is taken to mean no GPIO pin allocation<br>See note (2 & 3) below<br>Requires firmware build 185 or newer |
| 663 | -1 | 8 | GPIO | -1 is taken to mean no GPIO pin allocation<br>See note (2 & 3) below<br>Requires firmware build 185 or newer |
| 664 | -1 | 8 | GPIO | -1 is taken to mean no GPIO pin allocation<br>See note (2 & 3) below<br>Requires firmware build 185 or newer |
| 717 | 2 | 15 | GAP | Maximum time to wait for a remote friendly name to be read (AT+BTI command) |

All these S Registers, unless specifically mentioned become effective after a power cycle or reset.

Notes:
S Reg 651 to 654 take a GPIO pin number in the range 1 to 8 inclusive (or as per the full range for the specific module) which is used to map from the RTR, RTC,DV,IC bits in the Rfcomm Modem Control signal which is exchanged for serial port profile. When a fresh Modem Control Sig message arrives from the peer, if the corresponding S register is NOT -1 and the specific GPIO pin has been configured as an output (using S Reg 50 to 65 inclusive) then the state of the bit is output to that pin.

S Reg 661 to 664 take a GPIO pin number in the range 1 to 8 inclusive (or as per the full range for the specific module) which is used to map to the RTR, RTC,DV,IC bits in the Rfcomm Modem Control signal which is exchanged for serial port profile. If a GPIO pin is specified in one of these S Registers and it changes state AND there is an SPP connection, then the state of that input pin is copied into a rfcomm modem control message and sent to the peer.

This capability enables the state of between 2 and 4 (depends on the direction of the connection) digital pins to be exchanged between peers without any host intervention.

# Chapter 5
# Error Codes

## Error Responses

All error responses from the device will be in the form <cr,lf>**ERROR nn**<cr,lf>, where nn will be a number in the range 00 to 99 as follows:

| Error | Description |
|---|---|
| 01 | Register not recognized |
| 02 | Value for register is out of range |
| 03 | Incoming call NOT pending |
| 04 | No call to connect to. This error code has meaning for ATO only |
| 05 | Syntax Error |
| 06 | Empty String |
| 06 | Device Class could not be stored |
| 08 | Invalid Device Class Code |
| 09 | Invalid Bluetooth Address |
| 10 | Could not set Service or Friendly name |
| 11 | PS Store Write |
| 12 | PS Store Read |
| 13 | Not Idle |
| 14 | Incorrect Mode |
| 15 | Already Scanning |
| 16 | Pairing is already in progress |
| 17 | Not USED |
| 18 | Not USED |
| 19 | Not USED |
| 20 | Not safe to write to Non-volatile Store - Ongoing Bluetooth Connection |
| 21 | Link Key Cache is Empty |
| 22 | Link Key Database is Full |
| 23 | Malloc returned NULL - Resource Issue |
| 24 | Remote Address same as Local Address |
| 25 | Connection Setup Fail, DSR Not asserted |
| 26 | Unauthenticated licence |
| 27 | Max Responses (See S Register 518) too high. Memory allocation error |
| 28 | The length of Pin in AT+BTK is too long |
| 29 | Invalid Ring count specified for S Register 0 or 100. If S0<>0 and S100<>0 then S0 must |
| 30 | ADC Error |
| 31 | Analogue Value cannot be read as it is set for output |
| 32 | Analogue Value cannot be written as it is set for input |
| 33 | S Register Value is invalid |
| 34 | Both L and R modifier cannot be specified in ATD command |
| 35 | Invalid Major Device Class – valid value in range 0x00 to 0x1F inclusive |
| 36 | Pairing in progress – Command cannot be actioned – try again later |
| 37 | Invalid Sniff parameter specified. |
| 38 | Get Remote Friendly name Failed |
| 39 | Failed to change mode to Multipoint |
| 40 | 7 Bit mode requires parity to be even or odd |
| 41 | Stream Error |
| 42 | Stream Pending |

| Error | Description |
|---|---|
| 43 | Unknown AG command |
| 44 | Busy Try Later |
| 45 | Not Allowed |
| 46 | Invalid String |
| 47 | Generic Error |
| 48 | Inquiry in progress |
| 49 | Link Key Missing |
| 50 | String De-escape Error |
| 51 | Invalid Passcode (must be 6 decimal digits) |
| 52 | Invalid Pincode |
| 53 | Invalid UUID (must be 4 hex digits) |
| 54 | Connection in progress |
| 55 | Profile unsupported |
| 56 | No SPP Connection |
| 57 | I/O Mask is Zero (see SReg 619) |
| 58 | Invalid Friendly Name |
| 59 | Profile is not active |
| 60 | HDP : Invalid Handle |
| 61 | HDP : Unknown IEEE Nominal Code (Data Specialisation) |
| 62 | HDP : Report Error |
| 63 | HDP : Invalid IEEE Code |
| 64 | HDP : Invalid Parameter |
| 65 | HDP : Attribute not found |
| 66 | HDP : Invalid Number of Arguments |
| 67 | HDP : Object Closed |
| 68 | HDP : Association Failed |
| 69 | HDP : Too many Agents |
| 70 | HDP : Object Incomplete |
| 71 | HDP : PHDC Failed |
| 72 | HDP : PHDC Insufficient Resource |
| 73 | HDP : PHDC Invalid Parameter |
| 74 | HDP : PHDC Invalid State |
| 75 | HDP : PHDC Unknown |
| | |
| 99 | Functionality yet to be coded (please report to manufacturer) |

Table 5.3: BTM Error Responses in AT Mode

# Chapter 6
# Multipoint Protocol

## Introduction to MultiPoint Protocol

This chapter describes a packet based messaging interface which is used by a host to send commands, receive responses, receive asynchronous events and exchange multiplexed data with the Bluetooth Serial Module, henceforth described as the Module.

The Module consists of a bluetooth chipset with an approved bluetooth stack which allows simultaneous connections to a minimum of 3 slaves, and depending on hardware build and conditions, up to 7 slaves. It also allows connections to multiple profiles to one or more slaves. Hence this document adopts a concept of channels instead of slave connections.

Another way of viewing these channels are as logical data pipes.

The term ‘host’ in this document is taken to mean any entity which is a source of command messages, sink for response/event messages and both source and sink for multiplexed data packets.

To further eliminate any confusion, when the terms ‘command message’ and ‘confirm message’ are used, it implies a message from the host to the module. Likewise the terms ‘response message’ and ‘event message’ are used to imply a message from the module to the host.

There is an implied client/server model in the protocol described in this document and the host shall ensure that no new commands are issue to the module until after a response is received for that command or an appropriate timeout. Confirm messages do not have the same restrictions. Data packets do not follow that model. It is likely that asynchronous event messages will be sent before response messages but that should not be taken as a signal to issue new commands. The only exception to this is when an unknown command is received. In this case the transaction is terminated by an UNKNOWN_COMMAND event.

This document does NOT describe how the packets are physically exchanged between the host and the module. The transport medium could be either UART or USB. It also does NOT describe the format of any envelope that may be required to reliably and quickly transfer the message packet between the host and module.

This implies that when the packets proposed in this document are processed, they are assumed not to contain any errors by either peer entities.

### Flow control & Data Integrity

It must be recognized that the transport mechanism will be streaming in nature. If the transport medium is USB then flow control and data integrity is inherently provided by the USB protocol.

If UART is the medium, then it shall be assumed that there will be a minimum of a 5 wire interface, RX,TX,CTS,RTS and GND. Any host attached to the UART of the module shall **strictly** observe CTS/RTS hardware handshaking. Packet data integrity may or may not be provided depending on the build. It is expected that for a UART transport media, guaranteeing data integrity will be at the **severe** expense of data throughput.

### Packet Format

This section describes the general format of incoming and outgoing packets.

The term ‘incoming’ will henceforth imply packets sent by the host to the module and ‘outgoing’ in the reverse direction. That is, the direction terminology is module (server) centric.

All packets have octet granularity. When an octet is described as containing bit fields, it shall be taken that bit 0 is the least significant bit and bit 7 is the most significant bit.

Subfields in the packet which require multiple octets, shall be ordered so that the lowest significant octet is transmitted LAST over the transport media, unless specifically described otherwise – this is also referred to as Big Endien format. For example, a 16 bit word value will require 2 octets within the packet and the first transmitted octet will correspond to the upper byte. Similarly, a 6 byte Bluetooth address shall be transported most significant byte first. If the order is reversed then it will be specifically highlighted in the description of appropriate packets.

Subfields which are data arrays shall be described with the '[ ]' operator in descriptions which come in subsequent chapters.

Apart from data packets, all command, confirm, respond and event packets are of fixed size. If there isn't enough data to fill a packet, then the packet is filled with 0s. The protocol and fixed packet format is optimized to ensure maximum data throughput over the air. Subsequent sections describe the packets in detail.

## Host to Module Packets

These are packets used to convey commands and confirms to the module or raw data to be sent over an open Bluetooth connection.

## Command & Confirm Packets

The format for command and confirm packets is as per the table below.

| Octet | Field | Description |
|---|---|---|
| 0 | LENGTH | Total length of this packet, including this octet |
| 1 | CHANNEL | Always 0 |
| 2 | CMD_ID/CNF_ID | Described in the subsequent chapters and have CMD_ or CNF_ prefixes |
| 3 | FLOW_IN | Bit 0 to 6 specify a mask. A clear bit means the module should NOT send any more packets to that corresponding spp data channel.<br>Bit 7 is always 0 and will be used as an extension bit in the future.<br>It is assumed that the host will always be able to receive a response or status packet. |
| 4..N | DATA[] | Data as required and has meaning specific to CMD_ID or CNF_ID. For example, if the command is to make a connection to a peer device, then it will be at least a 6 octet array specifying the Bluetooth address of the peer. |

The value of CMD_ID shall be in the range 0 to 63 and commands are queued until a previous command has been completed by sending a response packet to the host.

The value of CNF_ID shall be in the range 64 to 127 inclusive.

Unknown command values result in an EVT_UNKNOWN_COMMAND event, with the command value reflected in the data field. If the octet value is specified in the range 128 to 255 (0x80 to 0xFF), then reflecting that value in the data field of an EVT_UNKNOWN_COMMAND instead of the COMMAND field of a response packet guarantees that the packet will NOT be mistakenly processed as an event.

Confirm packets are not queued by the UART packet processor in the module and are processed as soon they are received. This allows, for example, passkeys and pincodes to be submitted to the module while a response is being awaited to the CMD_CONNECTION_MAKE command. In other words confirm packets always got to the head of the packet queue.

## Data Packets

The format for data packets is as per the table below and can arrive at any time, that is, they do not adhere to a client/server model. The only method by which the host can be stopped from sending this message is by sending a 0 value in the FLOW_OUT field of a response or status message. The module will be prepared to receive at least one data packet after deasserting the appropriate flow control bit.

| Octet | Field | Description |
|---|---|---|
| 0 | LENGTH | Total length of this packet, including this octet |
| 1 | CHANNEL | 0 is an invalid value as this marks the packet as command/response or event.<br>1 to 7 are dedicated serial port profile connections<br>128 is dedicated as Hid Device data channel<br>All other channels are reserved for future use. |
| 2..N | DATA[] | For channels 1 to 7, this data array is unconditionally sent over the air in the appropriate.<br>For channel 128 data will be interpreted before transmission of appropriate HID reports. |

## Packet Processing Logic

Data and Confirm packets are processed as soon as they are received.

A command packet is processed in a transaction. Which means it is processed as soon as it is received **if and only if** there is no previous command being processed and waiting for completion. Completion happens when an appropriate response packet sent to the host. If a command transaction is currently in progress, then the packet will be inserted in a first-in, first-out queue. When an on-going command transaction is complete, the queue will be inspected and if non-empty then the oldest queued command is processed.

## Module to Host Packets

These are packets used to convey responses or events from the module and raw data received over an open Bluetooth connection or internal data source. Response packets **shall** always be as a result of a command packet and event packets are **asynchronously** sent to the host as and when required. *The host shall ensure that it is always ready to accept response and event packets, especially event packets as they can be sent at any time even where there is an incomplete transaction in progress.*

## Response Packets

The format for response packets is as per the table below.

| Octet | Field | Description |
|---|---|---|
| 0 | LENGTH | Total length of this packet, including this octet |
| 1 | CHANNEL | Always 0 |
| 2 | CMD_ID | Echoed from the command packet (Shall be > 0 and < 128) |
| 3 | FLOW_OUT | Bit 0 to 6 specify a mask. A clear bit means the host should NOT send any more packets to that corresponding data channel.<br>Bit 7 is always 0 and will be used as an extension bit in the future. |
| 4 | STATUS | 0 means success, otherwise see section “STATUS values” |
| N..M | DATA[] | Data as required and has meaning specific to the response for CMD_ID |

## Event Packets

The format for status packets is as per the table below.

| Octet | Field | Description |
|---|---|---|
| 0 | LENGTH | Total length of this packet, including this octet |
| 1 | CHANNEL | Always 0 |
| 2 | EVT_ID | Described in subsequent chapters, but bit 7 is always set,<br>hence >= 128 |
| 3 | FLOW_OUT | Bit 0 to 6 specify a mask. A clear bit means the host should NOT send any more packets to that corresponding data channel.<br>Bit 7 is always 0 and will be used as an extension bit in the future. |
| N..M | DATA[] | Data as required and has meaning specific to the response for EVT_ID |

The only difference between a response and an event packet is that, octet 2 is defined as CMD_ID and in the former and EVT_ID in the latter and in addition the STATUS field is missing in the event packet.

The value of CMD_ID shall be in the range 0 to 0x3F and EVT_ID shall take values in the range 0x80 to 0xFF. This allows bit 7 of that octet to be used to decode whether the packet is a response packet or an event packet.

The value of STATUS is in the range 0 to 255. A value of 0 means SUCCESS and any other value is a failure, where the value gives more details of the failure type. The values of STATUS are defined in a ‘C’ header file which can be obtained on request from Laird.

## Data Packets

The format for data packets is as per the table below. The only method by which the host can stop the module from sending this message is by sending a 0 value in the FLOW_IN field of command message and even that is only for channels 1 to 7 inclusive.

| Octet | Field | Description |
|---|---|---|
| 0 | LENGTH | Total length of this packet, including this octet |
| 1 | CHANNEL | 0 is an invalid value as this marks the packet as command/response or event.<br>The channel number is allocated as follows:-<br>1 to 7 are dedicated serial port profile connections.<br>0x20, 0x80, 0x90..0x97,0xA0 are dedicated as Hid Device data channels.<br>0x98..0x9F are dedicated for BLOB sending and receiving data from BLOBs.<br>0xB0 is used for conveying HDP data and 0xB1 is a HDP continuation data channel.<br>0xF0 is used to convey Enhanced Inquiry Response Data to the host.<br>All other channels are reserved for future use. |
| 2..N | DATA[] | Data to be processed for channel CHANNEL |

Data packets are symmetrical in format in both directions.

Note that only data channels 1 to 7 inclusive have flow control via the FLOW_IN and FLOW_OUT fields of command /confirm and response/event packets.

## Data Channel Numbers

The following table summarizes channel ID allocation for various connections and profiles.

| Channel Number | Profile | Comments |
|---|---|---|
| 0x0 | - | All traffic routed to/from the protocol parser – hence this is a command, confirm or response packet |
| 0x1<br>..<br>0x7 | SPP | Serial Port profile data channels<br>See Note 1 below |
| 0x20 | HID<br>DEVICE | Hid Device Channel – only one device allowed at a time<br>See Note 2 below |
| 0xA0 | HID<br>DEVICE | Hid Device Channel – only one device allowed at a time<br>See Note 3 below |
| 0x90<br>..<br>0x97 | HID<br>HOST | Hid Host Channels – multiple connections to devices is possible<br>See Note 4 below |
| 0x98<br>..<br>0x9F | BLOB<br>MANAGER | Blob Manager Channels<br>See Note 5 below |
| 0xA0 | HID<br>DEVICE | Hid Device Channel for sending and receiving raw hid reports. |
| 0xB0<br>..<br>0xB1 | HDP<br>DATA | HDP Data Channels<br>See section 0 for further details of the logical channel conveyed in these data channel |
| 0xF0 | Enhance Inq<br>Response | Enhanced Inquiry Response Data |

Note 2:
This channel is used for 'canned' Hid Keyboard Device reports. Data in this channel is interpreted as ascii and for each ascii character two INPUT reports are sent to the host – the first being the press event and the second the unpress event. If the ascii value is in the range 0x7F to 0xFF then it will be silently discarded.

Note 3:
This channel is used for raw Hid Device INPUT reports. If for example the built in keyboard hid descriptor is active, then the INPUT report will be 8 bytes long.

If a host sends an OUTPUT report it will appear in this channel as a single data packet

It is essential that all INPUT or OUTPUT reports maintain the packet boundaries.

Note 4:
These channels are used to receive INPUT reports from Hid devices and send OUTPUT reports when the module is configures as a HID host. The size and format of the reports shall be as per the Hid Descriptor active on that channel.

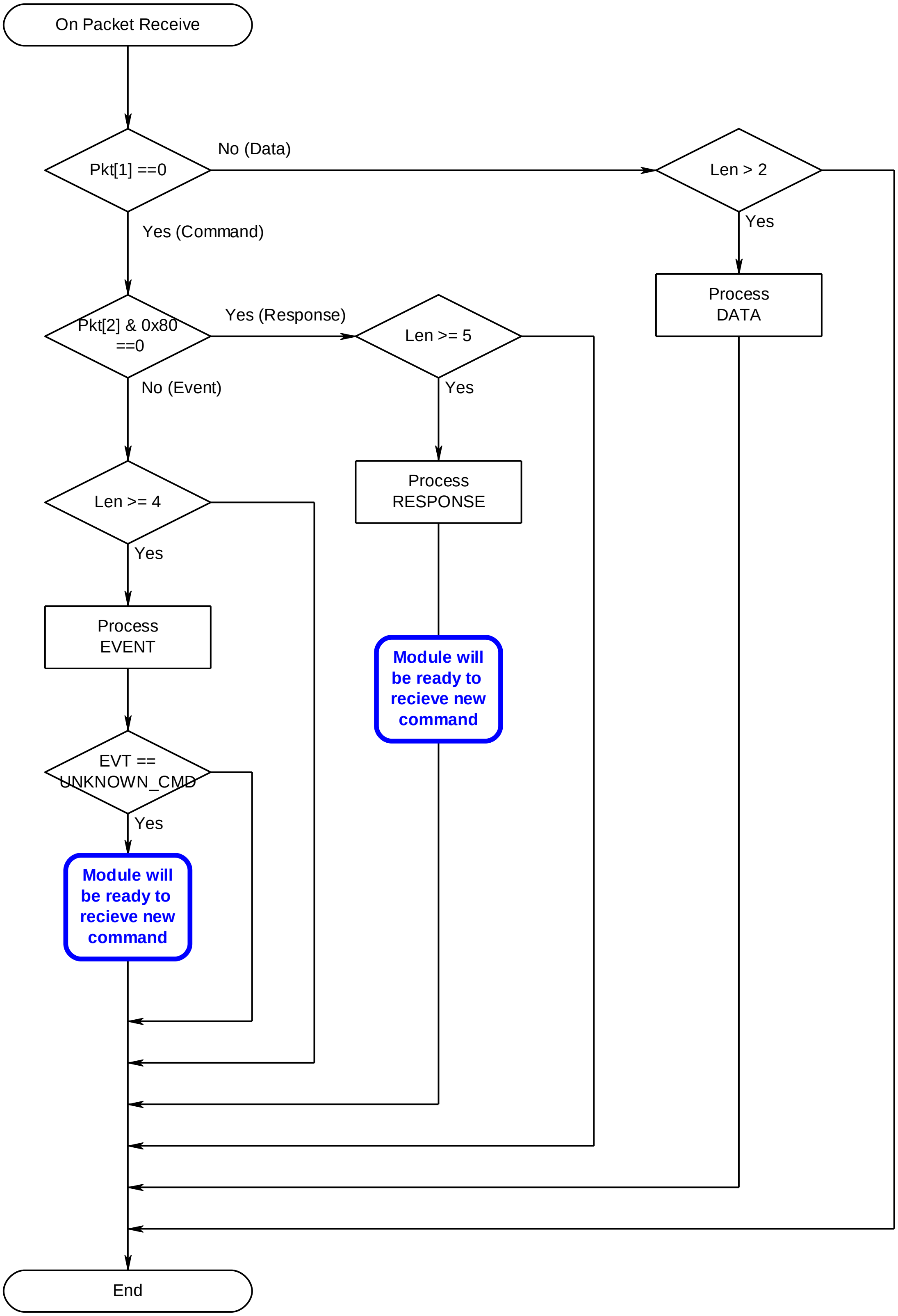
It is essential that all INPUT or OUTPUT reports maintain the packet boundaries.

Note 5:
These channels are used to send and receive data from BLOBS (Binary Long Objects) which are just arbitrary length data buffers. If a blob does not exist, then the data sent on that channel will be silently discarded. The blobs are identified by a 0 based index number, so to send data to blob 0, channel 0x98 is used and 0x99 for blob 1.

Once data is 'uploaded' into a blob, then CMD_BLOBMANAGE is used to manipulate that data. For example, the BLOB mechanism is used to upload new HID Device Descriptors into the module.

## Host Packet Receive Flowchart

As optimal data throughput is the design goal, the format and detail of packets have been constructed appropriately. It is recommended that the host should implement the following flowchart, for rapid servicing and flow control of packets.



## Host Command/Responses

This section describes all host commands and confirms in detail and is specified via the CMD_ID/CNF_ID field of all command and confirm packets.

The description for each command/confirm below is in the form of a command or confirm packet table and a corresponding response packet table when appropriate.

Each command has a unique CMD_ID value in the range 1 to 63 (0x01 to 0x3F). 0 is reserved and confirm will a CNF_ID in the range 64 to 127 (0x40 to 0x7F).

The actual value of CMD_ID in the Value column is described as [Descriptive_Name] where "Descriptive_Name" can be found in a 'C' header file which can be obtained on request from Laird.

The value of STATUS is similarly defined in a header file which can also be obtained from Laird.

The commands are grouped as :-

- Informational
- Configuration
- Connection
- Inquiry
- Pairing
- Miscellaneous

and are described in subsequent sub chapters.

## Information Commands

This group of commands are used to obtain information about the module.

## No Operation

This command results in no action other than to convey new FLOW_IN status to the module and get a response packet with the latest status for the FLOW_OUT bits.

It is expected that a host will use this packet to poll for a change in the flow bits.

| **COMMAND PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 4 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_NO_OPERATION] | |
| 3 | FLOW_IN | ?? | Runtime value |

| **RESPONSE PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 5 | |
| 1 | CHANNEL | 0 | |
| 2 | COMMAND | [CMD_NO_OPERATION] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | [OK] | Or [INVALID_LICENSE] |

## Get Connectable, Discoverable, Security Modes

This command is used to get the current connectable, discoverable and security modes.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 4 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_GET_MODES] | |
| 3 | FLOW_IN | ?? | Runtime value |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 8 | |
| 1 | CHANNEL | 0 | |
| 2 | COMMAND | [CMD_GET_MODES] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | OK or INVALID_LICENSE | |
| 5 | DISCMODE | 0..3 | Bit 0: 1 for discoverable mode<br>Bit 1: 0 for generic, 1 for limited discovery mode.<br>Bits 4..7: Future use specifying which limited inquiry access code to use. |
| 6 | CONNMODE | 0..3 | Bit 0: 1 for connectable mode<br>Bit 1: 1 for Auto Accept Channe1 |
| 7 | SECMODE | 12..15 | 12 = SSP + IO_CAP_NO_INPUT_NO_OUTPUT<br>13 = SSP + IO_CAP_DISPLAY_YES_NO<br>14 = SSP + IO_CAP_KEYBOARD_ONLY<br>15 = SSP + IO_CAP_DISPLAY_ONLY |

Notes:

1. SECMODE is now driven by the 'Simple Secure Pairing' procedure which got included in and after v2.1 of the bluetooth specification
2. For 'SECMODE' the 'No i/o capability' option is equivalent to 'Just works' scenario in Simple Secure pairing.
3. When this module interacts with a pre 2.1 device it will be unconditionally forced into legacy pairing mode.
4. It is recommended that the reader should become familiar with the 'simple secure pairing' concept introduced in and all subsequent version of Bluetooth after v2.1. The best introduction is to google the phrase "bluetooth simple secure pairing".
5. The reader is also welcome to contact Laird for an informal discussion.

## Read Local Bluetooth Address

This command is used to read the Bluetooth address of the module.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 4 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_READ_BLUETOOTH_ ADDRESS] | |
| 3 | FLOW_IN | ?? | Runtime value |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 11 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_READ_BLUETOOTH_ADDRESS] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | [OK] | |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address |

## Information

This command is used to extract information from the module, for example version number.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_INFORMATION] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | INFOTYPE | 0..255 | |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 14 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_INFORMATION] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | [OK] | |
| 5 | INFOTYPE | 0..255 | Echoed from command |
| 6.13 | DATA[8] | As per the table below | |

The type of information requested is specified by the INFOTYPE parameter, as per the table below.

**INFOTYPE = GET_VERSION (0)**

| Offset | Field Name | Range | Comments |
|---|---|---|---|
| 0 | Format/Firmware/Platform ID | 0..255 | Bit 7: 1 Bit654: Spare<br>Bit3210: Firmware/Platform ID |
| 1 | STACK_MAJOR | 0..255 | CCL Version number index |
| 2 | APP_MAJOR | 0..255 | Laird Application Version number index |
| 3 | DEVELOPER/BRANCH ID | 0..255 | Bit7654: Developer ID<br>Bit3210: Branch ID |
| 4..5 | MSB/LSB of BUILD NUBMER | 0..65535 | Odd number == engineering<br>Even Number == production |
| 6 | Reserved | 0..255 | Laird private use |
| 7 | TWIG number/SDK ID | 0..255 | B7654321 : Twig number<br>B1 : Laird private use |

**INFOTYPE = GET_MANUFACTURER (1)**

| Offset | Field Name | Range | Comments |
|---|---|---|---|
| 0..7 | Manufacturer/Stack information | E.g. “CSR/CCL” | Chip manufacturer, null terminated string |

**INFOTYPE = GET_CHIP_INFO (2)**

| Offset | Field Name | Range | Comments |
|---|---|---|---|
| 0..7 | Chip Designation | E.g. “BC4-EXT” | Chip designation, null terminated string |

**INFOTYPE = GET_PHYSICAL_MEDIUM (3)**

| Offset | Field Name | Range | Comments |
|---|---|---|---|
| 0 | Physical Medium | 0 | 0=Bluetooth |
| 1..7 | Reserved | 0 | |

**INFOTYPE = GET_HARDWARE_PLATFORMID (100, 0x64)**

| Offset | Field Name | Range | Comments |
|---|---|---|---|
| 0 | Hardware ID (msb) | 0..255 | |
| 1 | Hardware ID (lsb) | 0..255 | |
| 2..7 | Unused, set to 0’s | | |

Hardware ID will be 0x0100 for the BTM4xx series module

**INFOTYPE = MEMORY POOLS (224..239 0xE0..0xEF)**

| Offset | Field Name | Range | Comments |
|---|---|---|---|
| 0..1 | Pool Block Size | E.g. 0004 | Size of block |
| 2..3 | Available Blocks | E.g. 1234 | Available and free to use |

| INFOTYPE = CCL STACK TRUNK VERSION (240 0xF0) | | | |
|---|---|---|---|
| **Offset** | **Field Name** | **Range** | **Comments** |
| 6..7 | Stack Trunk Version | E.g. 1234 | Trunk Version of CCL stack |

| INFOTYPE = CCL STACK BRANCH VERSION (241 0xF1) | | | |
|---|---|---|---|
| **Offset** | **Field Name** | **Range** | **Comments** |
| 6..7 | Stack Branch Version | E.g. 1234 | Branch Version of CCL stack |

| INFOTYPE = CCL STACK VENA VERSION (248 0xF8) | | | |
|---|---|---|---|
| **Offset** | **Field Name** | **Range** | **Comments** |
| 6..7 | Stack VENA Version | E.g. 1234 | VENA Version of CCL stack |

## Configuration Commands

This group of commands is used to configure the module.

## Read ‘S’ Register

The module is configured using 32 bit integer values which can be stored in non-volatile memory.
Valid register numbers are in the range 0 to 255.
See section ‘S Registers’ in Chapter 4 for a full list of all registers.
The following command is used to read the current value of the S register REGNO.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_READ_SREG] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | REGNO | 0 to 255 | |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 10 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_READ_SREG] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | REGNO | 0 to 255 | Echoed from Command |
| 6..9 | REGVAL[] | Register Value | REGVAL[0] is the most significant octet. |

## Write 'S' Register

This command is used to write a new value to the S register REGNO.
See section 'S Registers' in Chapter 4 for a full list of all registers.

COMMAND PACKET

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 9 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_WRITE_SREG] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | REGNO | 0 to 255 | |
| 5..8 | REGVAL[] | New Register Value | REGVAL[0] is the most significant octet. |

RESPONSE PACKET

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 10 | |
| 1 | CHANNEL | 0 | |
| 2 | COMMAND | [CMD_WRITE_SREG] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | REGNO | 0 to 255 | Echoed from Command |
| 6..9 | REGVAL[] | Register Value | REGVAL[0] is the most significant octet. |

## Store 'S' Registers

This command is used to save the current 'S' register values in cache into the non-volatile memory so that they survive a power cycle.

COMMAND PACKET

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 4 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_STORE_SREG] | |
| 3 | FLOW_IN | ?? | Runtime value |

RESPONSE PACKET

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_STORE_SREG_] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

## Default 'S' Registers

This command is used to force all S register values in cache to factory defaults.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 4 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ DEFAULT_SREG] | |
| 3 | FLOW_IN | ?? | Runtime value |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ DEFAULT_SREG_] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

## Connection Commands

This group of commands is used to manage connections.

### Set Connectable Mode

This command is used to enable/disable connectable mode and to specify auto accept parameters for channels and muxs

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ CONNECTABLE_MODE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | ENABLE | 0..1, 0xFF | 0 = Disable, 1=Enable |
| 5 | ACCEPT | Bit mask | Bit 0: Set to auto accept channel |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ CONNECTABLE_MODE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | CURMODE | 0..1 | 0 = Not connectable |

## Service Incoming Connection

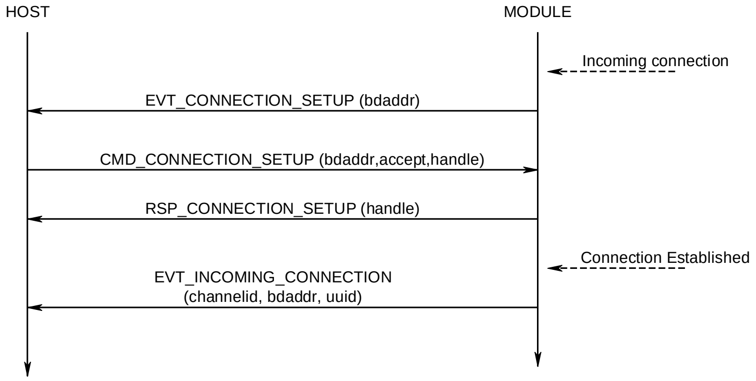
When the module is in connectable mode, incoming connection requests are passed up to the host via an EVT_CONNECTION_SETUP message – if and only if autoaccept is not enabled (via command or S register 14). The host accepts or rejects the remote connection request using this message.

**COMMAND PACKET**

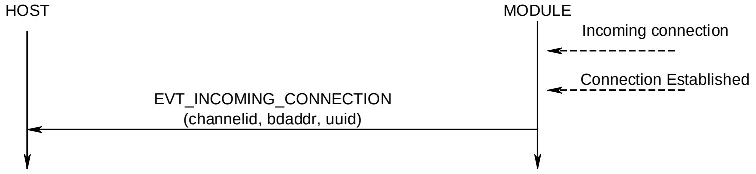
| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 12 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ CONNECTION_SETUP] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..9 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr |
| 10 | HANDLE | 0..255 | This value is echoed by the module in |
| 11 | ACCEPT | 0..1 | 0 = reject, |

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ CONNECTION_SETUP] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | HANDLE | 0..255 | Echoed from the command |

Receipt of the response is **not** an indication that the connection has been established. If the connection is to be accepted, the module will send EVT_INCOMING_CONNECTION when the connection has been fully established, as shown in the message sequence chart below.


HOST
MODULE
Incoming connection
EVT_CONNECTION_SETUP (bdaddr)
CMD_CONNECTION_SETUP (bdaddr,accept,handle)
RSP_CONNECTION_SETUP (handle)
Connection Established
EVT_INCOMING_CONNECTION
(channelid, bdaddr, uuid)

Note: If auto accept was specified when the module was put into connectable mode, then for incoming connections there will only be an EVT_INCOMING_CONNECTION message.



## Make Outgoing Connection

This command is used to make an outgoing connection to a profile in the remote peer.

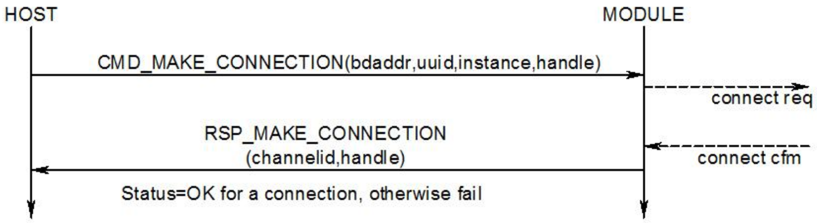
| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 13 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ MAKE_CONNECTION] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | HANDLE | 0..255 | This value is echoed by the module |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address |
| 11..12 | UUID[] | 0x1101(SPP) | Uuid of the profile to connect to. |
| 13 | RFU | 0 | Always set to 0 |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 7 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ MAKE_CONNECTION] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | HANDLE | 0..255 | Echoed from the command |
| 6 | CHANNEL | 1..7 (SPP)<br>0x20,0x9x,0xA0 (HID DEVICE) | Channel Id to be used for subsequent SPP data packets, and also when dropping the connection |

If the STATUS field in the response is MPSTATUS_OK, then a connection was successfully established. Any other value is a failure.

The content of ‘S’ register 11 is used to specify the max frame size to be used by the lower layers.

When at least one connection is active (regardless of profile) the DCD output pin will be asserted.


HOST
MODULE
CMD_MAKE_CONNECTION(bdaddr,uuid,instance,handle)
connect req
RSP_MAKE_CONNECTION
(channelid,handle)
connect cfm
Status=OK for a connection, otherwise fail

## Drop Connection

This command is used to destroy an existing channel.

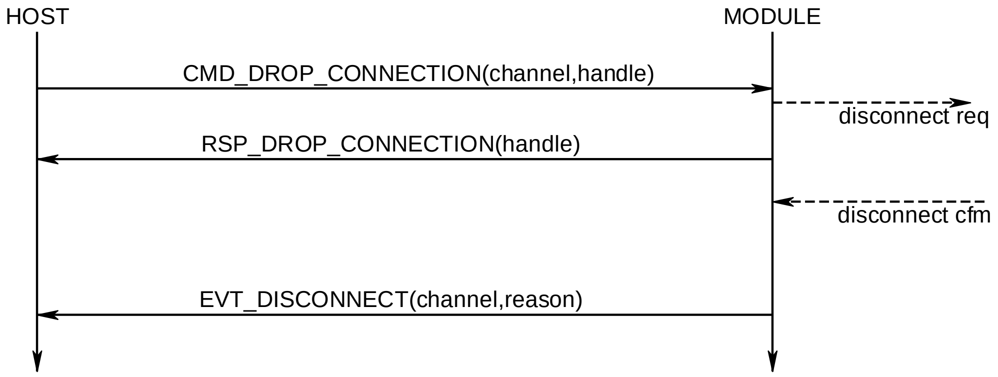
| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ DROP_CONNECTION] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | HANDLE | 0..255<br>Can be any value the host wants to set. | This value is echoed by the module in the response. |
| 5 | CHANNEL | 1..7, As appropriate for other profiles | As was specified in either RSP_MAKE_CONNECTION or EVT_INCOMMING_CONNECTION |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ DROP_CONNECTION] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | HANDLE | 0..255 | Echoed from the command |

If the STATUS field in the response is MPSTATUS_OK, then the request to drop the channel was successfully submitted to the lower layers of the stack.

When the channel is dropped, an EVT_DISCONNECT event will be sent to the host.

When at least one connection is active (regardless of profile) the DCD output pin will remain asserted.


HOST
MODULE
CMD_DROP_CONNECTION(channel,handle)
disconnect req
RSP_DROP_CONNECTION(handle)
disconnect cfm
EVT_DISCONNECT(channel,reason)

## Set Modem Lines

Bluetooth Serial Port Profile is capable of exchanging modem signals DTR, DSR, RTS, CTS, DCD and RI over air.

From a host's perspective, it can have DTR, RTS, DCD and RI as output lines. (Note DCD and RI are outputs for modems and 'host' in this context can mean either a PC or a peripheral like a modem).

Additionally UARTs are capable of sending BREAK signals. BREAK output signals are defined as a non-idle state TX pin for a period much greater than the character width at the current baud rate setting.

This command is used to send DTR, RTS, DCD and RI states to the peer device and also to specify a BREAK – Future Feature.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 7 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ CONTROLMODEMLINES] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | CHANNEL | 1..7 | Channel ID of an open channel |
| 5 | MODEM | Bit Mask | Bit 0: DTR state<br>Bit 1: RTS state<br>Bit 2: DCD state<br>Bit 3: RI state<br>Bits 4 .. 7 : Ignored |
| 6 | BREAK | 0 | Not Available |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ CONTROLMODEMLINES] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | CHANNEL | 1..7 | Echoed from command |

The STATUS value will be MPSTATUS_OK if the message was successful.

Modem signals sent by the peer device are presented to the host in the message EVT_MODEM_STATUS defined in subsequent chapters.

Note: BREAK signal capability is currently not provided by the lower stack, and so it is mentioned in the context of this command message for future implementation.

## RSSI and Link Quality

This command is used to obtain the RSSI and Link Quality values for a given open connection. This is a parameter associated with the ACL connection to a peer device and does not have any meaning with channel ids.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 10 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_RSSI_LINKQUAL] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..9 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_RSSI_LINKQUAL] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr |
| 11 | RSSI | -128 to 127 | Rssi value. Will be 0 if the signal is within the golden range |
| 12 | LINKQUAL | 0 – 255 | |

The definitions of RSSI and LINKQUAL are paraphrased from the Bluetooth specification as follows:-

**RSSI**

This value is the difference between the measured Received Signal Strength Indication (RSSI) and the limits of the Golden Receive Power Range (see below for definition). Any positive RSSI value returned by the Host Controller indicates how many dB the RSSI is above the upper limit, any negative value indicates how many dB the RSSI is below the lower limit. A value of zero indicates that the RSSI is inside the Golden Receive Power Range. Note: how accurate the dB values will be depends on the Bluetooth hardware. The only requirements for the hardware are that the Bluetooth device is able to tell whether the RSSI is inside, above or below the Golden Device Power Range.

**GOLDEN RECEIVE POWER RANGE**

The lower threshold level of the golden receive power range corresponds to a received power between -56 dBm and 6 dB above the actual sensitivity of the receiver. The upper threshold level is 20 dB above the lower threshold level to an accuracy of +/- 6 dB

**LINK QUAL**

Link_Quality is a value from 0-255, which represents the quality of the link between two Bluetooth devices. The higher the value, the better the link quality is. Each Bluetooth module vendor will determine how to measure the link quality.

In the case of CSR, this value is a measure of BER (Bit Error Rate).

### Get Open Channel List

This command is used to obtain a list of channel IDs corresponding to connections which are open. It is a good method of querying the module to see how many bluetooth connections have been established and their corresponding channel ID numbers.

A host should not really need to use this command as it should be keeping track of the following two events and responses: EVT_DISCONNECT, EVT_CONNECTION_SETUP, RSP_MAKE_CONNECTION.

| **COMMAND PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 10 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_CHANNEL_LIST] | |
| 3 | FLOW_IN | ?? | Runtime value |

| **RESPONSE PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 22 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_CHANNEL_LIST] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | Count | 0..N | Number of connections which have been established |
| 6..21 | Channel ID List[16] | Array of 16 bytes | A nonzero value implies a connection exists and the array element value identifies the channel ID |

## Inquiry Commands

This group of commands is used to performing inquiries and putting the module into discoverable mode.

### Inquiry Request

This command is used to perform a Bluetooth inquiry.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_INQUIRY_REQ] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | MAXRESP | 1..255 | Maximum number of responses before aborting the inquiry procedure |
| 5 | TIMEOUT | 1..120 | Time in seconds, before aborting the inquiry procedure. |
| 6 | FLAGS | 0..1 | Bit 0 : 1 to allow repeat addresses.<br>Bits 1..6 reserved for future<br>Bit 7 : Get Enhanced Inquiry responses |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 7 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_INQUIRY_REQ] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | TOTAL | ?? | The total number of inquiry responses that were received from peers. |
| 6 | DUMP | ?? | The total number of inquiry result events that were NOT sent because the transmit buffer of the module was full. This will be as a result of the host deasserting its RTS line. |

As a result of this command, as and when peer devices respond with inquiry responses, for each inquiry response, if bit 7 of the FLAGS field is 0 then an event EVT_INQUIRY_RESULT is sent to the host. If bit 7 of FLAGS field is 1 then, given that enhanced inquiry data has variable length data for any given response, the entire inquiry response is sent via data channel 0xF0 to the host, with format described in subsection below.

When the number of inquiry responses specified in the command are received OR the specified time has elapsed, the final response will be sent to indicate to the host that the inquiry procedure is complete.

If the DUMP field in the response is non-zero it is indicating that the host is not reading it’s receive buffer fast enough and is resulting in RTS being deasserted towards the module.

FLAGS bit 1-4 in future will be used to specify a limited access code inquiry

See message sequence diagram in following page which illustrates that the RSP_INQUIRY_REQ message terminates the inquiry process.

## Enhanced Inquiry Data Packet Format

When enhanced inquiry responses are requested via FLAGS bit 7 being set, each inquiry response will be sent to the host in data channel 0xF0 and the packet is formatted as follows:-

LL F0 AAAAAAAAAAAA CCCCCC RR EE…….EE

Where::

**LL** is the total length of the packet, and given only the EE..EE field is of variable length, the length of the EE..EE field is calculated by subtracting 12 (decimal) from LL.

**F0** is the channel number and is fixed

**AAAAAAAAAAAA** is a 6 byte field containing the bluetooth address of the responding device.

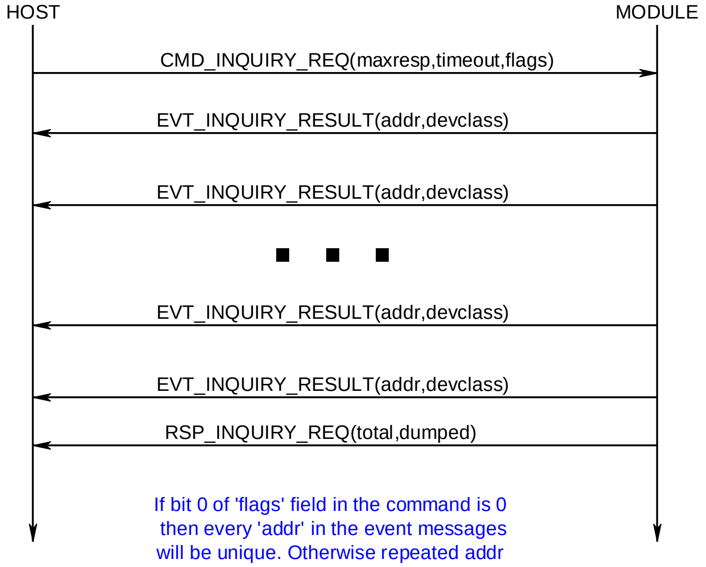
**CCCCCC** is the device class code of the responding device

**RR** is the measured rssi value for that response (8 bit signed integer)

**EE..EE** is a variable length field, with a maximum of 240 bytes, containing the enhanced inquiry data which is formatted as multiple len/tag/data structures as specified in the bluetooth specification. Where both len and tag fields are single bytes and len does not include itself.

Any data passed from the baseband must match the format defined in the Bluetooth Specification Version 2.1 + EDR [1], vol3, Part C – Generic Access Profile, 8 Extended Inquiry Response Data Format (page 1305 in the *.pdf file).

A typical message sequence will be as follows:-


HOST
MODULE
CMD_INQUIRY_REQ(maxresp,timeout,flags)
EVT_INQUIRY_RESULT(addr,devclass)
EVT_INQUIRY_RESULT(addr,devclass)
■ ■ ■
EVT_INQUIRY_RESULT(addr,devclass)
EVT_INQUIRY_RESULT(addr,devclass)
RSP_INQUIRY_REQ(total,dumped)

If bit 0 of 'flags' field in the command is 0 then every 'addr' in the event messages will be unique. Otherwise repeated addr will be seen.
Inquiry process is a stochastic process and so there is no guarantee that a peer device will be guaranteed to be detected.
However if a device reponds then it will get reported to the via an EVT message

## Set Discoverable Mode

This command is used to enable/disable discoverable mode.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ DISCOVERABLE_MODE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | MODE | 0..1, 0xFF | 0 = Disable,<br>1 = Generic Access Code<br>0xFF = Read current mode |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ DISCOVERABLE_MODE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | CURMODE | 0..1 | 1 = Generic Access Code |

The module will use the parameters stored in ‘S’ Registers 7 and 8 to set the inquiry scan interval and window. Inquiry scan is basically how often (interval) the radio will listen for an inquirer and for how long (window) each time.

## Pairing Commands

This group of commands is used to manage either incoming or outgoing pairings and to manage the trusted device database which resides in the non-volatile memory of the module.

The trusted device database is a database with a two tables, each with records of two fields. One field is the Bluetooth address of a paired device and the other is used to store the 16 byte link key.

One database is classed a ROLLING database and is used to store new pairing information as they happen. If the database is full, then the oldest is discarded to make space for the latest one.

The other database is classed as a PERSISTANT database which stores pairing information which can ONLY be deleted when a new pairing is initiated to that particular device OR on request from the host.

The host protocol provides for a command to transfer a record between these two databases. In addition there is a command for the host to determine if a device is trusted. There is also a command to manually insert a device and it's link key into the database.

Depending on the peer device, either a legacy pairing procedure will happen or a simple secure pairing. A legacy pairing will happen if the peer device is older than v2.1 of the Bluetooth specification. Simple Secure Pairing (for v2.1 and newer devices) uses a Diffie-Hellman procedure to exchange the secret link key, but is vulnerable to man-in-the-middle attack.

When pairing is initiated and a legacy 2.0 or older device is not involved, then the basebands perform an i/o capability negotiation with each other to see whether it shall perform a 'Just Works' unathenticated pairing with no man-in-the-middle (MITM) protection or an authenticated pairing which requires user interaction.

The i/o capability is one of :-

Display Only

Keyboard Only

Display with Yes/No button

For example…

If one end has Display only, but the other end has keyboard only, then the negotiation results in one end displaying a 6 digit passcode on the Display Only side, which is then required to be entered at the keyboard only end.

If both ends have Display with Yes/No, then during the procedure both ends will display a 6 digit passcode which needs to be visually compared and then the Yes/No buttons are used to confirm that they match. This provides for a 1 in a million probability that a man-in-the-middle attach will be successful.

To enable this new interaction with a user during pairing a new EVT_SIMPLE_PAIRING has been defined.

## Pair Initiate

This command is used to initiate a pairing with a peer device which is assumed to be ready and waiting for a pairing. If that device is compliant with v2.0 and older of the Bluetooth specification then a legacy pairing will result and the pincode pin[] in this message will be used otherwise there will be a simple pairing procedure.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 28 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ PAIR_INITIATE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | TIMEOUT | 5..120 | Pairing timeout in seconds |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr of device to be paired |
| 11..27 | PIN[] | 17 byte string array | Null Terminated Pin code String. Max pin code length is 16. |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ PAIR_INITIATE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

If pairing is successful the event message EVT_LINK_KEY or EVT_LINK_KEY_EX will be sent to the host before the response, to close the procedure, as shown in the message sequence chart below.

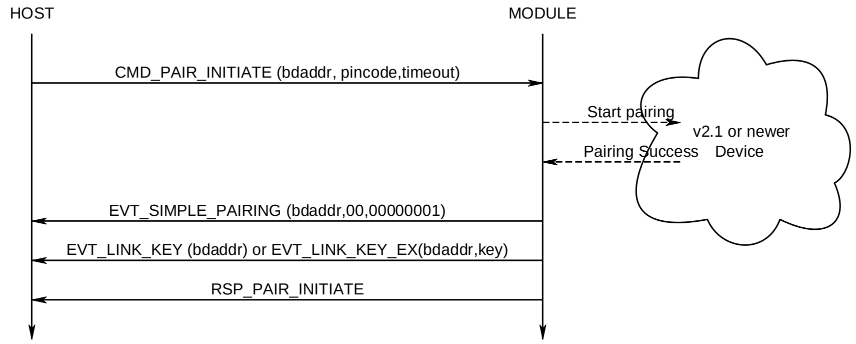
Pair Initiate may fail if there are any existing open connections. The status byte in the response will have an appropriate value.

## Legacy Pairing

The message sequence diagram for legacy pairing is as follows:-

## Simple Secure Pairing ‘Just Works’

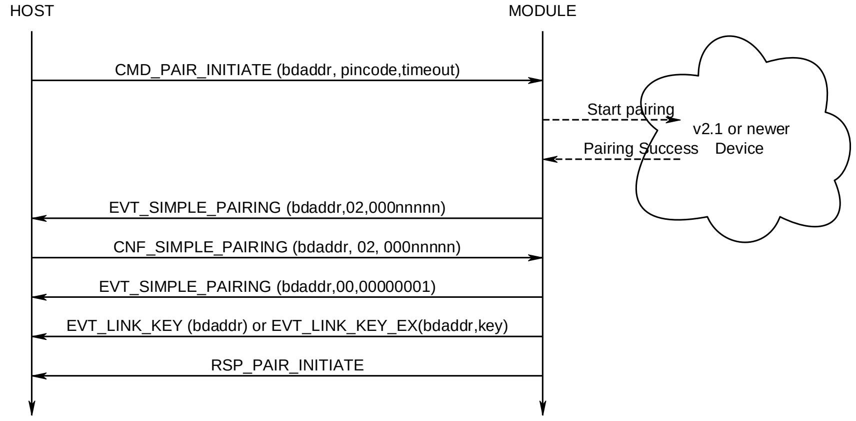
The message sequence diagram for ‘Just Works’ when the device has specified no i/o capability is as follows:-


HOST
MODULE
CMD_PAIR_INITIATE (bdaddr, pincode,timeout)
Start pairing
v2.1 or newer Device
Pairing Success
EVT_SIMPLE_PAIRING (bdaddr,00,00000001)
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)
RSP_PAIR_INITIATE

The EVT_SIMPLE_PAIRING will have a success/fail indication. If it fails, then the EVT_LINK_KEY event will not be sent to the host.

## Simple Secure Pairing ‘Display Yes/No’

The message sequence diagram for when the device has display and yes/no capability is as follows:-

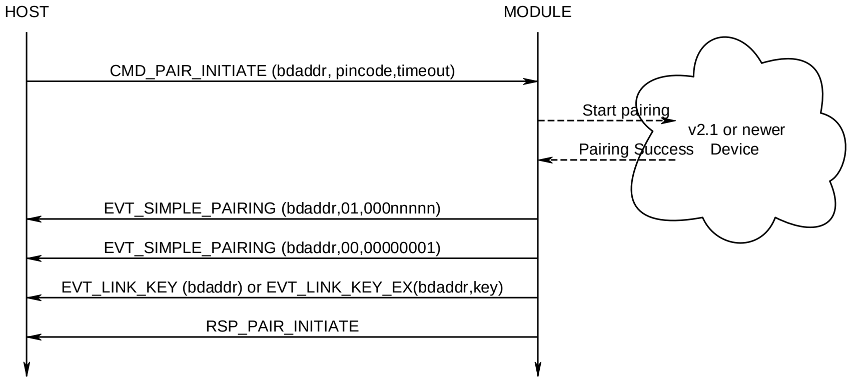

HOST
MODULE
CMD_PAIR_INITIATE (bdaddr, pincode,timeout)
Start pairing
v2.1 or newer Device
Pairing Success
EVT_SIMPLE_PAIRING (bdaddr,02,000nnnnn)
CNF_SIMPLE_PAIRING (bdaddr, 02, 000nnnnn)
EVT_SIMPLE_PAIRING (bdaddr,00,00000001)
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)
RSP_PAIR_INITIATE

The first EVT_SIMPLE_PAIRING contains a passcode that needs to be confirmed by the host. To confirm the passcode, the host shall send a CNF_SIMPLE_PAIRING packet which has a passcode value of 00000001 and to reject it shall use a value of 00000000.

If pairing is successful, another EVT_SIMPLE_PAIRING with a success value will be sent and then a EVT_LINK_KEY event will be sent to the host, if not the second EVT_SIMPLE_PAIRING witll have a 00000000 value.

## Simple Secure Pairing 'Display Only'

The message sequence diagram for when the device has display only capability is as follows:-

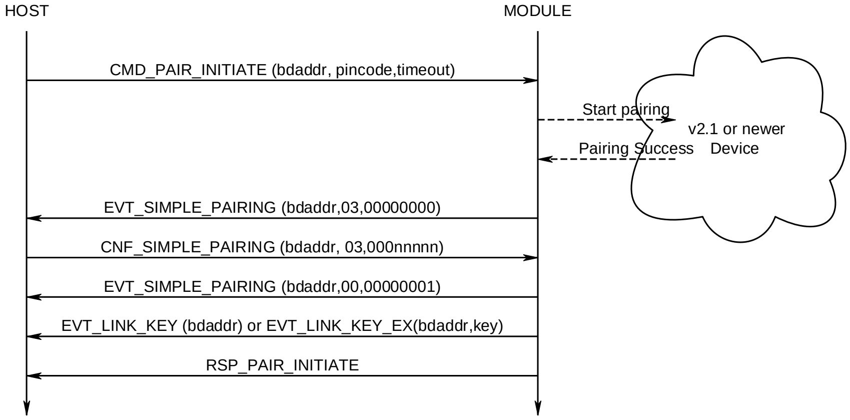

HOST
MODULE
CMD_PAIR_INITIATE (bdaddr, pincode,timeout)
Start pairing
v2.1 or newer
Device
Pairing Success
EVT_SIMPLE_PAIRING (bdaddr,01,000nnnnn)
EVT_SIMPLE_PAIRING (bdaddr,00,00000001)
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)
RSP_PAIR_INITIATE

The first EVT_SIMPLE_PAIRING contains the passcode which needs to be displayed. When the peer confirms the passcode or otherwise a second EVT_SIMPLE_PAIRING will be sent to the host with an appropriate success/fail code.

If pairing is successful, EVT_LINK_KEY will be sent to the host, if not RSP_PAIR_INITIATE will indicate a non-ok status code

## Simple Secure Pairing 'Keyboard Only'

The message sequence diagram for when the device has keyboard only capability is as follows:-


HOST
MODULE
CMD_PAIR_INITIATE (bdaddr, pincode,timeout)
Start pairing
v2.1 or newer
Device
Pairing Success
EVT_SIMPLE_PAIRING (bdaddr,03,00000000)
CNF_SIMPLE_PAIRING (bdaddr, 03,000nnnnn)
EVT_SIMPLE_PAIRING (bdaddr,00,00000001)
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)
RSP_PAIR_INITIATE

The first EVT_SIMPLE_PAIRING indicates that a simple pairing procedure has started and at that point the host shall respond with a confirm packet which contains the passcode that was visually obtained from the peer device (Or both peers decided to use the same code). When the peer confirms the pairing a second EVT_SIMPLE_PAIRING will be sent to the host with an appropriate success/fail code.

If pairing is successful, EVT_LINK_KEY will be sent to the host, if not RSP_PAIR_INITIATE will indicate a non-ok status code

## Incoming Pairing Procedure

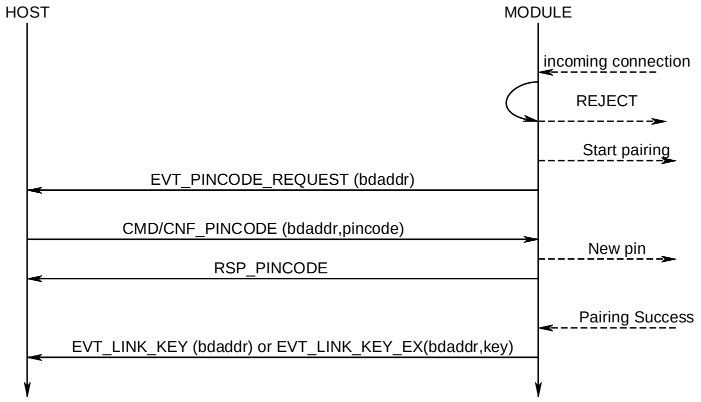
As long as the module is in connectable mode (see CMD_CONNECTABLE_MODE) it will be in a mode to accept pairing from peers.

In that situation, when a pairing procedure is detected, a pincode request event packet will be sent to the host and the host shall respond with a CMD_PINCODE or CNF_PINCODE command as shown in the message sequence diagram below. It shall respond with CNF_PINCODE if it is in the middle of making a connection and a response for 'make connection' is still pending.

If the host does not send a pincode command then the peer which initiated the pairing will eventually timeout.

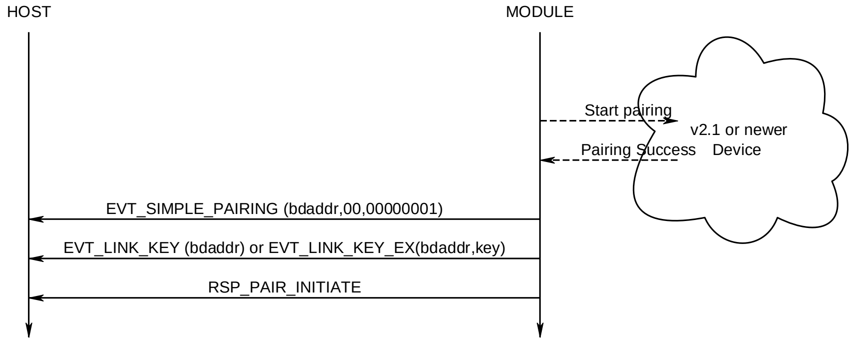
## Legacy Pairing

The message sequence diagram for incoming legacy pairing is as follows:-


HOST
MODULE
incoming connection
REJECT
Start pairing
EVT_PINCODE_REQUEST (bdaddr)
CMD/CNF_PINCODE (bdaddr,pincode)
New pin
RSP_PINCODE
Pairing Success
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)

## Simple Secure Pairing 'Just Works'

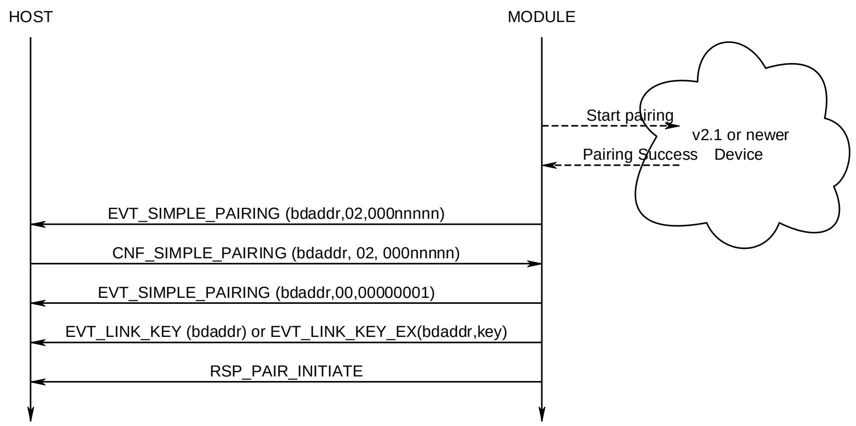
The message sequence diagram for 'Just Works' when the device has specified no i/o capability is as follows:-


HOST
MODULE
Start pairing
v2.1 or newer Device
Pairing Success
EVT_SIMPLE_PAIRING (bdaddr,00,00000001)
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)
RSP_PAIR_INITIATE

The EVT_SIMPLE_PAIRING will have a success indication.

## Simple Secure Pairing 'Display Yes/No'

The message sequence diagram for when the device has display and yes/no capability is as follows

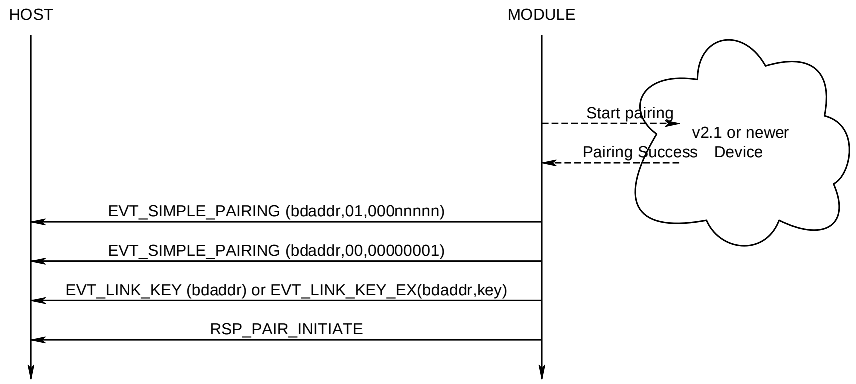

HOST
MODULE
Start pairing
v2.1 or newer Device
Pairing Success
EVT_SIMPLE_PAIRING (bdaddr,02,000nnnnn)
CNF_SIMPLE_PAIRING (bdaddr, 02, 000nnnnn)
EVT_SIMPLE_PAIRING (bdaddr,00,00000001)
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)
RSP_PAIR_INITIATE

The first EVT_SIMPLE_PAIRING contains a passcode that needs to be confirmed by the host. To confirm the passcode, the host shall send a CNF_SIMPLE_PAIRING packet which echoes the passcode, To reject it shall use any non-matching value.

If pairing is successful, another EVT_SIMPLE_PAIRING with a success value will be sent and then a EVT_LINK_KEY event will be sent to the host, if not the second EVT_SIMPLE_PAIRING witll have a 00000000 value.

## Simple Secure Pairing 'Display Only'

The message sequence diagram for when the device has display only capability is as follows:-

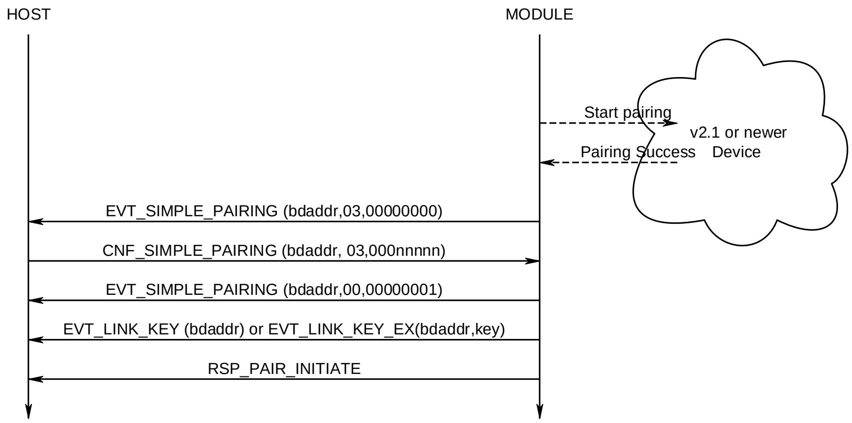

HOST
MODULE
Start pairing
v2.1 or newer Device
Pairing Success
EVT_SIMPLE_PAIRING (bdaddr,01,000nnnnn)
EVT_SIMPLE_PAIRING (bdaddr,00,00000001)
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)
RSP_PAIR_INITIATE

The first EVT_SIMPLE_PAIRING contains the passcode which needs to be displayed. When the peer confirms the passcode or otherwise a second EVT_SIMPLE_PAIRING will be sent to the host with an appropriate success/fail code.

If pairing is successful, EVT_LINK_KEY will be sent to the host, if not RSP_PAIR_INITIATE will indicate a non-ok status code

## Simple Secure Pairing 'Keyboard Only'

The message sequence diagram for when the device has keyboard only capability is as follows:-


HOST
MODULE
Start pairing
v2.1 or newer Device
Pairing Success
EVT_SIMPLE_PAIRING (bdaddr,03,00000000)
CNF_SIMPLE_PAIRING (bdaddr, 03,000nnnnn)
EVT_SIMPLE_PAIRING (bdaddr,00,00000001)
EVT_LINK_KEY (bdaddr) or EVT_LINK_KEY_EX(bdaddr,key)
RSP_PAIR_INITIATE

The first EVT_SIMPLE_PAIRING indicates that a simple pairing procedure has started and at that point the host shall respond with a confirm packet which contains the passcode that was visually obtained from the peer device (Or both peers decided to use the same code). When the peer confirms the pairing a second EVT_SIMPLE_PAIRING will be sent to the host with an appropriate success/fail code.

If pairing is successful, EVT_LINK_KEY will be sent to the host, if not RSP_PAIR_INITIATE will indicate a non-ok status code

## SIMPLE PAIRING Confirmation

This confirmation packet is used to provide a response to the module as a result of a EVT_SIMPLE_PAIRING event and is used to convey to the module additional information required to complete a simple pairing procedure.

This is a confirmation packet and so will not result in a response from the module.

| **EVENT PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 15 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [CNF_ SIMPLE_PAIRING] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4..9 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of pairing device |
| 10 | action | 2..3 | See description below |
| 11..14 | actionval | 4 bytes | See description below |

**Action :: 0 & 1**
This is illegal and will be ignored by the module.

**Action :: 2**
This informs the module the response to a YES/NO query. Set 'actionval' to 00000000 for NO and non-00000000 for YES.

**Action :: 3**
This informs the module the response to a passcode query. In this case the 'actionval' is used to convey the passcode value.

Note:
This confirmation packet is required to allow a successful connection to an unpaired 2.1 and newer device because the MP protocol does not process new commands unless the previous command has been completed by having sent an appropriate response packet.

4 The packet processor in the module queues all commands until a response is sent, but processes confirmation packets as they come.

## PinCode (Command)

This command is used to send a pincode in response to an EVT_PINCODE_REQUEST message and will result in a RSP_PINCODE from the module to the host.

This command can also be used to register a pincode for all subsequent incoming legacy pairings from BT2.0 and older devices. In that case the BDADDR[] field MUST be set to all 0's

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 28 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_PINCODE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | TIMEOUT | 5..120 | Pairing timeout in seconds |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of device requesting the pairing.<br>Or all 0's to just register a pincode for incoming pairings |
| 11..27 | PIN[] | 17 byte string array | Null Terminated Pin code String.<br>Max pin code length is 16. |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_PINCODE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

If pairing is successful the event message EVT_LINK_KEY or EVT_LINK_KEY_EX will be sent to the host after the response. The latter event wil be sent if S Register 47 is set to 1.

## PinCode (Confirmation)

This is a confirmation packet which is used to send a pincode in response to an EVT_PINCODE_REQUEST message while making an outgoing connection to a legacy pairing device and there has been a pairing challenge by the peer prior to connection acceptance.

This packet, just like the command version of this packet can also be used to register a pincode for all subsequent incoming legacy pairings from BT2.0 and older devices. In that case the BDADDR[] field MUST be set to all 0's

| **COMMAND PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 28 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CNF_PINCODE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | TIMEOUT | 5..120 | Pairing timeout in seconds |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of device requesting the pairing.<br>Or all 0's to just register a pincode for incoming pairings |
| 11..27 | PIN[] | 17 byte string array | Null Terminated Pin code String.<br>Max pin code length is 16. |

If pairing is successful the event message EVT_LINK_KEY or EVT_LINK_KEY_EX will be sent to the host after the response. The latter event wil be sent if S Register 47 is set to 1.

Note:
This confirmation packet is required to allow a successful connection to an unpaired 2.0 and older device because the MP protocol does not process new commands unless the previous command has been completed by having sent an appropriate response packet.

5 The packet processor in the module queues all commands until a response is sent, but processes confirmation packets as they come hence in this case CMD_PINCODE will not work.

### Trusted Database Record Count

This command is used to obtain the number of trusted devices in the database specified.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ TRUSTED_DB_COUNT] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | DBTYPE | 0..1 | 0 = ROLLING DATABASE<br>1 = PERSISTANT DATABASE |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 8 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ TRUSTED_DB_COUNT] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | DBTYPE | 0..1 | Echoed from command |
| 6 | COUNT | 0..N | Number of trusted devices in this database |
| 7 | MAXCOUNT | 0..N | Maximum number of devices that can be stored in this database |

Notes

ROLLING database is used to store all new pairings automatically. It is up to the host to transfer a record from ROLLING to the PERSISTANT database so that it does not get overwritten.

## Trusted Database Read Record

This command is used to read the Bluetooth address of the ITEMNO pairing in the database specified, counted from the top. ITEMNO is indexed from 1.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_TRUSTED_DB_READ] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | DBTYPE | 0..1 | 0 = ROLLING DATABASE<br>1 = PERSISTANT DATABASE |
| 5 | ITEMNO | 1..N | Item number to read |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 13 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_TRUSTED_DB_READ] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | DBTYPE | 0..1 | Echoed from command |
| 6 | ITEMNO | 1..N | Echoed from command |
| 7..12 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr.<br>Will be all 0's if the item specified does not exist |

If S Reg 47 is set to 1, then it is possible to read a record so that both the address and link key information is supplied by sending the command CMD_ TRUSTED_DB_ISTRUSTED. In that case the event EVT_LINK_KEY_EX will be sent before the response to CMD_TRUSTED_DB_ISTRUSTED.

### Trusted Database Delete Record

This command is used to delete a pairing from the databases. It is not necessary to specify the database type, as both databases will be scanned.

**COMMAND PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 10 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ TRUSTED_DB_DELETE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..9 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr of device to be unpaired |

**RESPONSE PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 11 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ TRUSTED_DB_DELETE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr, echoed from the command |

### Trusted Database Change Type

This command is used to transfer a record to the database specified.

**COMMAND PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 11 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_TRUSTED_DB_CHANGETYPE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | DBTYPE | 0..1 | |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr |

RESPONSE PACKET

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 12 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_TRUSTED_DB_CHANGETYPE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | DBTYPE | 0..1 | Echoed from command |
| 6..11 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr, echoed from the command |

### Trusted Database Is Peer Trusted

This command is used to check if a device is trusted.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 10 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ TRUSTED_DB_ISTRUSTED] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..9 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 11 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ TRUSTED_DB_ISTRUSTED] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5..10 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr, echoed from the command |

The STATUS value will be MPSTATUS_OK if the device is trusted, any other value means not trusted.

If S Reg 47 is set to 1, then if the peer device is found in the trusted device database, then the event EVT_LINK_KEY_EX will be set to the host BEFORE the response. That event contains the address and link key associated with that address.

## Trusted Database Add Key (Out-Of-Band Facilitator)

This command is used to manually add a link key/address pair into the rolling trusted database.

The module does not care how the key was generated and the only validation it performs is to check that it is 16 bytes long.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 1E | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ TRUSTED_DB_ADD] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..9 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr |
| 10..25 | KEY[16] | 16 byte Link Key | Any random value |
| 26..29 | FLAGS[4] | 00 00 00 00 | For future use and MUST be set to 00000000 |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 11 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ TRUSTED_DB_ADD] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

The STATUS value will be MPSTATUS_OK if the device was successfully added to the database.

If the device bluetooth address is already in the trusted device database, then the old one is deleted and this new one replaces it.

## Miscellaneous Commands

### Get Security Mode

This command is used to get the current security mode of the module. Bluetooth v2.1 and newer devices cannot disable security. Therefore only values 12 to 15 inclusive are valid. These values specify Simple Secure Pairing with appropriate i/o capabilities.

Specifying a value of 0xFF means leave the mode as it is, but inform the host with regards to current mode.

**COMMAND PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_SECURITY_MODE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | SECMODE | 0xFF | 0xFF = Get current mode |

**RESPONSE PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_SECURITY_MODE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | SECMODE | 12..15 | Current mode<br>12 = SSP with no input no output<br>13 = SSP with yes/no display<br>14 = SSP with keyboard only<br>15 = SSP with Display only |

1. Notes:
   SECMODE is now driven by the 'Simple Secure Pairing' procedure which got included in and after v2.1
   of the bluetooth specification
2. For 'SECMODE' the 'No i/o capability' option is equivalent to 'Just works' scenario in Simple Secure pairing.
3. When this module interacts with a pre 2.1 device it will be unconditionally forced into legacy pairing mode.
4. It is recommended that the reader should become familiar with the 'simple secure pairing' concept introduced in and all subsequent version of Bluetooth after v2.1. The best introduction is to Google the phrase "bluetooth simple secure pairing".
5. The reader is also welcome to contact Laird for an informal discussion.

## Get Remote Friendly Name

This command is used to get the friendly name of the specified peer device.

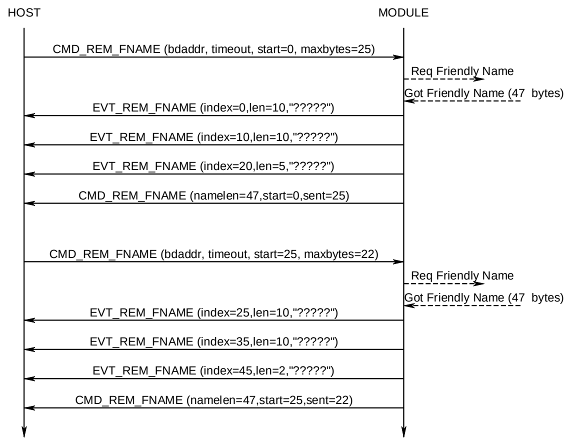
According to the Bluetooth specification a friendly name can be up to 248 bytes long. Sending this name in one go to the host could violate the max packet length capability of the host due to memory restrictions in the host OR transmit buffers in the module may not be able to cope. Therefore, the mechanism for getting the name to the host is via multiple event packets EVT_REM_FNAME. The host decides how many bytes of the name is to be passed up to it via these events from the offset it also specifies. This implies that in a memory constraint environment, it will be possible to relay the name to the host using multiple commands.

For example, if the host has space for only 10 bytes and a peer happens to have a very long name, the host can ask for 10 byte fragments of the name over multiple get name requests.

| **COMMAND PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 13 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_GET_REM_FNAME] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..9 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth addr |
| 10 | TIMEOUT | 2..120 | Timeout in seconds |
| 11 | START | N | Offset into the friendly name string |
| 12 | MAXBYTES | M | Max characters to read |

| **RESPONSE PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 8 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_GET_REM_FNAME] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | NAMELEN | 0..248 | Actual size of the friendly name |
| 6 | START | N | Echoed from the command |
| 7 | SENTLEN | S | Total number of bytes sent |

1. Note: SENTLEN could be less than MAXBYTES. It can happen if there is no space in the module’s TX buffer to send events.


HOST
MODULE
CMD_REM_FNAME (bdaddr, timeout, start=0, maxbytes=25)
Req Friendly Name
Got Friendly Name (47 bytes)
EVT_REM_FNAME (index=0,len=10,"?????")
EVT_REM_FNAME (index=10,len=10,"?????")
EVT_REM_FNAME (index=20,len=5,"?????")
CMD_REM_FNAME (namelen=47,start=0,sent=25)
CMD_REM_FNAME (bdaddr, timeout, start=25, maxbytes=22)
Req Friendly Name
Got Friendly Name (47 bytes)
EVT_REM_FNAME (index=25,len=10,"?????")
EVT_REM_FNAME (index=35,len=10,"?????")
EVT_REM_FNAME (index=45,len=2,"?????")
CMD_REM_FNAME (namelen=47,start=25,sent=22)

## Get Local Friendly Name

This command is used read the local friendly name which is stored in non volatile memory.

Unlike the remote friendly name where there is no control over the max length, the local friendly name is limited to 31 characters.

This length still may too big to send to the host in one packet. Therefore the name is sent in a similar fashion to ‘get friendly name’ described above. However, in this case, the event EVT_LCL_FNAME is used to get the name to the host.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_GET_LCL_FNAME] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | START | n | Offset into the friendly name string |
| 5 | MAXBYTES | m | Max characters to read |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_GET_LCL_FNAME] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | NAMELEN | 0..31 | Actual size of the friendly name |
| 6 | START | n | Echoed from the command |
| 7 | SENTLEN | S | Total number of bytes sent in preceding events. |

The name string is sent to the host in EVT_LCL_FNAME packets. See description of Get Remote Friendly name

2. Note:
SENTLEN could be less than MAXBYTES. It can happen if there is no space in the module’s TX buffer to send events.

### Set Local Friendly Name

This command is used to set the local friendly name in non-volatile memory so that it is used after a power up/reset cycle.

Since the module can cope with large packets, the name sent to the module in a single command packet as a null terminated string.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 37 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_SET_LCL_FNAME] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | FLAGS | 1..2 | 1 = Write to non-vol store for use on next power up<br>2 = Make the name visible now |
| 5 | NAMELEN | 1..30 | |
| 6..36 | NAME[31] | Null terminated string | Not more than 30 characters |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_SET_LCL_FNAME] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

### Set Device Class

This command is used to set the device class code so that the base band makes it visible immediately. This is opposed to setting it via S Register 120 which is only expedited on power up.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 7 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_SET_DEVCLASS] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..6 | DEVCLASS[3] | New 24 bit device class | MSB of device class is at offset 4 |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_SET_DEVCLASS] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

### Factory Default

This command is used to clear non-volatile memory in the module so that it reverts to factory default state.

The FLAGS field is a bit mask which is used to selectively clear various types of non-volatile memory and should be set to FF.

**COMMAND PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_FACTORYDEFAULT] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 5 | FLAGS | 8 bit mask | See notes below<br>0xFF to delete all groups, present and future |

**RESPONSE PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_FACTORYDEFAULT] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

Notes:
Non-volatile information in the module is divided into various subgroups and the FLAGS bits allow selective resetting to factory state for those subgroups. The bits are allocated as follows:-
Bit 0 : S registers
Bit 1 : Local Friendly Name
Bit 2 : HID Device Descriptos
Bit 3 : Reserved – Always set to 1
Bit 4 : Reserved – Always set to 1
Bit 5 : Reserved – Always set to 1
Bit 6 : Link Keys
Bit 7 : Special S registers (240 to 255 in the multipoint space)

## Get Digital/Analog I/O

This command is used read the states of up to 16 digital input lines and optionally request an analogue input reading.

This response packet contains 2 octets containing the digital input states. If an analogue input reading is requested then the ADC reading will be supplied in an EVENT_ADC event.

**COMMAND PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_GET_IO] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | digId | 0 | 0 = Digital I/o in Module |
| 5 | analogId | 0 | 0 = No ADC access<br>1..255 = FUTURE USE |

**RESPONSE PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_GET_IO] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | digId | 0 | Echoed from command packet |
| 6..7 | digIn[2] | ?? | Digital inputs 0 to 15. Bit 15 is bit 7 in the MSB.<br>See also Note 2 |

Note 1:
If analogId field in the command is non zero an EVENT_ADC will be generated when the ADC is read and available
Note 2:
Bit 0 corresponds to GPIO0, Bit 1 to GPIO1 etc. Please refer to the module’s data sheet to check which GPIO pins are available for use.

## Set Digital I/O

This command is used to control the states of up to 16 digital output lines.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_SET_IO_EX] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | ioId | 0 | 0 = Digital I/o in Module |
| 5..6 | mask[2] | 0000..FFFF | Only set bits are specify which bits in ioVal will be submitted to the digital i/o<br>See Note 1 |
| 7..8 | ioVal[2] | 0000..FFFF | Only the bits set in mask will bet updated at the output<br>See Note 1 |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_SET_IO] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

Note 1:
Bit 0 corresponds to GPIO0, Bit 1 to GPIO1 etc. Please refer to the module's data sheet to check which GPIO pins are available for use.

## Reset

This command is used to reset the module.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 7 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_RESET] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..6 | RESVD[3] | 00 00 00 | Reserved for future use, set to 0s |

There will be no response to this command packet. However, on reset there will be at least one EVT_STATUS event so that can be used to detect that the device has rebooted.

## Blob Manage

This command is used to manage binary data uploaded from the host through data channels 0x98 to 0x9F (the number of blob data channels is compile time dependent). The binary data is referred to as a 'blob' (**b**inary **l**ong **ob**ject).

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 14 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_BLOBMANAGE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4 | subCmdId | 0..9 | |
| 5 | blobId | 0..1 | Identifies the blob that this command acts on.<br>Data Channels will correspond to (0x98 + blobId) |
| 6..9 | ParmA | Offset 6 is MSB<br>Offset 9 is LSB | Parameter A |
| 10..13 | ParmB | Offset 10 is MSB<br>Offset 13 is LSB | Parameter B |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 15 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_ BLOBMANAGE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | subCmdId | Echoed from command | |
| 6 | blobId | Echoed from command | |
| 7..10 | ParmA | Offset 7 is MSB<br>Offset 10 is LSB | Depends on subCmdId |
| 11..14 | ParmB | Offset 11 is MSB<br>Offset 14 is LSB | Depends on subCmdId |

**Action to perform**

SubCmdId field is used to specify the action that the blob manager should take on the blob specified in field blobId and are described in the following sub sections.

## Clear :: 0

Use this subcommand to clear the blob specified by 'blobId', On return ParmA and ParmB are set to 0.

**Get Size :: 1**

Use this subcommand to get the number of bytes in the blob specified by 'blobId', On return ParmA contains the number and ParmB is set to 0.

**Read :: 2**

Use this subcommand to transmit the content of the blob specified by 'blobId'. If there is data in the blob then it will result in one or more data packets being sent to the host on channel (0x98+BlobId).The data in the blob is not destroyed. On return ParmA contains the number of bytes in the blob and ParmB contains the number actually sent in the data packets.

**Save Hid Descriptor :: 3**
Use this subcommand to first do a basic validation of the data block identified by blobId to see that it contains a valid HID descriptor and if so, save the data to a non-volatile memory set aside to save an array of hid descriptors. The index into that array is specified by ParmA which is referred to as the HidId

In the response, ParmA contains 0 for success, 1 for invalid hid descriptor, 2 for failed to save to nonvol array and 000000FF for invalid blobId or invalid HidId.

**Load Hid Descriptor :: 4**
Use this subcommand to append the content of the hid descriptor in non-volatile memory identified by a hidId which is specified in ParmA into the data object identified by blobId.

In the response, ParmA contains 0 for success, 2 for failed to load from nonvol array and 000000FF for invalid blobId or invalid HidId.

After the blob is loaded, use subcommand Read::2 to force the transmission of the hid descriptor to the host.

**Save Custom Hid Descriptor Service name :: 5**
Use this subcommand to save the service name (Less than 30 characters) that will be used when a custom hid device sdp record is registered (When SReg 39 > 0).

In the response, ParmA contains 0 for success, 1 for invalid service anme, 2 for failed to save to nonvol array and 000000FF for invalid blobId.

**Load Custom Hid Descriptor Service name :: 6**
Use this subcommand to append the content of the hid service name in non-volatile memory.

In the response, ParmA contains 0 for success, 2 for failed to load from nonvol array and 000000FF for invalid blobId.

After the blob is loaded, use subcommand Read::2 to force the transmission of the name to the host.

**Commit blob as Enhanced Inquiry :: 7**
Use this subcommand to commit the current content of the blob specified as Enhanced Inquiry Responses. The data is validated so that it conforms to the format specified in described in the Bluetooth Specification Version 2.1 + EDR [1], vol3, Part C – Generic Access Profile, 8 Extended Inquiry Response Data Format (page 1305 in the .pdf-file).

Basically the data block consists of one or more objects, where the first byte of each object specifies the length and the second byte specifies the type of data and the rest of the bytes for that object are the associated data.

For example, to send the string “HelloWorld” and “LairdRules!” as the manufacturer specific data objects, then the blob should be loaded with the following data :-

0B FF ‘H’ ‘e’ ‘l’ ‘l’ ‘o’ ‘W’ ‘o’ ‘r’ ‘l’ ‘d’ 0C FF ‘L’ ‘a’ ‘I’ ‘r’ ‘d’ ‘R’ ‘u’ ‘l’ ‘e’ ‘s’ ‘!’

Where 0B and 0C are length bytes and FF is the type value for ‘manufacturer specific data’

Before submission, the data block is verified to see if the length bytes are consistent with total length of the block.

**Save Enhanced Inquired Response Data :: 8**
Use this subcommand to save the blob data in non-volatile memory so that the data is used to install the custom EIR data response on every power up. Before committing to memory, the data block is verified to see if the length bytes are consistent with total length of the block.

**Load Enhanced Inquired Response Data:: 9**
Use this subcommand to load the blob specified with data from non-volatile memory associated with Enhanced Inquiry responses. Basically the data that was saved using subcommand 8 described.

## HDP Profile Commands

Health Device Profile (HDP) is available on the module in both Agent and Manager roles as defined by the Continua Alliance (see www.continua.org). The manager role is for **testing** the agent implementation only and **it is not expected or designed for eventual certification by the Continua Alliance**.

It is assumed that the reader is familiar with all the HDP and IEEE documentation and relevant guidelines published by the Continua Alliance.

For further background information related to the HDP implementation in this module please refer to Chapter 8 “AT Applications Examples.”

The HDP commands, responses and events described in the subsequent chapters are merely the MP way of managing the functionality and does not add any new features other than those described in the AT Applications Example section. Hence the reader is recommended to refer to the AT Applications Example section for further details.

To assist with this cross referencing of details, the table below shows the equivalent commands in AT and MP mode respectively to achieve the same effect in the module.

| AT Mode Commands | MP Mode Commands | Comments |
|---|---|---|
| AT+HAEiiii,”name”<br>AT+HMEiiii,”name” | CMD_HDP_ENDPOINT | Affects an entry in the SDP record |
| AT+HAL<br>AT+HML | CMD_HDP_SDPREGISTER | Register and activate HDP service record |
| AT+HABeeeee,iiii | CMD_HDP_BIND | Create an agent data specialization of type iiii which will connect to a manager with Bluetooth address ‘eeeeee’ |
| AT+HAAhhhh | CMD_HDP_ASSOCIATE | |
| AT+HARhhhh,pppp[…] | CMD_HDP_SCANREPORT_FIXED<br>CMD_HDP_SCANREPORT_VAR | In MP, the attribute list for var report is supplied via a blob channel. |
| AT+HAGhhhh,aaaa,ssss<br>AT+HMGhhhh,oooo,aaaa | CMD_HDP_ATTRIBUTE_READ | Read an attribute value. In MP mode the data value is sent in data channel B0 |
| AT+HAShhhh,aaaa,ssss,dddddd | CMD_HDP_ATTRIBUTE_WRITE | Write an attribute value. In MP mode the data value is taken from data channel B0 |
| AT+HMThhhh,ttttt | CMD_HDP_SETTIME | Used at manager end to send new date/time to the agent |
| AT+HADhhhh | CMD_HDP_DISASSOCIATE | |

The following table shows the equivalent AT asynchronous responses and MP events.

| AT Mode Async Responses | MP Mode Events | Comments |
|---|---|---|
| HDA:DISASSOCIATED hhhh<br>HDM:DISASSOCIATED hhhh | EVT_HDP_DISASSOCIATED | Sent to host when the agent disassociates from the manager or fails to connect to it |
| HDA:ASSOCIATED hhhh,..<br>HDM:ASSOCIATED hhhh,.. | EVT_HDP_ASSOCIATED | |
| HDA:TIME hhhh,ttttt | EVT_HDP_TIMEUPDATE | At Agent end only.<br>Agent host is informed of new date/time from manager.<br>Manager triggers this using the command AT+HMD or CMD_HDP_SETTIME |

## HDP related S Registers

To enable HDP profile, Bit 2 in **SReg 3** must be set to 1 and **SReg 70** specifies whether it is an agent role ( = 0 ) or a manager role ( = 1 ).

In addition **SReg 71** is used to specify the auto-disassociate timeout in the agent role only, where 0 is used to denote no timeout. If this is a non-zero value, then after an association or the triggering of a scan report if no further activity occurs in that time, then an automatic disassociation will be initiated.

**SReg 72** is used to specify the maximum transmit PDU size for HDP packets in the underlying Bluetooth transport layer

## Create Endpoint in SDP Record

This command is used to create a data specialization source (for agent role) or a sink ( for manager role) which will be registered using the CMD_HDP_SDPREGISTER command. This SDP entry will allow peers to determine which data specialization is serviced by the module.

It is not necessary to specify which role explicitly in the command because S Reg 70 is used to determine whether the module has been configured for Agent or Manager role. For Agent role SReg 70 is set to 0 and 1 for Manager.

This command is relevant for both Agent and Manager roles.

**COMMAND PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 22 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_ENDPOINT] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..5 | SpecType[2] | As per IEEE spec [4]=msb,[5]=lsb | Data Specialization code. E.g 100F (4111 dec) for Weigh Scale |
| 6..21 | Name[15+1] | Null Terminated Name | Max name size is 15 and name MUST be terminated by a NULL |

**RESPONSE PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_ENDPOINT] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

Multiple data specialization endpoints can be created by submitting this command an appropriate number of times.

Internally, in the module, the SpecType[2] and Name[15+1] information is cached in heap memory until the command CMD_HDP_SDPREGISTER transfers that information into the SDP record for final submission to the underlying Bluetooth stack.

### Register SDP record

This command is used to register and activate a HDP related SDP record after the endpoints have been created using CMD_HDP_ENDPOINT. Only after this has been done, can incoming HDP connections be serviced.

It is not necessary to specify which role explicitly in the command because S Reg 70 is used to determine whether the module has been configured for Agent or Manager role. For Agent role SReg 70 is set to 0 and 1 for Manager.

This command is relevant for both Agent and Manager roles.

| **COMMAND PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 4 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_SDPREGISTER] | |
| 3 | FLOW_IN | ?? | Runtime value |

| **RESPONSE PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_SDPPREGISTER] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

Internally, in the module, the SpecType[2] and Name[15+1] information previously cached in heap memory from CMD_HDPENDPOINT commands is transferred into a SDP record and submitted to the underlying stack. After this a peer will be able see this device offering HDP services.

## Bind Agent to a Manager

This command is used to bind the Bluetooth address of a manager and an agent data specialization to **create an object** within the module which can be subsequently referenced by  a 16 bit handle which is returned in the response if the command is successfully accepted.

This command is relevant for Agent role only.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 14 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_BIND] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..9 | BDADDR[] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address |
| 10.11 | SpecType[2] | As per IEEE spec [10]=msb,[11]=lsb | Data Specialization code. E.g 100F (4111 dec) for Weigh Scale |
| 12.13 | AssocTout[2] | [12]=msb,[13]=lsb | Automatic deassociation timeout. 0 = no timeout |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 7 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_BIND] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5..6 | HANDLE[2] | [5]=msb,[6]=lsb | Handle identifying this instance of data specialization agent |

The handle returned in the response, if STATUS equals 0, is subsequently used in many commands. If the STATUS byte is not 0, then the HANDLE value shall be 0.

## Associate with Manager

This command is used to associate an agent with a manager, using the pre-defined object created using the CMD_HDP_BIND command which shall have returned a handle to represent that object.

This command is relevant for Agent role only.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_ASSOCIATE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..5 | HANDLE[2] | [4]=msb,[5]=lsb | Handle identifying an instance of data specialization agent/manager combination |

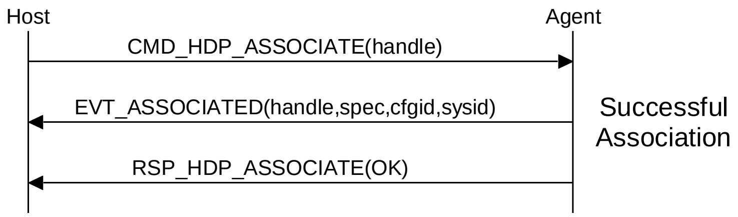
**RESPONSE PACKET**

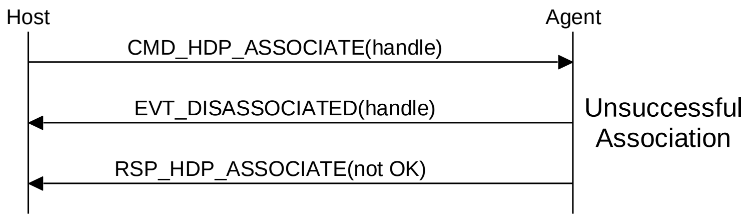
| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_ASSOCIATE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

The host shall wait until the response is received before submitted further commands.

If association is successful prior to receiving the response, the host will be sent EVT_ASSOCIATED, which will contain the config id of the MDS configuration that was negotiated and the system id of the peer. See EVT_ASSOCIATED for further details.

If association was not successful, because for example, the manager is not in range or the Bluetooth device specified in CMD_HDP_BIND does not offer HDP services, then the EVT_DISASSOCIATED will be sent to the host prior to the response message. This is shown in the message sequence diagram below.


Host
Agent
CMD_HDP_ASSOCIATE(handle)
EVT_ASSOCIATED(handle,spec,cfgid,sysid)
RSP_HDP_ASSOCIATE(OK)
Successful
Association


Host
Agent
CMD_HDP_ASSOCIATE(handle)
EVT_DISASSOCIATED(handle)
RSP_HDP_ASSOCIATE(not OK)
Unsuccessful
Association

### Send Fixed Scan Report to Manager

This command is used to associate (if not already associated) **and** send a fixed scan report from an agent to the bound manager. The binding specified via the CMD_HDP_BIND command which shall have returned a handle to represent that combination object.

This command is relevant for Agent role only, and will result in a EVT_HDP_SCANREPORT event at the manager end.

A fixed report sends the values for a list of attributes to a manager where the list is predefined in the MDC_ATTR_ATTRIBUTE_VAL_MAP attribute of the NU collection object.

| **COMMAND PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 9 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_SCANREPORT_FIXED] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..5 | HANDLE[2] | [4]=msb,[5]=lsb | Handle identifying an instance of data specialization agent/manager combination |
| 6..7 | PERSONID[2] | [6]=msb,[7]=lsb | This identifies a person ID which the manager can uses appropriately |
| 8 | HOSTCONTEXT | XX | This is echoed back in the response and helps the host to keep track of them |

| **RESPONSE PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 8 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_SCANREPORT_FIXED] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5..6 | HANDLE[2] | [4]=msb,[5]=lsb | Echoed back from the command |
| 7 | HOSTCONTEXT | XX | This is echoed back from the command |

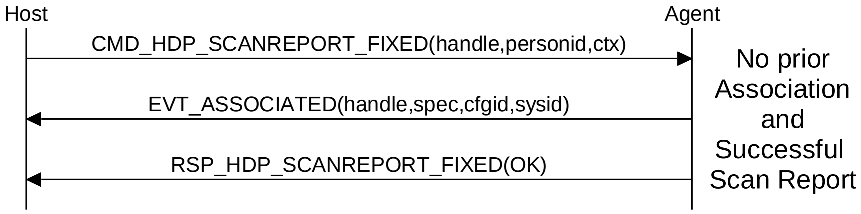
The host shall wait until the response is received before submitted further commands.

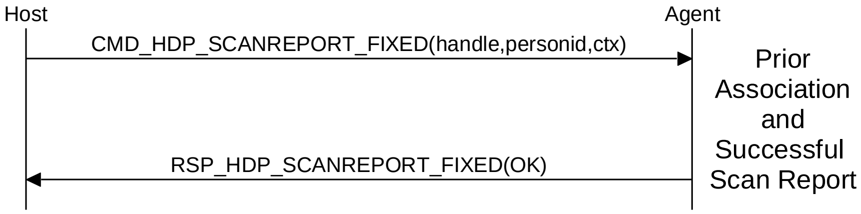
If the agent is not associated, then this command will first result in an association request and if that is successful the host will be sent EVT_ASSOCIATED which will contain the config id of the MDS configuration that was negotiated and the system id of the peer. See EVT_ASSOCIATED for further details.

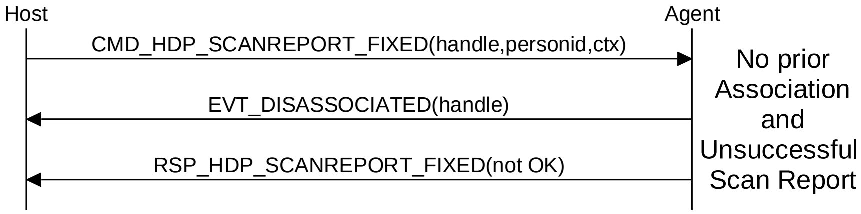
If the agent is already associated, then there will be a response as soon as there is acknowledgement from the manager that the scan report has been received.

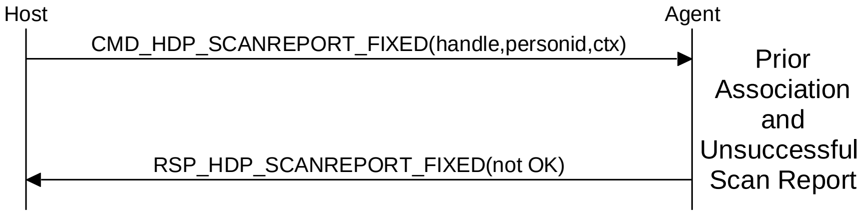
If association was not successful, because for example the manager is not in range or the Bluetooth device specified in CMD_HDP_BIND does not offer HDP services, then the EVT_DISASSOCIATED will be sent to the host prior to the response message. This is shown in the message sequence diagram below.

### Message Sequence chart for Fixed Scan Report


Host
Agent
CMD_HDP_SCANREPORT_FIXED(handle,personid,ctx)
EVT_ASSOCIATED(handle,spec,cfgid,sysid)
RSP_HDP_SCANREPORT_FIXED(OK)
No prior Association and Successful Scan Report


Host
Agent
CMD_HDP_SCANREPORT_FIXED(handle,personid,ctx)
RSP_HDP_SCANREPORT_FIXED(OK)
Prior Association and Successful Scan Report


Host
Agent
CMD_HDP_SCANREPORT_FIXED(handle,personid,ctx)
EVT_DISASSOCIATED(handle)
RSP_HDP_SCANREPORT_FIXED(not OK)
No prior Association and Unsuccessful Scan Report


Host
Agent
CMD_HDP_SCANREPORT_FIXED(handle,personid,ctx)
RSP_HDP_SCANREPORT_FIXED(not OK)
Prior Association and Unsuccessful Scan Report

## Send VAR Scan Report to Manager

This command is used to associate (if not already associated) and then send a VAR scan report from an agent to the bound manager, the binding is specified via the CMD_HDP_BIND command which shall have returned a handle to represent that combination.

This command is relevant for Agent role only, and will result in a EVT_HDP_SCANREPORT event at the manager end.

A VAR report sends the values for a list of attributes in the NU Collection object to a manager where the list is pre-supplied by the host. If an attribute does not exist in the NU collection then it will be silently ignored. The list is provided by the host via the blob data channel for the BLOBID specified in the command (that is, channel number 0x98 plus BLOBID).

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 10 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_SCANREPORT_VAR] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..5 | HANDLE[2] | [4]=msb,[5]=lsb | Handle identifying an instance of data specialization agent/manager combination |
| 6..7 | PERSONID[2] | [6]=msb,[7]=lsb | This identifies a person ID which the manager can uses appropriately |
| 8 | HOSTCONTEXT | XX | This is echoed back in the response and helps the host to keep track of them |
| 9 | BLOBID | 0..N | See section “Blob Manage” for details of how to send blob data using channel 0x98 for blobID=0 etc |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 8 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_SCANREPORT_VAR] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5..6 | HANDLE[2] | [4]=msb,[5]=lsb | Echoed back from the command |
| 7 | HOSTCONTEXT | XX | This is echoed back from the command |

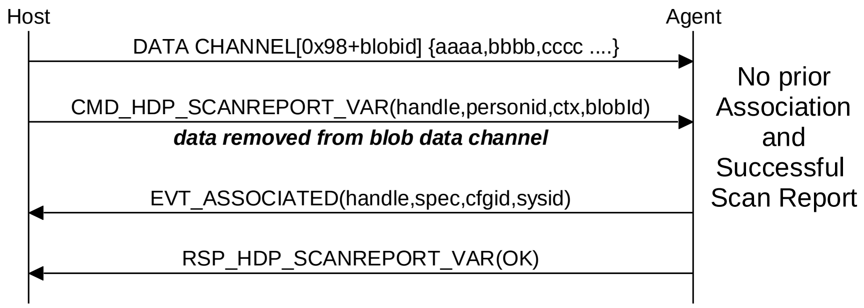
The host shall wait until the response is received before submitted further commands.

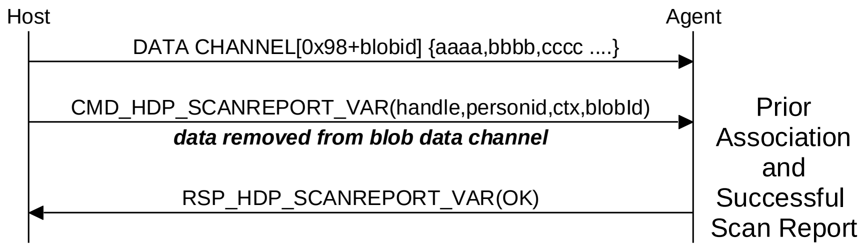
If the agent is not associated, then this command will first result in an association request and it that is successful the host will be sent EVT_ASSOCIATED which will contain the config id of the MDS configuration that was negotiated and the system id of the peer. See EVT_ASSOCIATED for further details.

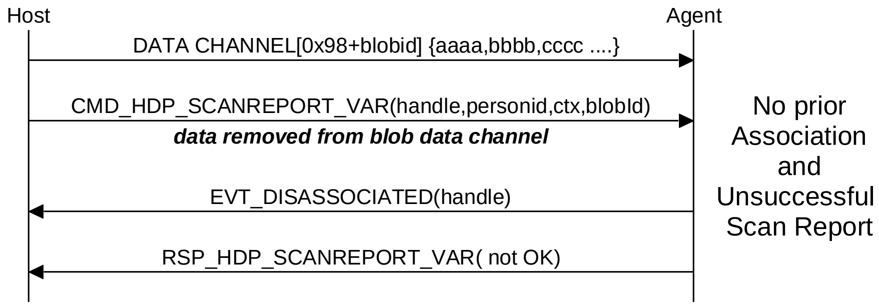
If the agent is already associated, then there will be a response as soon as there is acknowledgement from the manager that the scan report has been received.

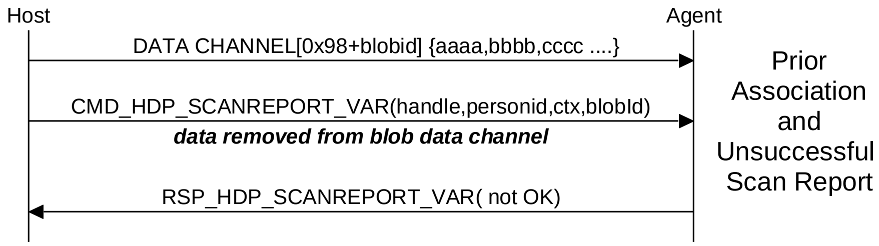
If association was not successful, because for example the manager is not in range or the Bluetooth device specified in CMD_HDP_BIND does not offer HDP services, then the EVT_DISASSOCIATED will be sent to the host prior to the response message. This is shown in the message sequence diagram below.

## Message Sequence chart for VAR Scan Report


Host
Agent
DATA CHANNEL[0x98+blobid] {aaaa,bbbb,cccc ....}
CMD_HDP_SCANREPORT_VAR(handle,personid,ctx,blobId)
data removed from blob data channel
EVT_ASSOCIATED(handle,spec,cfgid,sysid)
RSP_HDP_SCANREPORT_VAR(OK)
No prior Association and Successful Scan Report


Host
Agent
DATA CHANNEL[0x98+blobid] {aaaa,bbbb,cccc ....}
CMD_HDP_SCANREPORT_VAR(handle,personid,ctx,blobId)
data removed from blob data channel
RSP_HDP_SCANREPORT_VAR(OK)
Prior Association and Successful Scan Report


Host
Agent
DATA CHANNEL[0x98+blobid] {aaaa,bbbb,cccc ....}
CMD_HDP_SCANREPORT_VAR(handle,personid,ctx,blobId)
data removed from blob data channel
EVT_DISASSOCIATED(handle)
RSP_HDP_SCANREPORT_VAR( not OK)
No prior Association and Unsuccessful Scan Report


Host
Agent
DATA CHANNEL[0x98+blobid] {aaaa,bbbb,cccc ....}
CMD_HDP_SCANREPORT_VAR(handle,personid,ctx,blobId)
data removed from blob data channel
RSP_HDP_SCANREPORT_VAR( not OK)
Prior Association and Unsuccessful Scan Report

## Read Attribute Value

This command is valid for both agent and manager role and is used to read the value of the attribute specified via the attribute id and the qualifier Id. The value is returned in HDP data channel 0xB0 formatted as described below.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 10 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_ATTRIBUTE_READ] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..5 | HANDLE[2] | [4]=msb,[5]=lsb | Handle identifying an instance of data specialization agent/manager combination |
| 6..7 | ATTRID[2] | [6]=msb,[7]=lsb | |
| 8..9 | QUALIFIERID[2] | [8]=msb,[9]=lsb | See Note 1 |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 8 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_ATTRIBUTE_READ] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | HOSTCONTEXT | XX | This is echoed back from the command |
| 6..7 | HANDLE[2] | [6]=msb,[7]=lsb | Echoed back from the command |

Note 1:
For manager role , this is the object ID (MDS=0,NU=1 etc) and for Agent please refer to 'Section Weigh Scale Data Specialization'. As more data specializations are added, the qualifierID will be specified appropriately for that specialization.

The host shall wait until the response is received before submitted further commands.

If the attribute exists, then the data will be sent in a Logical HDP data packet transported in one or more physical data packets over channel 0xB0. The physical packets of incoming data in channel 0xB0 shall be viewed as a **stream** of data making up logical packets.

The format of the logical packet is as follows and also addressed in a dedicated section later:-

LEN[2],00,HANDLE[2],ATTRID[2],QUALIFIERID[2],DATA[N]

Where LEN[2], HANDLE[2],ATTRID[2],QUALIFIERID[2] are in big endian format (msb sent first) and N will be equal to LEN[2]+9.

The 00 after the LEN[2] field signifies that this logical packet consists of attribute data.

Since all data is transparently treated in the module, the endienness of DATA[N] should be determined by trial and error with the aid of a HDP manager and finalized to be correct by the time the implementation is submitted for certification by the Continua Alliance. However, if the attribute data type is a 16 or 32 bit integer/float, then it will be little endian.

## Write Attribute Value

This command is valid for agent role only and is used to write the value of the attribute, already preloaded in HDP data channel 0xB0, where the attribute id and the qualifierId is supplied in the command packet.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 10 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_ATTRIBUTE_WRITE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..5 | HANDLE[2] | [4]=msb,[5]=lsb | Handle identifying an instance of data specialization agent/manager combination |
| 6..7 | ATTRID[2] | [6]=msb,[7]=lsb | |
| 8..9 | QUALIFIERID[2] | [8]=msb,[9]=lsb | See Note 1 |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 8 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_ATTRIBUTE_WRITE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |
| 5 | HOSTCONTEXT | XX | This is echoed back from the command |
| 6..7 | HANDLE[2] | [6]=msb,[7]=lsb | Echoed back from the command |

***Note 1:***
***For manager role , this is the object ID (MDS=0,NU=1 etc) and for Agent please refer to Chapter 8. As more data specializations are added, the qualifierID will be specified appropriately for that specialization.***

The host shall wait until the response is received before submitted further commands.

If the attribute exists, then the data will is moved from channel 0xB0 to the attribute container variable. If the attribute does not exist, then the data is discarded. All the data in the channel is treated as the data for the attribure, that is, there is no qualifying information affixed to the data. In addition, if the attribute exists, but the data length does not match that of the attrivute, then the write will fail (with an appropriate error code) and the data in the channel will be discarded.

Since all data is transparently treated in the module, the endienness of DATA[N] should be determined by trial and error with the aid of a HDP manager and finalized to be correct by the time the implementation is submitted for certification by the Continua Alliance. However, if the attribute data type is a 16 or 32 bit integer/float, then it will be little endian.

## Set Date and Time

This command is valid for manager role only and is used to update the date and time in the associated agent identified by the handle. When any HDP manager sends a time update to an agent, it will result in an EVT_HDP_UPDATE event to the host of that agent.

**COMMAND PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 14 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_SET_TIME] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..5 | HANDLE[2] | [4]=msb,[5]=lsb | Handle identifying an instance of data specialization agent/manager combination |
| 6..13 | DATETIME[8] | Ccyymmddhhssnnxx | See Note 1 |

**RESPONSE PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 14 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_SET_TIME] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

Note 1:
The date/time information is supplied in this command as a string of 8 bytes "ccyymmddhhssnnxx" where each byte is as follows:-

CC Century e.g for 2011 the value shall be 0x14

YY Year e.g for 2011 the value shall be 0x0B

MM Month e.g for January the value shall be 0x01 and for say December 0x0C

DD Day e.g for 31 the value shall be 0x1F (0 is illegal value)

HH Hour e.g for 6:45pm the value shall be 0x12

NN Minutes e.g for 6:45pm the value shall be 0x2D

XX fraction This is a fraction of a seconds in hundredths of unit. Valid 00..0x63 (99)

For example the date and time "2 Feb 2011, 16:43:33.78" shall be sent at the string "150C020C102D214E"

The host shall wait until the response is received before submitted further commands.

### Disassociate From Manager

This command is used to disassociate an agent identified by the handle specified in the command from a manager.

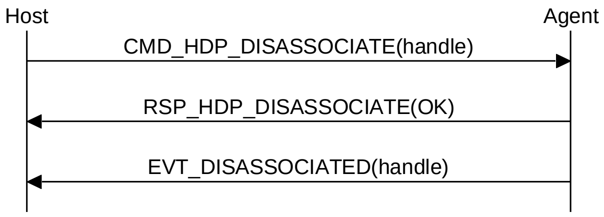
This command is relevant for Agent role only.

| COMMAND PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 6 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_DISASSOCIATE] | |
| 3 | FLOW_IN | ?? | Runtime value |
| 4..5 | HANDLE[2] | [4]=msb,[5]=lsb | Handle identifying an instance of data specialization agent/manager combination |

| RESPONSE PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 5 | Fixed |
| 1 | CHANNEL | 0 | Fixed |
| 2 | COMMAND | [CMD_HDP_DISASSOCIATE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | As appropriate | |

The host shall wait until the response is received before submitted further commands.

The response will be sent immediately after initiating the disassociation and when the procedure is complete an EVT_DISASSOCIATED will be sent to the host as shown in the message sequence diagram below.



## Module Events

This section describes all module originated asynchronous events in detail and is specified via the EVT_ID field of all event packets.

The description for each event below is in the form of an event packet tables.

Each event has a unique EVT_ID value in the range 129 to 255 (0x81 to 0xFF), 0x80 is reserved.

The actual value of EVT_ID in the Value column is described as [Descriptive_Name] where "Descriptive_Name" can be found in a 'C' header file which can be obtained on request from Laird.

### Inquiry Events

This group of events is inquiry related.

### Inquiry Result

This event is used to send the inquiry response from a peer as a result of an inquiry request.

| **EVENT PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 13 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ INQUIRY_RESULT] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4..9 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of responding device |
| 10..12 | DEVCLASS[3] | 0x000000 .. 0xFFFFFF | Device class of responding device |

## Information Events

This group of events is used to convey information about the module, for example to status.

### Unknown Command

This event is used to inform the host that a command was received with an unknown COMMAND value. The COMMAND value is echoed in offset 4.

| **EVENT PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | | |
| 1 | CHANNEL | | |
| 2 | EVENT | [EVT_UNKNOWN_COMMAND] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | Command | xx | CMD_ID value echoed from command |

### Status

This event is used to asynchronously send current status to the host. This event is sent to the host after power up to inform the host that the module is ready and operational. The information contained in this message can also be obtained by sending the CMD_GET_MODES command.

| **EVENT PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 8 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ STATUS] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | STATUS | OK or INVALID_LICENSE | |
| 5 | DISCMODE | 0..1 | 1 for discoverable mode |
| 6 | CONNMODE | 0..1 | Bit 0: 1 for connectable mode<br>In future bit 1, if set, may indicate that incoming SPP connections are auto accepted |
| 7 | SECMODE | 12..15 | 12 = SSP + IO_CAP_NO_INPUT_NO_OUTPUT<br>13 = SSP + IO_CAP_DISPLAY_YES_NO<br>14 = SSP + IO_CAP_KEYBOARD_ONLY<br>15 = SSP + IO_CAP_DISPLAY_ONLY |

### Invalid Packet Size

This event is used to inform the host that a command packet has been received whose length does not match the size of the struct published in the interface header file BmHostProtocol.h.

**EVENT PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 7 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_INVALID_PKTSIZE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | CMD_ID | 1..127 | Echoed from the command |
| 5 | ACT_SIZE | A | Actual size of the packet |
| 6 | DES_SIZE | D | Desired size of the packet |

## Connection Events

This group of events are connection related.

### Connection Setup

This event is used to inform the host that a remote device is requesting a connection.

The host shall respond with a CMD_CONNECTION_SETUP with an accept or reject flag.

**EVENT PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 12 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ CONNECTION_SETUP] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4..9 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of device requesting connection |
| 10..11 | UUID[] | Server profile uuid the peer wishes to connect to | 0x1101 = SPP<br>0x1124 = HID DEVICE |

The UUID field tells the host to which server profile the peer wishes to connect to.

### Incoming Connection

This event is used to inform the host that an incoming connection has been established.

**EVENT PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 13 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ CONNECTION_SETUP] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | CHANNEL | 1..254 | Channel ID to be used to send/receive data, see note 1 for channel number allocation. |
| 5..10 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of device requesting connection |
| 11..12 | UUID[] | Server profile uuid the peer wishes to connect to | 0x1101 = SPP<br>0x1124 = HID DEVICE |

The UUID field tells the host to which server profile the peer has connected to.

Note 1: Channel number allocation is as follows:-

| Profile | Channel ID Range |
|---|---|
| Command Parser | 0x00 |
| SPP | 0x1 .. 0x7 |
| HID DEVICE(Canned) | 0x20 |
| HID HOST(Raw) | 0x90 .. 0x97 |
| HID DEVICE(Raw) | 0xA0 |
| BLOB | 0x98 .. 0x9F |
| HDP | 0xB0..0xB1 (B1 is continuation channel) |
| Enhanced Inquiry Response | 0xF0 |

### Disconnect

This event is used to inform the host that a connection has been dropped by the remote device.

| EVENT PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 6 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ DISCONNECT] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | CHANNEL | 1..7 | Channel number |
| 5 | REASON | 0..255 | As per the table below |

The reason value is specified in the Bluetooth specification. Please note that values in the range 0xF0 to 0xFF are custom values defined for this implementation and do not appear in the Bluetooth specification.

| | |
|---|---|
| 0x01<br>…<br>0x3F | See header file MPSTATUS.H which will be supplied on request. |
| 0xFF | High probability that the remote device went out range for longer than the link supervision timeout or was powered down. |

### Modem Status

This event is used to convey modem status signals originating from the peer device for an SPP connection.

| EVENT PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 6 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ MODEM_STATUS] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | CHANNEL | 1..7 | Channel ID of an open SPP channel |
| 5 | MODEMSIG | Bit Mask | Bit 0: DSR state<br>Bit 1: CTS state<br>Bit 2: DCD state<br>Bit 3: RI state |
| 6 | BREAKSIG | 0 | For future implementation |

## Miscellaneous Events

### Link Key

This event is used to inform the host that a new link key has been created for the device indicated and the result of writing to the ROLLING database.

| EVENT PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 11 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ LINK_KEY] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4..9 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of paired device |
| 10 | DBRESULT | 0: Success | Any other value is a failure and the reason is a STATUS value as per MPSTATUS.H |

### Link Key Ex

This event is sent to the host when CMD_TRUSTED_DB_ISTRUSTED is processed and a link key for that peer device exists and S Register 47 is set to 1. It is used to convey the link key along with the bluetooth address to the host.

| EVENT PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 30 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ LINK_KEY_EX] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4..9 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of paired device |
| 10..25 | KEY[16] | 16 byte array | The link key |
| 26..29 | RFU[4] | 4 byte array | Reserved for future |

### Pin Code Request

This event is used to inform the host that a remote device has requested a pairing and a pin code is required to complete the procedure.

| EVENT PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 10 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ PINCODE_REQUEST] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4..9 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of pairing device |

The host shall send a CMD_PINCODE in response to this event.
This event is only received if ‘accept pairing while in connectable mode’ is enabled via S Register 15.

### Simple Pairing

This event is used to inform the host that a simple pairing procedure is in progress and the 'action' byte in offset 10 tells the host what to do

**EVENT PACKET**

| **Offset** | **Field** | **Value** | **Comments** |
|---|---|---|---|
| 0 | LENGTH | 15 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_ SIMPLE_PAIRING] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4..9 | BDADDR[6] | Nap[0,1]:Uap[2]:Lap[3,4,5] | Bluetooth address of pairing device |
| 10 | action | 0..3 | See description below |
| 11..14 | actionval | 4 bytes | See description below |

The host shall react to this event based on the value of the 'action' byte in offset 10 as follows:-

**Action :: 0**

This informs the host that a simple pairing procedure has just been completed and 'actionval' will be 00000001 for success and 00000000 for fail. The host shall NOT react to this event with the CNF_SIMPLE_PAIRING packet.

**Action :: 1**

This informs the host that a simple pairing procedure is in progress and all it shall do is display the passcode in 'actionval' as a 6 digital decimal number WITH leading zeroes. The host shall NOT react to this event with the CNF_SIMPLE_PAIRING packet.

**Action :: 2**

This informs the host that a simple pairing procedure is in progress and it shall display the passcode in 'actionval' as a 6 digital decimal number WITH leading zeroes. The host shall react to this event with the CNF_SIMPLE_PAIRING packet with a 00000000 value for No and a non-00000000 value for Yes. The former value, if the host deems that the passcode is not acceptable, and the latter if acceptable.

**Action :: 3**

This informs the host that a simple pairing procedure is in progress and the module is expecting a passcode from the host embedded in a CNF_SIMPLE_PAIRING packet. The host shall react to this event with the CNF_SIMPLE_PAIRING packet with a passcode value. The passcode in the CNF_SIMPLE_PAIRING is either obtained by reading the display on the peer device, or if the peer device is also a keyboard only, then the same random 6 digit value (with leading 0s) that is entered at both ends.

## Local Friendly Name

This event is used to send a fragment of the local friendly name to the host. The maximum length of the fragment is 10, so at least 3 of these events are required to convey a local friendly name, if it has the maximum length of 30.

| **EVENT PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 16 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_LCL_FNAME] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | INDEX | 0..29 | Start index into the string |
| 5 | LEN | 1..10 | Number of valid characters in the NAME[] field that follows |
| 6..15 | NAME[10] | Xx xx xx xx | The name fragment |

## Remote Friendly Name

This event is used to send a fragment of the remote friendly name to the host. The maximum length of the fragment is 10, so at least 25 of these events are required to convey a remote friendly name, if it has the maximum length of 248.

| **EVENT PACKET** | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 16 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_REM_FNAME] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | INDEX | 0..247 | Start index into the string |
| 5 | LEN | 1..10 | Number of valid characters in the NAME[] field that follows |
| 6..15 | NAME[10] | Xx xx xx xx | The name fragment |

## HDP Profile Related Events

### Associated

This event is used to inform the host that an agent has associated with the manager   and contains the handle, data specialization nominal code, device config id (as per the IEEE standard)   and a unique 8 byte identification number for the agent (or manager).

It is relevant for both agent and manager roles.

**EVENT PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 11 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_HDP_ASSOCIATED] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | ROLE | 00=Agent, 01==Manager | |
| 5..6 | HANDLE[2] | [5]=msb, [6]=lsb | Handle of the agent |
| 7..8 | SPECTYPE[2] | [7]=msb, [8]=lsb | Data specialization nominal code. E.g 4111 (0x100F) for Weigh Scale |
| 9..10 | DEVCFGID[2] | [9]=msb, [10]=lsb | The negotiated config ID. Will be defined in the appropriate IEEE data specialization standard. |
| 11..18 | SYSID[8] | 8 bytes | This is a system ID which is unique to the agent instant. |

### Deassociated

This event is used to inform the host that an agent has disassociated from the manager and contains the handle of the agent

It is relevant for both agent and manager roles.

**EVENT PACKET**

| Offset | Field | Value | Comments |
|---|---|---|---|
| 0 | LENGTH | 7 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_HDP_DISASSOCIATED] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | ROLE | 00=Agent, 01==Manager | |
| 5..6 | HANDLE[2] | [5]=msb, [6]=lsb | Handle of the agent |

### Time Update

This event is generated for an agent role only and is used to inform the host that the agent has received an updated date & time from the manager.

| EVENT PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 11 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_HDP_TIMEUPDATE] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | ROLE | 00=Agent, 01==Manager | |
| 5..6 | HANDLE[2] | [5]=msb, [6]=lsb | Handle of the agent |
| 11..14 | DATETIME[8] | ccyymmddhhssnnxx | See Note 1 |

Note 1:
The date/time information is supplied in this command as a string of 8 bytes “ccyymmddhhssnnxx” where each byte is as follows:-

CC Century e.g for 2011 the value shall be 0x14

YY Year e.g for 2011 the value shall be 0x0B

MM Month e.g for January the value shall be 0x01 and for say December 0x0C

DD Day e.g for 31 the value shall be 0x1F (0 is illegal value)

HH Hour e.g for 6:45pm the value shall be 0x12

NN Minutes e.g for 6:45pm the value shall be 0x2D

XX fraction This is a fraction of a seconds in hundredths of unit. Valid 00..0x63 (99)

For example the date and time “2 Feb 2011, 16:43:33.78” shall be sent at the string “150C020C102D214E”

## Debug Events

### Debug Packet

This event is used to convey debugging information to the host, and will be available in engineering/beta builds only.

| EVENT PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 16 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_DEBUG_PACKET] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4 | TYPE_FLAG | XX | Bit 0: First Packet<br>Bit 1: Last Packet<br>Bit 2..5: Reserved<br>Bit 6..7: Message Type |
| 5..15 | DATA[] | Contains Ascii data | String conveying message |

### Malloc Statistics

This event is used to convey pool malloc statistics to the host and will be available in engineering/beta builds only.

| EVENT PACKET | | | |
|---|---|---|---|
| **Offset** | **Field** | **Value** | **Comments** |
| 0 | LENGTH | 16 | |
| 1 | CHANNEL | 0 | |
| 2 | EVENT | [EVT_DEBUG_PACKET] | |
| 3 | FLOW_OUT | ?? | Runtime value |
| 4..5 | ELSIZE[2] | 0..N | Pool Element Size |
| 6..7 | NUMELS[2] | 0..N | Number of elements |
| 8..9 | TAKEN[2] | 0..N | Number of elements taken |
| 10..11 | MAXTKN[2] | 0..N | Tide mark for taken |
| 12..13 | OVFLO[2] | 0..N | Number of allocations from next bigger element because this size is maxed out |

## Data Channels

This section provides details of some data channels which require further explanation.

## HDP Data Channels

### Host to Module Direction

Channels B0 and B1 are used to upload attribute data which is then transferred to the appropriate data specialization data variable on receipt of a CMD_HDP_ATTRUBUTE_WRITE command.

The host shall ensure that the correct number of bytes for that attribute are accumulated in the channel as length is the only validation performed on the data. The module does not interpret the data in any way apart from length. With regards to the endienness of the data, this shall be determined by trial and error using an appropriate certified HDP manager.

It is entirely possible that an attribute can be defined which contains data requiring more than 253 bytes. A data packet cannot contain data more than 253 bytes so this could present a problem.

The solution to that is both channels B0 and B1 write into a buffer in the module to allow the host to accumulate attribute data using several data packets, but when B0 is used, it always first deletes any data already accumulated in the buffer, and then writes to that buffer, whereas writing to channel B1 shall always append the data to the buffer.

## Module to Host Direction

Channels B0 and B1 are used to send logical hdp packets to the host. Channel B0 is always the first fragment of the logical packet and subsequent fragments are sent in channel B1. This means that when the host receives a packet on channel B0, it shall delete all data accumulated for an existing ongoing logical packet.

### Logical Packet Format

The format of the HDP logical packet is

| LEN | PACKET_TYPE | DATA |
|---|---|---|
| 2 bytes | 1 byte | N bytes |

where LEN will be set to N+3 and big endian, so that the first byte of LEN as sent on the wire will be the msb. The DATA field structure depends on the logical packet type specified by PACKET_TYPE and the following subsections describe the types of packets available at the time of writing.

### Packet Type: Attribute Value

This logical packet is sent to the host as a result of processing the CMD_HDP_ATTRIBUTE_READ command in either the agent or manager roles.

For this logical packet type the PACKET_TYPE field is set to 0x00 and the DATA field consists of 4 fields as follows:-

| AGENT HANDLE | ATTR NOMINAL CODE | ATTR QUALIFIER FIELD | ATTR VALUE |
|---|---|---|---|
| 2 bytes | 2 bytes | 2 bytes | N bytes |

Where the ‘Agent Handle’, ‘Attr Nominal Code’ and ‘Attr Qualifier ID’ fields are all 2 byte fields in big endian format (msb first) and are echoed from the CMD_HDP_ATTRIBUTE_READ command and ‘Attr Value’ is the actual value of the attribute.

The length N can be calculated by subtracting 9 from the LEN field of the logical packet.

## Packet Type: Scan Report

This logical packet is sent to the host in a manager role as a result of scan report arriving from an agent.

For this logical packet type the PACKET_TYPE field is set to 0x01 and the DATA field consists of multiple fields as follows:-

| AGENT HANDLE | PERSON ID | NUM OF OBJECTS | DATA LISTs |
|---|---|---|---|
| 2 bytes | 2 bytes | 1 byte | N bytes |

Where the ‘Agent Handle’ and ‘PersonID’ fields are 2 byte fields in big endian format (msb first) and ‘Num of Objects’ is a one byte field which specifies the number of objects and the field ‘Data Lists’ consists of multiple composite fields structured as follows:-

| FIELD TYPE | DATA |
|---|---|
| 1 byte | N bytes |

Where ‘Field Type’ can be 0x00 for ‘OBJECT HANDLE’ and 0x01 for ‘ATTRIBUTE TAG/VALUE’ and the size of the ‘Data’ field depends on the Field Type. The available Field Types/Data are described in the following sections :-

### Field Type: Object Handle

This is the format of the ‘Object Handle’ field type and is always 3 bytes long:-

| 00 | OBJECT HANDLE |
|---|---|
| Always 0x00 | 2 bytes |

### Field Type: Attribute Tag/Value

This is the format of the ‘Attribute Tag/Value’ field type which is of variable size:-

| 01 | ATTR CODE | ATTR VALUE LEN | ATTR VALUE |
|---|---|---|---|
| Always 0x01 | 2 bytes | 2 bytes | N bytes |

The size of this field is ‘ATTR VALUE LEN’ + 5

### Example: Scan Report

A sample logical ‘scan report’ packet is as follows:-

```
0021 01 72B4 1234 01
   00 0001
   01 0A56 0004 12345678
   01 0990 0008 2112021216453378
```

Which is interpreted and expanded as follows-

```
SCAN REPORT handle=29364 personId=4660, reports=1
                  O:1 (0001)
                  A:2646 (0A56),<len=4> 12345678
                  A:2448 (0990),<len=8> 2112021216453378
```

**Sample code to interpret a ScanReport logical packet**

The following code shows how a logical ‘Scan Report’ packet could be separated into its constituent parts:-

```
void PrintScanReport(unsigned char *pRxPkt, unsigned nRxPktLen)
{
    char baMsg[512];
    uint16 nHandle;
    uint16 nPersonId;
    uint16 nObject;
    uint16 nAttrId;
    uint16 nAttrLen;
    uint8 *pSrc;
    char *pMsg;

    nHandle   = (pRxPkt[3]<<8)+pRxPkt[4];
    nPersonId = (pRxPkt[5]<<8)+pRxPkt[6];
    printf(baMsg,"SCAN REPORT handle=%d personId=%d, reports=%d",
                              nHandle,  nPersonId,    pRxPkt[7]);
    pSrc = &pRxPkt[8];
    while(pSrc < &pRxPkt[nRxPktLen])
    {
        switch(*pSrc)
        {
        case HDP_SCANREPORT_INFOTYPE_OBJECT: /* 0x00 */
            pSrc++;
            if( pSrc >= &pRxPkt[nRxPktLen] )
            {
                printf("INSUFFICIENT LENGTH -- ABORT display of msg");
                break;
            }
            nObject = (pSrc[0]<<8)+pSrc[1];
            printf("              O:%d (%04X)",nObject,nObject);
            pSrc+=2;
            break;
        case HDP_SCANREPORT_INFOTYPE_ATTRIBUTE: /* 0x01 */
            pMsg = baMsg;
            pSrc++;
            nAttrId = (pSrc[0]<<8)+pSrc[1];
            pSrc+=2;
            nAttrLen = (pSrc[0]<<8)+pSrc[1];
            pSrc+=2;
            if( &pSrc[nAttrLen] > &pRxPkt[nRxPktLen] )
            {
                printf("INSUFFICIENT LENGTH -- ABORT display of msg");
                break;
            }
            pMsg += sprintf(pMsg,"            A:%d (%04X),<len=%d> ",
                                                   nAttrId,
                                                         nAttrId,
                                                               nAttrLen);
            {
                uint16 nBlockLen = (nAttrLen>24)
                                      ? 24
                                      : nAttrLen;
                uint8 *pData = pSrc;
                while(nBlockLen--)
                {
                    pMsg += sprintf(pMsg,"%02X",*pData++);
                }
                if( nAttrLen > 24 )
                {
                    pMsg += sprintf(pMsg,"...");
                }
            }
            pSrc +=   nAttrLen;
            printf(baMsg);
            break;
        default:
            printf("OBJECT TYPE TAG unknown %d -- ABORT display of msg",*pSrc);
            return;
        }
    }
```

# Chapter 7
# Multipoint Application Examples

## BLOB Manager

BLOB stands for ‘Binary Long OBject’.

There are many Bluetooth related operations which require large strings to be submitted to the underlying Bluetooth stack. For example, friendly names, extended inquiry responses to name a few. These strings can be larger than the data packets allowed by the multipoint protocol defined in this specification.

The blog manager is basically a software entity in the module which enables these large objects to be uploaded into the module in small packets and have them accumulated in a single object.

The blob manager can be compile time configured to manage up to N objects and unless special firmware builds have been generated, this manual assumes that N is 2. Each blob is given a zero based indentifier. BlobId 0 is the first object etc.

A command packet called CMD_BLOBMANAGE exists to manage these blobs as required. This command takes 4 parameters. Parameter 1 is the subcommand ID which tells the blob manager what to do, parameter 2 is the blob ID, parameters 3 & 4 are 4 byte integer values used as arguments for the subcommand specified in parameter 1.

The response packet also contains 4 parameters in exactly the same fashion. Where parameters 1 and 2 are echoed from the command and parameters 3 & 4 depend on the subcommand.

Recall that this entity manages blobs and CMD_BLOBMANAGE is the command to act on them. To get data into the blobs requires the use of data packets with specific dedicated channel numbers. Channel numbers 0x98 hex to 0x9F hex are reserved for use with blobs 0 to 7 respectively.

If data is sent in a data packet with a channel number corresponding to a blob which does not exist, then the data is silently discarded.

Data packets sent to the same blob get appended to any existing data in that blob.

Please be warned that sending data to a blob reduces memory for other uses so it is highly recommended that the blob be cleared or used up as quickly as possible. The bluetooth chipset has very limited ram.

Once data is accumulated in a blob, CMD_BLOBMANAGE is used to perform various actions on that blob which is specified via parameter 1 described as ‘subcommand id’ above. Some of the actions possible are:-

CLEAR:This empties the blob identified by the blobId parameter 2

GETSIZE: This returns the size of the blob in bytes in parameter 3 of the response

COPYREAD: This sends a copy of all the data in the blob back over the UART in data channel (blobId+0x98)

HIDSET: This moves the data to the nonvolatile memory location where custom hid descriptors are stored. Many hid descriptors can be stored and each is identified by a 0 indexed identifier and in this case the hid id is specified in parameter 3 of the command

HIDGET:This appends the content of hid descriptor in nonvol storage identified by the hid id in parameter 1 into the blob identified by parameter 2.

See description of the command CMD_BLOB_MANAGE for all the actions possible.

## HID Connections

HID stands for ‘Human Interface Device’ and was originally described in detail in a specification published by the USB organization. The Bluetooth SIG has built on that idea, but uses wireless instead of usb as the transport mechanism.

The HID specifications are very dry and heavy tomes from a developers perspective which belies the user experience which is ‘it just works’ and ‘is simple’.

The objective of the HID functionality provided in the Laird Bluetooth module is to provide the same ‘it just works’ and ‘is simple’ concept, but for developers.

With this in mind it is encouraged that the developer view the module’s HID functionality as a black box and the only concepts to be aware and fully understand are input reports, output reports and how to create a hid descriptor.

The terminology for input/output is HID Host centric where ‘input’ means information flow from the HID Device to the HID Host and vice versa ‘output’ means information flow from HID Host to HID Device. USB developers will be familiar with this concept.

Input and Output are packets of information whose format and size are predefined in the HID Descriptor that totally describes the devices functionality. For example, the standard PC keyboard is defined by a HID Descriptor which specifies that when a key is pressed or unpressed an 8 byte INPUT packet shall be sent to the host and likewise, if the host wants to update one of the leds on the keyboard (for example the numlock led) then it shall send a 1 byre OUTPUT packet. How the bits in the INPUT and OUTPUT packets are to be interpreted are again specified in the HID Descriptor.

In a nutshell, when something happens at the device end, it informs the host via an INPUT packet, which is also called ‘HID Input Report’ and likewise the host sends information at any time using OUTPUT packets which are also called ‘HID Output Reports’.

This implies that a developer using HID supplied in the Laird module only needs to ask, “What is the current active Hid Device Desriptor’ and then from there decide how to generate and process the reports. A simple interface is supplied at the UART of the module to enable appropriate mapping of data into and out of INPUTand OUTPUT reports. The same interface also enables the developer to upload custom HID Descriptors into non-volatile memory of the module.

By default, if no HID Descriptor is uploaded, and the module is configured to expose a HID Device profile, then a HID Descriptor for a 104 key keyboard is exposed which means INPUT reports will be 8 bytes long and OUTPUT reports will be 1 byte long. In this case when a key press is to be conveyed to the host, an 8 byte data packet has to submitted to the module via the UART with data channel id 0xA0. Likewise any OUTPUT packets sent by the host will appear on data channel 0xA0. If a HID Host profile is active, then the INPUT packets will appear on data channel 0x90 and it will send OUTPUT reports as data on channel 0x90.

For the built-in HID Device keyboard descriptor it has been made even simpler to use if all you want is to send ascii characters in the range 0x00 to 0x7F inclusive. In that case all that is required is to send the ascii string in a data packet on channel 0x20. The data parser in the module will generate two INPUT reports for each ascii character. The first INPUT report will specify a key press and the second INPUT report will specify the unpress event.

Note::
If S reg 3 specified ONLY HID profile and S Reg 39 specifies the built in keyboard hid descriptor, then the device class for the device in S Reg 128 is overridden automatically.

## Sending INPUT Reports

Once a connection is established, a report is sent by a device end by sending an entire INPUT report in a **single** data packet with channel ID 0xA0.

For example, if the descriptor specifies a standard keyboard and if the ‘a’ key pressed then the following data is sent over the UART to to the module :-

0A **A0**    00 00 04 00 00 00 00 00

And to convey that the left shift has been also been pressed then the data is    :-

0A **A0**    02 00 04 00 00 00 00 00

## Getting OUPUT Reports from a Host

Once a connection is established, a report is sent by a host to the device by sending an entire OUPUT report in a **single** data packet with channel ID 0x90, for example

03 **90**    01

## Uploading a HID Descriptor into the Module

Uploading of HID descriptors, which can be large blocks of arbitrary binary data, is done using the blob manager. The blob manager is an entity in the module which allows for blocks of binary data to be received over the UART and accumulated in ‘blob’ objects, of which two are made available. The two blobs of data have identifiers 0 and 1 respectively. In addition, data channels 0x98 to 0x9F are dedicated to data transfer to/from those blobs. Where channel 0x98 is for blob 0, 0x99 is for blob 1. There is also a command called CMD_BLOBMANAGE which is used to perform various actions on the blobs. See definition of that command for more details, suffice to say that there are subcommands for clearing, getting size, saving to nonvol storage and getting from nonvol storage.

In the case of uploading a HID Descriptor, the blob commands to use are ‘clear’ and ‘save’.

For example, if the contrived thirteen byte HID Descriptor 05 01 09 06 A1 01 05 07 29 65 81 00 C0 is to be uploaded using blob 1 into nonvol location 1, (where this nonvol location reference is used in S Register 39) then the following packets are submitted to the module on the UART.

0E 00 2D 7F    00 01 00000000 00000000      //CMD_BLOBMANAGE --- clear blob 1

0B **99**    05 01 09 06 A1 01 05 07 29 65 81      //send data into blob 1

04 **99**    00 C0      //send more data into blob 1 which is appended to any existing data

0E 00 2D 7F    03 01 00000001 00000000      // CMD_BLOBMANAGE --- save blob 1 into nonvol storage id 1

## Specifying a Custom Hid Descriptor for Use

After a custom hid descriptor is uploaded into the module where a hid descriptor in the range 0..N will have been specified, the module can be configured to use that descriptor when HID Device profile is active by modifying S Register 39. Basically, take the 0 based hid ID, add 1 to it and store that value in that register.

## Specifying Service Record Name for Custom Hid

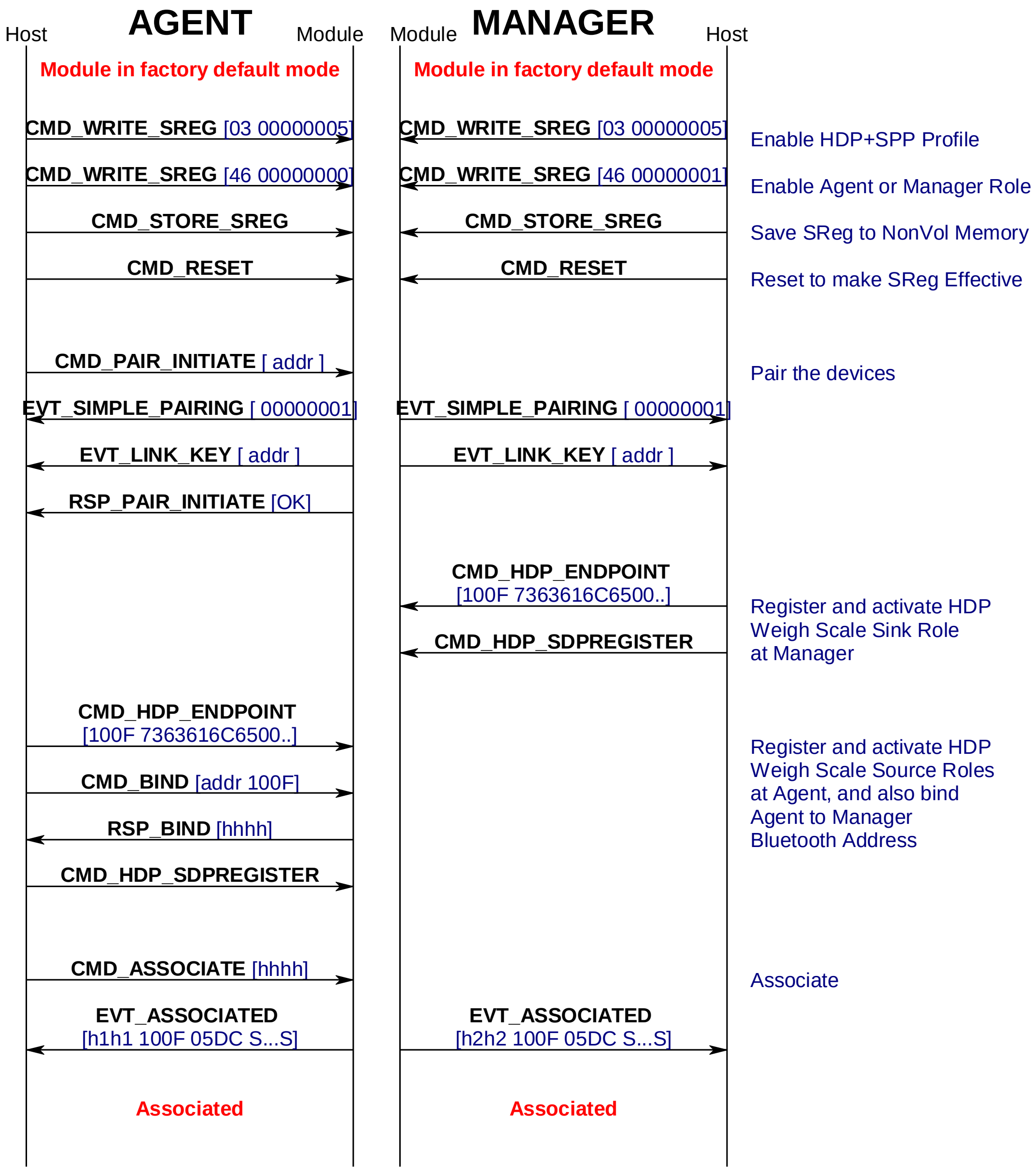
For a custom hid descriptor, the device can also register a custom service name if it has been saved using the blob manager. At any time the default service name "BTHIDCUSTOM” can be invoked by deleting the service name from non-voaltile memory. This can be done by writing an empty name via the blob manager.

## HDP Usage Message

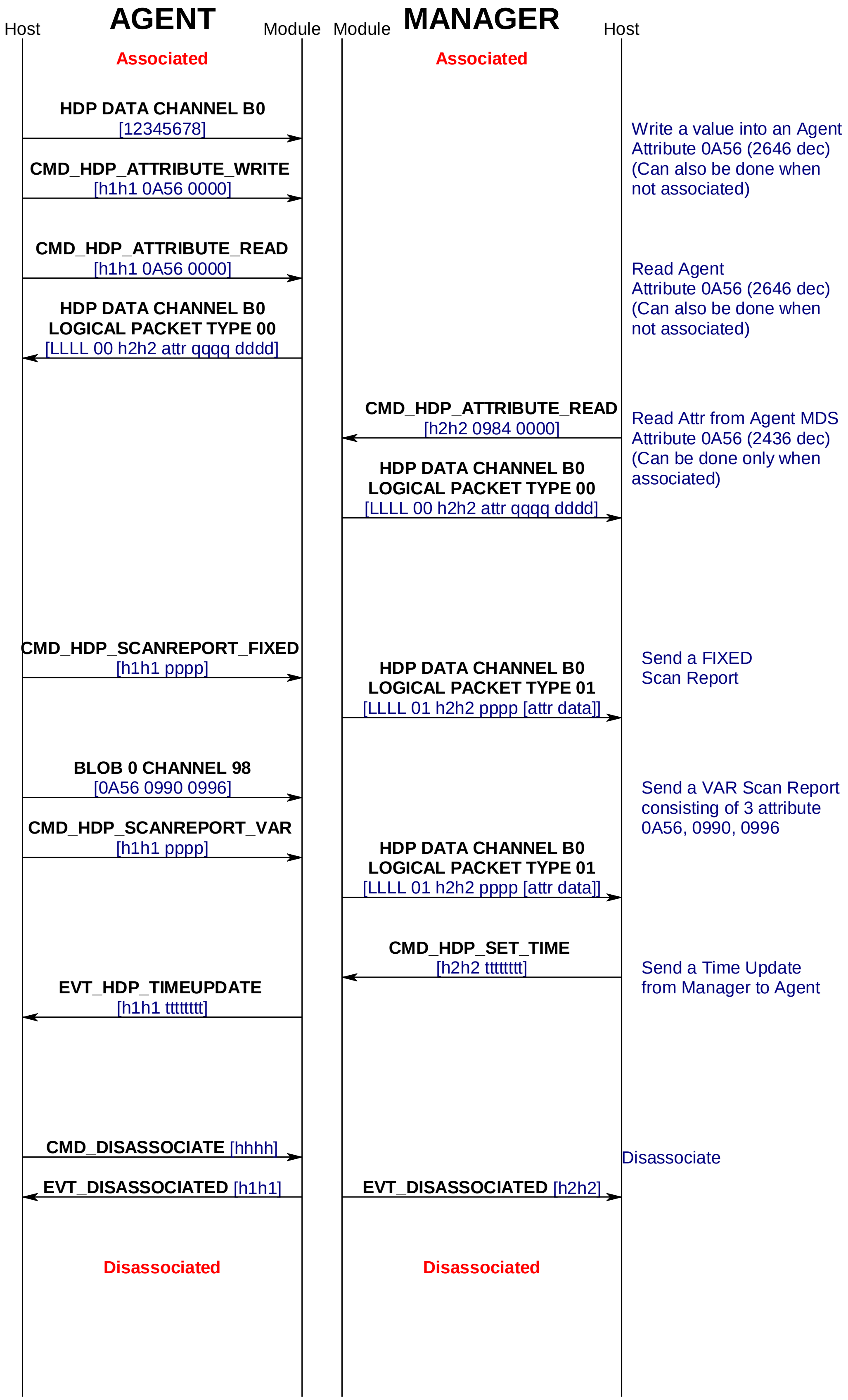
The module offers both HDP Agent and HDP Manager roles with IEEE Data Specialization functionality. HDP Manager functionality is provided mainly for prototyping and testing an Agent implementation and is not intended for eventual Continua Alliance certification.

Given two modules in factory default state, the following sections illustrate a typical usage session which consists of a pairing, an association, scan report, time update from manager and disassociation.

### Message Sequence Chart


Host
AGENT
Module
Module
MANAGER
Host
Module in factory default mode
Module in factory default mode
CMD_WRITE_SREG [03 00000005]
CMD_WRITE_SREG [03 00000005]
Enable HDP+SPP Profile
CMD_WRITE_SREG [46 00000000]
CMD_WRITE_SREG [46 00000001]
Enable Agent or Manager Role
CMD_STORE_SREG
CMD_STORE_SREG
Save SReg to NonVol Memory
CMD_RESET
CMD_RESET
Reset to make SReg Effective
CMD_PAIR_INITIATE [ addr ]
Pair the devices
EVT_SIMPLE_PAIRING [ 00000001]
EVT_SIMPLE_PAIRING [ 00000001]
EVT_LINK_KEY [ addr ]
EVT_LINK_KEY [ addr ]
RSP_PAIR_INITIATE [OK]
CMD_HDP_ENDPOINT [100F 7363616C6500..]
Register and activate HDP Weigh Scale Sink Role at Manager
CMD_HDP_SDPREGISTER
CMD_HDP_ENDPOINT [100F 7363616C6500..]
Register and activate HDP Weigh Scale Source Roles at Agent, and also bind Agent to Manager Bluetooth Address
CMD_BIND [addr 100F]
RSP_BIND [hhhh]
CMD_HDP_SDPREGISTER
CMD_ASSOCIATE [hhhh]
Associate
EVT_ASSOCIATED [h1h1 100F 05DC S...S]
EVT_ASSOCIATED [h2h2 100F 05DC S...S]
Associated
Associated

Continued on next page


Host
AGENT
Module
Module
MANAGER
Host
Associated
Associated
HDP DATA CHANNEL B0
[12345678]
CMD_HDP_ATTRIBUTE_WRITE
[h1h1 0A56 0000]
Write a value into an Agent Attribute 0A56 (2646 dec) (Can also be done when not associated)
CMD_HDP_ATTRIBUTE_READ
[h1h1 0A56 0000]
HDP DATA CHANNEL B0
LOGICAL PACKET TYPE 00
[LLLL 00 h2h2 attr qqqq dddd]
Read Agent Attribute 0A56 (2646 dec) (Can also be done when not associated)
CMD_HDP_ATTRIBUTE_READ
[h2h2 0984 0000]
HDP DATA CHANNEL B0
LOGICAL PACKET TYPE 00
[LLLL 00 h2h2 attr qqqq dddd]
Read Attr from Agent MDS Attribute 0A56 (2436 dec) (Can be done only when associated)
CMD_HDP_SCANREPORT_FIXED
[h1h1 pppp]
HDP DATA CHANNEL B0
LOGICAL PACKET TYPE 01
[LLLL 01 h2h2 pppp [attr data]]
Send a FIXED Scan Report
BLOB 0 CHANNEL 98
[0A56 0990 0996]
CMD_HDP_SCANREPORT_VAR
[h1h1 pppp]
HDP DATA CHANNEL B0
LOGICAL PACKET TYPE 01
[LLLL 01 h2h2 pppp [attr data]]
Send a VAR Scan Report consisting of 3 attribute 0A56, 0990, 0996
CMD_HDP_SET_TIME
[h2h2 tttttttt]
EVT_HDP_TIMEUPDATE
[h1h1 tttttttt]
Send a Time Update from Manager to Agent
CMD_DISASSOCIATE [hhhh]
EVT_DISASSOCIATED [h1h1]
EVT_DISASSOCIATED [h2h2]
Disassociate
Disassociated
Disassociated

## Agent UART Traffic for Chart

This section shows the UART traffic for a module operating as a HDP Weigh Scale agent communicating with a manager it is NOT a log of the uart traffic for the message sequence chart illustrated in the previous section.

```
*********************************************
*********************************************
<70 006.896 08 00 81 7F  00 00 00 0C
         EVT_STATUS
//State: RESET_GETADDR
>70 000.000 04 00 02 7F
         CMD_READ_BDADDR
<70 000.078 0B 00 02 7F 00  0016A4FEF000
         RSP_READ_BDADDR  (MPSTATUS_OK)
>70 000.015 05 00 17 7F  00
         CMD_INFORMATION
//State: RESET_GETVER
<70 000.078 0E 00 17 7F 00  00 8100001300790673
         RSP_INFORMATION  (MPSTATUS_OK)
<70 001.810 08 00 81 7F  00 01 01 0C
         EVT_STATUS
>70 057.018 09 00 04 7F  03 00000005
         CMD_WRITE_SREG
<70 000.109 0A 00 04 7F 00  03 00000005
         RSP_WRITE_SREG  (MPSTATUS_OK)
>70 046.192 09 00 04 7F  46 00000000
         CMD_WRITE_SREG
<70 000.109 0A 00 04 7F 00  46 00000000
         RSP_WRITE_SREG  (MPSTATUS_OK)
>70 004.790 04 00 05 7F
         CMD_STORE_SREG
<70 000.109 05 00 05 7F 00
         RSP_STORE_SREG  (MPSTATUS_OK)
>70 024.523 07 00 29 7F 000000
         CMD_RESET
<70 001.918 08 00 81 7F  00 01 01 0C
         EVT_STATUS
>70 775.201 1C 00 10 7F  10 0016A4FEF001 3132333400000000000000000000000000
         CMD_PAIR_INITIATE
<70 007.300 0F 00 95 7F  0016A4FEF001 00 00000001
         EVT_SIMPLE_PAIRING
<70 000.156 0B 00 89 7F  0016A4FEF001 00
         EVT_LINK_KEY
<70 001.014 05 00 10 7F 00
         RSP_PAIR_INITIATE  (MPSTATUS_OK)
>70 032.246 16 00 2E 7F  100F 7363616C650000000000000000000000
         CMD_HDP_ENDPOINT
<70 000.109 05 00 2E 7F 00
         RSP_HDP_ENDPOINT  (MPSTATUS_OK)
>70 009.298 0E 00 32 7F  0016A4FEF001 100F 0000
         CMD_HDP_BIND
<70 000.125 07 00 32 7F 00  B538
         RSP_HDP_BIND  (MPSTATUS_OK)
>70 026.037 04 00 2F 7F
         CMD_HDP_SDPREGISTER
<70 000.125 05 00 2F 7F 00
         RSP_HDP_SDPREGISTER  (MPSTATUS_OK)
>70 011.918 06 00 31 7F  B538
         CMD_HDP_ASSOCIATE
<70 000.109 05 00 31 7F 00
         RSP_HDP_ASSOCIATE  (MPSTATUS_OK)
<70 002.013 08 00 81 7F  00 00 01 0C
         EVT_STATUS
<70 003.775 13 00 97 7F  00 B538 100F 05DC 4C414952444D4752
         EVT_HDP_ASSOCIATED
>70 029.235 06 B0 12345678
>70 018.127 0B 00 36 7F  B538 0A56 0000 CD
         CMD_HDP_ATTRIBUTE_WRITE
<70 000.109 08 00 36 7F 00  CD 0004
         RSP_HDP_ATTRIBUTE_WRITE  (MPSTATUS_OK)
>70 013.697 0B 00 35 7F  B538 0A56 0000 AB
         CMD_HDP_ATTRIBUTE_READ
<70 000.109 0F B0 000D00B5380A56000012345678
//HDP Channel : ATTRIBUTE for handle=46392 Attr=2646  QualifierId=0}    VALUE = 12345678
<70 000.016 08 00 35 7F 00  AB 0004
         RSP_HDP_ATTRIBUTE_READ  (MPSTATUS_OK)
>70 042.900 09 00 33 7F  B538 1234 AB
         CMD_HDP_SCANREPORT_FIXED
<70 000.421 08 00 33 7F 00  B538 AB
         RSP_HDP_SCANREPORT_FIXED  (MPSTATUS_OK)
>70 021.185 08 B0 0A5609900996
>70 014.571 0A 00 34 7F  B538 1234 CD 00
         CMD_HDP_SCANREPORT_VAR
<70 000.109 08 00 34 7F 8F  B538 CD
         RSP_HDP_SCANREPORT_VAR  (MPSTATUS_ATTRLIST_INVALID)
>70 035.365 0A 98 34 7F  0A56 0990 09 96
```

```
        CMD_HDP_SCANREPORT_VAR
>70 008.752 0A 00 34 7F  0A56 0990 09 96
        CMD_HDP_SCANREPORT_VAR
<70 000.125 08 00 34 7F 8B  0A56 09
        RSP_HDP_SCANREPORT_VAR  (MPSTATUS_INVALID_BLOBID)
>70 019.906 08 98 0A5609900996
>70 012.823 0A 00 34 7F  B538 1234 CD 00
        CMD_HDP_SCANREPORT_VAR
<70 000.296 08 00 34 7F 00  B538 CD
        RSP_HDP_SCANREPORT_VAR  (MPSTATUS_OK)
<70 043.166 0E 00 98 7F  B538 140B020C102D214E
    EVT_HDP_TIMEUPDATE
>70 008.486 06 00 30 7F  B538
        CMD_HDP_DISASSOCIATE
<70 000.125 05 00 30 7F 00
        RSP_HDP_DISASSOCIATE  (MPSTATUS_OK)
<70 000.125 07 00 96 7F  00 B538
        EVT_HDP_DISASSOCIATED
```

## Manager UART Traffic for Chart

This section shows the UART traffic for a module operating as a HDP manager communicating with a Weigh Scale agent.

```
**********************************************
**********************************************
**********************************************
<71 003.900 08 00 81 7F  00 00 00 0C
         EVT_STATUS
//State: RESET_GETADDR
>71 000.000 04 00 02 7F
         CMD_READ_BDADDR
<71 000.094 0B 00 02 7F 00  0016A4FEF001
         RSP_READ_BDADDR  (MPSTATUS_OK)
>71 000.015 05 00 17 7F  00
         CMD_INFORMATION
//State: RESET_GETVER
<71 000.078 0E 00 17 7F 00  00 8100001300790673
         RSP_INFORMATION  (MPSTATUS_OK)
<71 001.810 08 00 81 7F  00 01 01 0C
         EVT_STATUS
>71 062.806 09 00 04 7F  03 00000005
         CMD_WRITE_SREG
<71 000.109 0A 00 04 7F 00  03 00000005
         RSP_WRITE_SREG  (MPSTATUS_OK)
>71 022.449 09 00 04 7F  46 00000001
         CMD_WRITE_SREG
<71 000.109 0A 00 04 7F 00  46 00000001
         RSP_WRITE_SREG  (MPSTATUS_OK)
>71 016.209 04 00 05 7F
         CMD_STORE_SREG
<71 000.109 05 00 05 7F 00
         RSP_STORE_SREG  (MPSTATUS_OK)
>71 011.310 07 00 29 7F 000000
         CMD_RESET
<71 000.717 08 00 81 7F  00 00 00 0C
         EVT_STATUS
<71 001.950 08 00 81 7F  00 01 01 0C
         EVT_STATUS
<71 787.103 0F 00 95 7F  0016A4FEF000 00 00000001
         EVT_SIMPLE_PAIRING
<71 000.188 0B 00 89 7F  0016A4FEF000 00
         EVT_LINK_KEY
>71 018.564 16 00 2E 7F  100F 7363616C650000000000000000000000
         CMD_HDP_ENDPOINT
<71 000.125 05 00 2E 7F 00
         RSP_HDP_ENDPOINT  (MPSTATUS_OK)
>71 005.413 04 00 2F 7F
         CMD_HDP_SDPREGISTER
<71 000.125 05 00 2F 7F 00
         RSP_HDP_SDPREGISTER  (MPSTATUS_OK)
<71 067.751 08 00 81 7F  00 00 01 0C
         EVT_STATUS
<71 002.028 13 00 97 7F  01 72B4 100F 05DC 0016A4FEF000B539
         EVT_HDP_ASSOCIATED
>71 093.210 0B 00 35 7F  72B4 0984 0000 AB
         CMD_HDP_ATTRIBUTE_READ
<71 000.110 15 B0 00130072B40984000000080016A4FEF000B539
//HDP Channel : ATTRIBUTE for handle=29364 Attr=2436  QualifierId=0}    VALUE = 00080016A4FEF000B539
<71 000.000 08 00 35 7F 00  AB 000A
         RSP_HDP_ATTRIBUTE_READ  (MPSTATUS_OK)
<71 011.419 23 B0 00210172B4123401000001010A5600047856341201099000080000000000000000
//HDP Channel : SCAN REPORT handle=29364 personId=4660, reports=1
//              O:1 (0001)
//                A:2646 (0A56),<len=4> 78563412
//                A:2448 (0990),<len=8> 0000000000000000
<71 113.210 47 B0 00450172B4123401000001010A56000478563412010990000800000000000000000109960002
06C3010A5600047856341201099000080000000000000000010996000206C3
//HDP Channel : SCAN REPORT handle=29364 personId=4660, reports=1
//              O:1 (0001)
//                A:2646 (0A56),<len=4> 78563412
//                A:2448 (0990),<len=8> 0000000000000000
//                A:2454 (0996),<len=2> 06C3
//                A:2646 (0A56),<len=4> 78563412
//                A:2448 (0990),<len=8> 0000000000000000
//                A:2454 (0996),<len=2> 06C3
>71 043.056 0E 00 37 7F  72B4 140B020C102D214E
         CMD_HDP_SET_TIME
<71 000.109 05 00 37 7F 00
         RSP_HDP_SET_TIME  (MPSTATUS_OK)
<71 008.721 07 00 96 7F  01 72B4
         EVT_HDP_DISASSOCIATED
<71 002.059 08 00 81 7F  00 01 01 0C
         EVT_STATUS
```

## Sniff Mode Explained

Bluetooth connections are master/slave in nature. A master sends packets and a slave has to acknowledge that packet in the next timeslot. Timeslots in Bluetooth are 625 microseconds wide. This implies that a master will always know when packets will be sent and received, which further means it is able to optimize power usage by switching on power hungry circuitry only when needed.

A slave on the other hand does NOT have prior knowledge of when a packet will be received and has to assume that a packet will be received from a master on every receive slot. This means that it has to leave its receiving circuitry switched on for most of the receive slot duration. The result of this is high power consumption where a slave with no data transmission still consumes around 31mA whereas a master consumes only 6mA.

This problem was identified very early in the evolution of Bluetooth (especially since headsets spend all their time as a slave in a Bluetooth connection) and it was solved by having a mode called Sniff, with appropriate lower layer negotiating protocol.

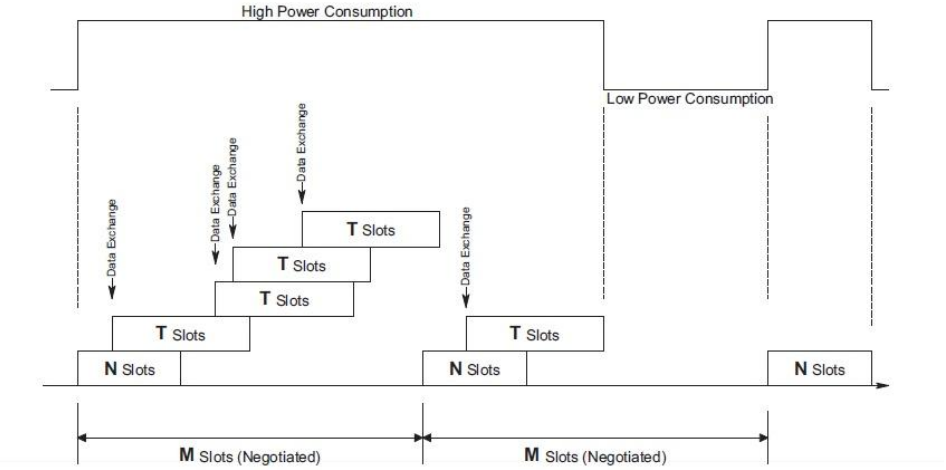
Sniff mode during connection is basically an agreement between the slave and its master that null packets will only be exchanged for N timeslots every M slots. The slave can then assume that it will never be contacted during N-M slots, and so can switch its power hungry circuitry off. The specification goes further by also specifying a third parameter called ‘timeout’ (T) which specifies ‘extra’ timeslots that the slave will agree to listen for, after receiving a valid data packet.

Put another way, if a data packet is received by the slave, then it knows that it MUST carry on listening for at least T more slots. If within that T slot time period another data packet is received, then the timer is restarted. This mechanism ensures low power consumption when there is no data transfer – at the expense of latency. When there is a lot of data to be transferred, it acts as if sniff mode were not enabled.

It is stated above that during sniff mode, a slave listens for N slots every M slots. The Bluetooth specification states that a master can have up to 7 slaves attached to it with all slaves having requested varying sniff parameters. It may therefore be impossible to guarantee that each slave gets the M parameter it requested. In light of this, the protocol for enabling sniff mode specifies that a requesting peer specify the M parameter as a minimum and maximum value. This will allow the master to interleave the sniff modes for all slaves attached.

For this reason, the sniff parameters are specified in the bluetooth module via four S registers. SRegister 73 (561 in AT mode) is used to specify ‘N’, SRegister 74 (562 in AT mode) is used to specify ‘T’ and SRegisters 75/76 (563/564 in AT mode) are used to specify minimum ‘M’ and maximum ‘M’ respectively. Although the specification defines these parameters in terms of timeslots, the S register values have to be specified in units of milliseconds and the firmware does the necessary translation to timeslots.

The relationship between M,N and T and power consumption is illustrated in the diagram below.



Power consumption when sniff mode is active

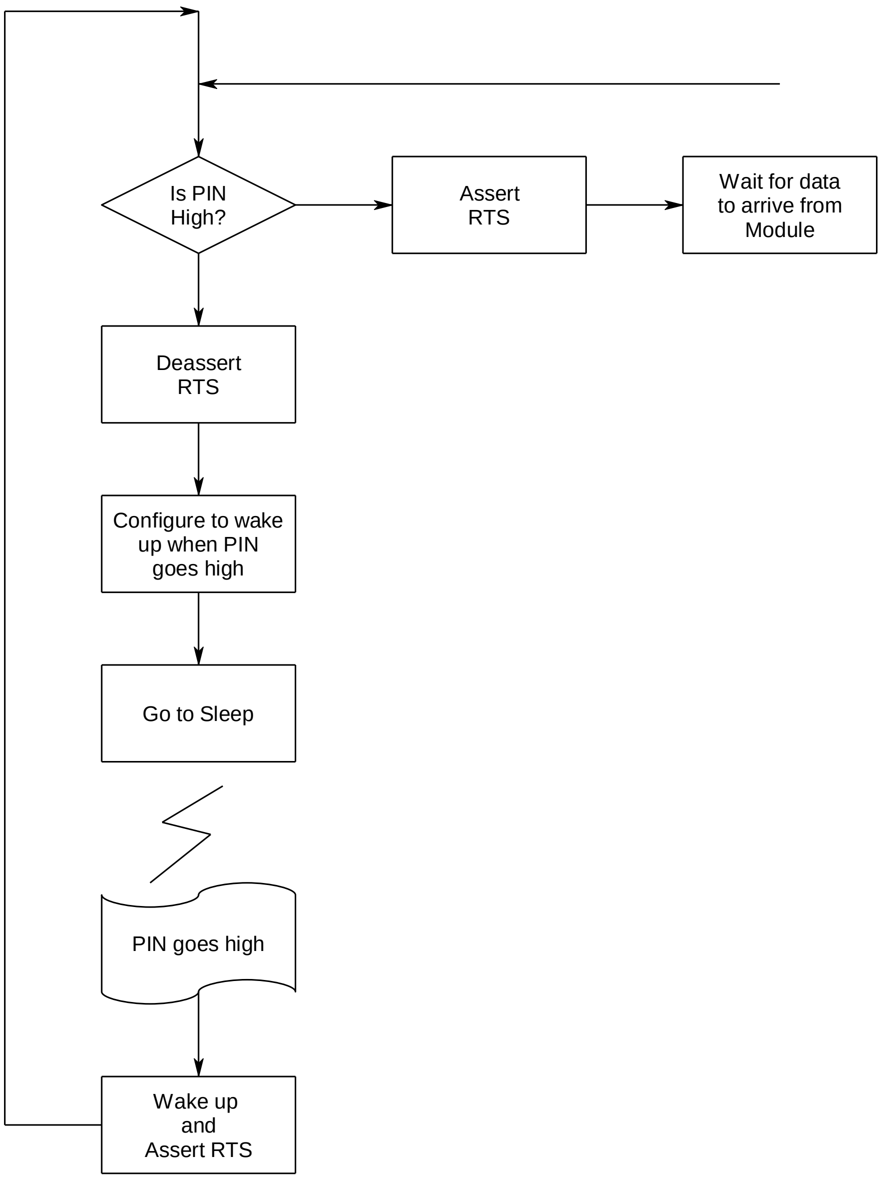
## UART Host Power Saving Facility

There are circumstances where a cpu driving the module consumes a lot of power and some means is necessary to reduce that power consumption WHILE the module is in a bluetooth connection.

To facilitate that, the module has many GPIO pins and using S Registers 50 to 65, one (and only one) GPIO pin can be configured with the value 13 so that it is configured as an output.

The state of the pin will be 0 when the module's UART transmit buffer is empty and 1 when there is at least one byte waiting to be transmitted to the UART host.

Hence a workable power saving strategy by the CPU would be as illustrated below.



## Out of Band (OOB) Pairing

When two device pair using the legacy procedure or the simple secure pairing method, the end result is that they both end up with the SAME 16 byte random key which is then subsequently used to authenticate and encrypt subsequent connections.

This means the list of pairing, kept in each device, as a minimum, needs to store the peer bluetooth address along with the 16 byte link key.

The Bluetooth specifications do not mandate that this link key shall only be generated/exchanged over the bluetooth radio, and in fact mention ‘Out-Of-Band’ pairing as a valid means of expediting the pairing of two devices.

The specification does not describe how the out-of-band pairing occurs.

Whatever OOB means is chosen, it does imply that some externally generated key has to be added to the trusted device database in the device.

The module caters for this link key addition using the CMD_TRUSTED_DB_ADD command when in the multipoint protocol mode and the AT+KY command when in the AT protocol mode.

## Throughput Analysis

There are various factors that affect overall data throughput and they are:-

1. The baudrate at the UART determines the maximum throughput and will be a theoretical maximum of 80% of the baudrate if none parity and 1 stop bits is used. That theoretical maximum reduces to around 67% if parity is enabled along with 2 stopbits.
2. The radio utilization, in the sense that at any time, up to three non-transient operations could be active. The radio could be servicing on-going connections, it could be scanning for inquiries and it could be scanning for incoming connections. For the latter two, the scanning operation has a duty cycle and the worst case of 100% will have a major impact on the throughput as the radio is being time shared between the connections and the scanning operations
3. The rf connection quality. If the quality is bad and there are many retries of packets, then the throughput can drop to close to 0 before the connection is automatically dropped. For Basic Rate connection packets the best throughput is limited to around 600kbps in asymmetric data transfer falling to around 400kbps for symmetric transfers when Base Rate RF packets are used. This can triple when EDR packets are used.
4. The size of the RFCOMM frame, which according to the BT spec can be a value between 23 and 32767. The bigger the value the better, but the incremental gain around 1000 and above is negligible for embedded bluetooth stack with limited ram. This value is set via S Register 11 in multipoint mode and 9011 in AT mode
5. In the multipoint protocol which is packet based, the size of the MP packet payload has an impact and in fact the packets should be as large as possible, and yet the MP protocol limits the maximum payload to 253 bytes due to the length field of the packet being only a single byte.

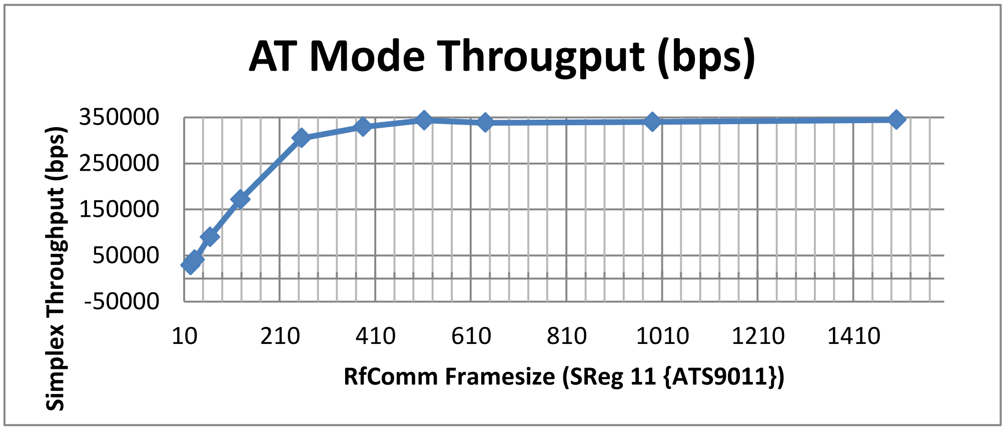
The charts that follow, where actual throughput is plotted against the rfcomm frame size, show that in multipoint mode the packet structure and scanning for inquiries and paging have a significant impact on the throughput.

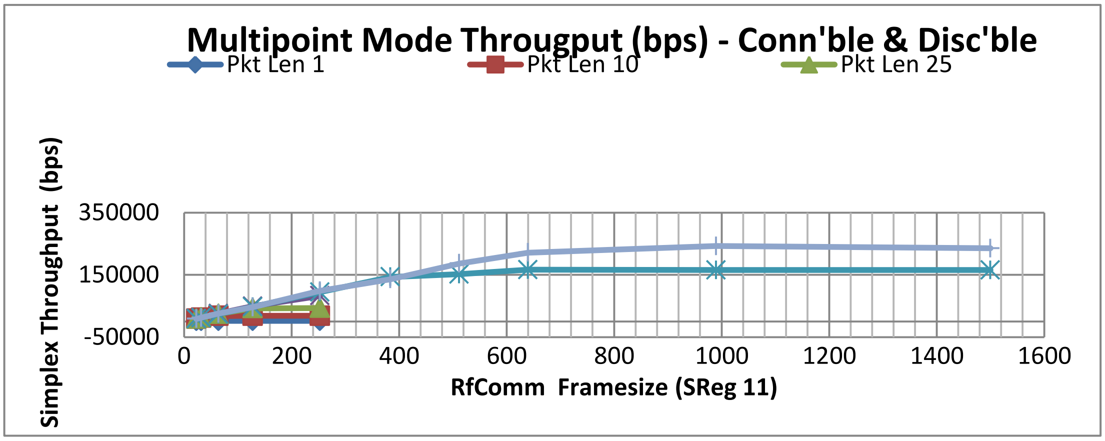
With regards to MP mode, the UART host should optimize performance by sending data to transmit in as large packets as possible and completely disabling all scanning operations by setting S Registers 4 and 5 to 0.

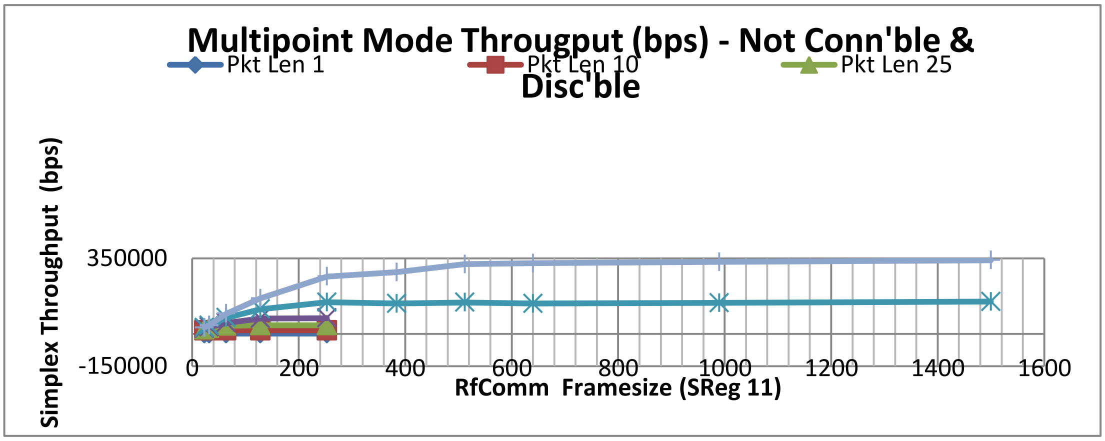
It is entirely possible for the host to bombard the module with the worst case scenario of 3 byte packets with just 1 data byte payloads. In this case if too many of these packets are sent and the framesize is large (say 64 and above) it is entirely possible for the module to lose the connection by resetting. This happens because the module panics when it runs out of memory. On the rare occasion that this happens, it is possible to mitigate this issue by increasing the value of S register 81. By default this value is set to 30%.

Testing by Laird has shown that with a framesize of larger than 64 and sending a storm of 3 byte packets (with 1 byte payload) and the default value of 30%, it is possible to panic the module into a reset. Testing with a value of 50% in S Reg 81 solves the problem. But increasing the value of S Register 81 has an impact on how many simultaneous SPP connections can be sustained.

Basically, users will have to fine tune S Registers 7,8,9,10,11,81 and MP packet sizes to ensure desired throughput operation.


AT Mode Througput (bps)
Simplex Throughput (bps)
350000
250000
150000
50000
-50000
10
210
410
610
810
1010
1210
1410
RfComm Framesize (SReg 11 {ATS9011})


Multipoint Mode Througput (bps) - Conn'ble & Disc'ble
Pkt Len 1
Pkt Len 10
Pkt Len 25
Simplex Throughput (bps)
350000
150000
-50000
0
200
400
600
800
1000
1200
1400
1600
RfComm Framesize (SReg 11)


Multipoint Mode Througput (bps) - Not Conn'ble & Disc'ble
Pkt Len 1
Pkt Len 10
Pkt Len 25
Simplex Throughput (bps)
350000
-150000
0
200
400
600
800
1000
1200
1400
1600
RfComm Framesize (SReg 11)

## UART Protocol Selection & Indication Via GPIO

S register 255 is used to select either MP(1) or AT(2) protocol mode for communications over the UART.

If S Register 255 is set to 0, then it implies that a GPIO will be used to select the protocol such that 0 will be used to set AT mode and 1 for MP mode.

To configure a particular GPIO pin for this functionality, set the appropriate S Reg (in range 50 to 65) to a value of 14. Only the first S Register in the range 50 to 65 is used, any further S Registers with the value 14 will be ignored.

If in addition, at least one S register in the range 50 to 65 is set to a value of 15, then on power up, that pin will be configured as an output and will be set to 0 if AT protocol is active and 1 if MP is active.

If S Register 255 is 0 and no GPIO is configured for this functionality, then the protocol will default to MP.

## Firmware Upgrade via UART

The module has the capability of upgrading the firmware via the UART port using a Windows PC based utility supplied by Laird.

Firmware upgrades over the air is not planned as this is not inherently supported by the chipset vendor.

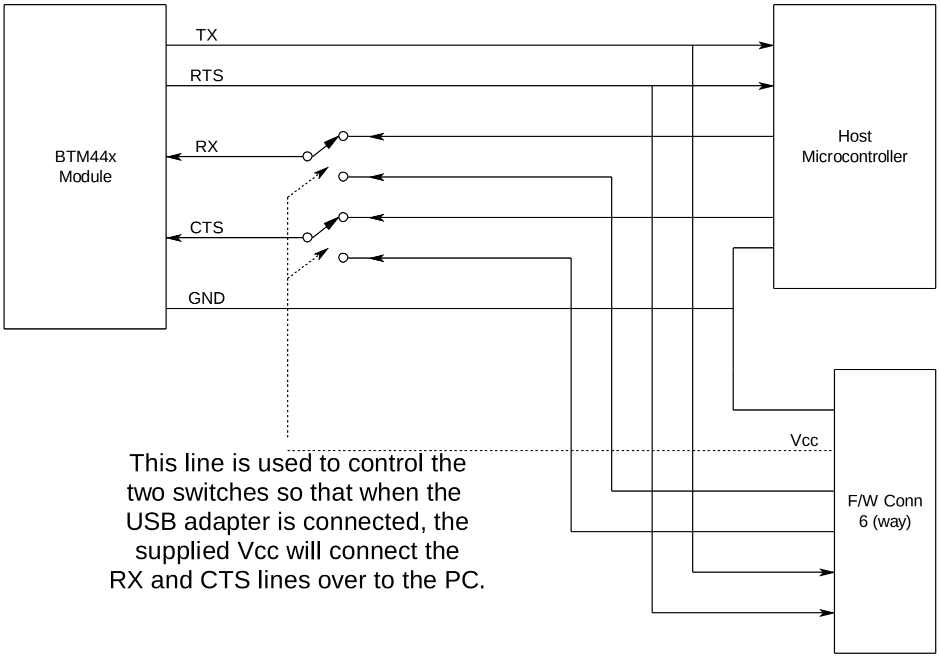
The upgrade process requires a direct connection to RX,TX,CTS and RTS lines of the module, via appropriate RS232 level conversion, to a built-in serial port on the Windows PC.

The new firmware will be deployed in a .dfu file as and when new firmware is available.

If the user requires the ability to upgrade the firmware when their product is in the field, then provision must be made so that the RX,TX,CTS and RTS lines are exposed to the 'outside' world. This will be complicated if, as in most usage cases, the BT module is driven by a host microcontroller in the user's end product. In that case, the module's RX and CTS input lines will be driven by the host microcontroller and hence cannot also be driven by a Windows PC unless those two lines are gated appropriately.

One solution would be to incorporate the hardware logic illustrated below and use a USB to Serial adapter as per http://www.ftdichip.com/Products/Cables/USBTTLSerial.htm which does not require RS232 levels.

Note that this solution **should** work in theory and Laird does not warrant that it will work given it has not been implemented and tested. The purpose of the suggestion is to make the user evaluate the convenience arising from it and variations thereof.



This line is used to control the two switches so that when the USB adapter is connected, the supplied Vcc will connect the RX and CTS lines over to the PC.

## UART Interface

UART_TX, UART_RX, UART_RTS and UART_CTS on the module form a conventional asynchronous serial data port. The interface is designed to operate correctly when connected to other UART devices such as the 16550A. The signaling levels are nominal 0V and 3.3V and are inverted with respect to the signaling on an RS232 cable. The interface is programmable over a variety of bit rates; no, even or odd parity; stop bit and mandatory hardware cts/rts flow control.

The default condition on power-up is pre-assigned in the external Flash. The mandatory two-way hardware flow control is implemented by UART_RTS and UART_CTS. UART_RTS is an output and is active low. UART_CTS is an input and is active low. These signals operate according to normal industry convention.

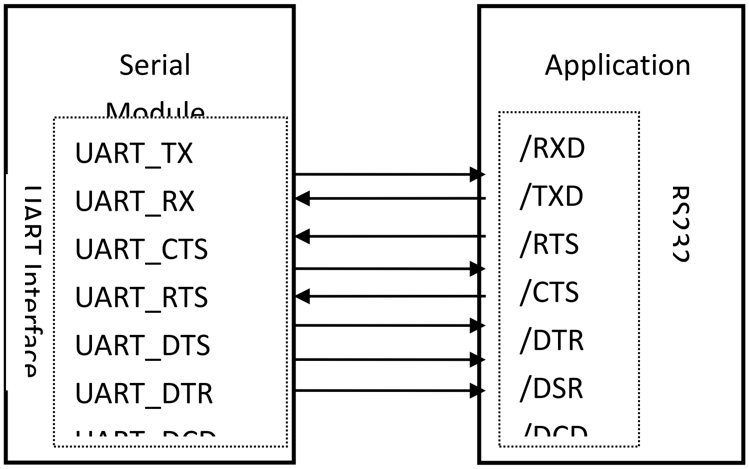
UART_DSR is an active low input. It should be connected to DTR output of the host. Depending on the value of SReg507, when in AT mode, this pin should be asserted by the host to ensure connection is maintained. A deassert is taken to mean that the connection should be dropped, or an online command mode is being requested.

The module communicates with the customer application using the following signals:

RS-232

Port /TXD @ application sends data to the module's UART_RX signal line

Port /RXD @ application receives data from the module's UART_TX signal line



Note:
UART_RTS/CTS handshaking is mandatory because the underlying bluetooth stack in the module uses buffer levels to regulate the data throughput and will deassert the output UART_RTS line when buffer space gets low. When this happens the host SHALL stop sending more data to the module. Failure to comply will result in buffer overflows and will result in unpredictable behavior and may result in a reset triggered via the watchdog circuitry within the module.

## The HCOMMAND & EVENT Values

The following is a listing of a snapshot of the file BMHOSTPROTOCOL.H at the time of writing this document. Laird does NOT guarantee that this listing will be kept up to date.

For development purposes, please request the latest version of the appropriate ‘C’ header file.

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
//The following are COMMAND (octet 2) values in command/response packets
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
#define CMD_NO_OPERATION                                0x01
#define CMD_READ_BDADDR                                  0x02
#define CMD_READ_SREG                                   0x03
#define CMD_WRITE_SREG                                  0x04
. . . .
. . . .
. . . .
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
//The following are EVENT (octet 2) values in event packets
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
#define EVT_STATUS                                    0x81
#define EVT_INVALID_PKTSIZE                            0x82
#define EVT_UNKNOWN_COMMAND                                0x83
#define EVT_INQUIRY_RESULT                              0x84
#define EVT_MODEM_STATUS                                 0x85
. . . .
. . . .
. . . .
```

### STATUS Values

The following is a listing of a snapshot of the file MPSTATUS.H at the time of writing this document. Laird does NOT guarantee that this listing will be kept up to date.

For development purposes, please request the latest version of the appropriate ‘C’ header file.

```
#define MPSTATUS_OK                                    0x00

#define MPSTATUS_ILLEGAL_COMMAND                         0x01
#define MPSTATUS_NO_CONNECTION                           0x02
#define MPSTATUS_HARDWARE_FAIL                          0x03
#define MPSTATUS_PAGE_TIMEOUT                           0x04
. . . .
. . . .
. . . .
```

# Chapter 8
# AT Application Examples

## Connection Management

Commands ATD, ATA, ATH, AT+BTP,AT+BTG are all connection related and will be discussed generically in this section.

On connection, depending on the value of S Register 531, the module will enter data pass through mode (S Reg 531 = 0) or will remain in command mode ( S Reg 531 > 0).

In pass through mode any data received from the host is passed to the transmit buffer of the rf connection in the case of SPP (uuid=1101) and for all other profiles, it will depend on whether a 'canned' mode has been provided. Data coming from the remote are sent out to the host transparently – even in canned mode.

A 'canned' mode, which exists for HID Device profile, is where the incoming character from the host is translated into appropriate multibyte packets that the peer is expecting. For example, with HID standard keyboard profile where each key press results in an 8 byte HID INPUT report to the host, the ascii character is appropriately expanded into the relevant 8 bytes denoting that the key has been presses and then immediately another 8 byte report to denote that the same key has been unpressed.

If a canned mode is not available for a profile, then module will not be allowed to get into pass-through mode.

In command and online mode, the command ATX"data" is used to force the module to send data and conversely any incoming data is presented to the host in an RX"data" asynchronous response.

## Incoming Connections

The module can be configured using the AT+BTP or AT+BTG command so that it will scan for incoming connections from other Bluetooth devices. It can also be configured via S Register 512 to be in this mode by default on power up.

When the lower layers detect an incoming call, a **RING 123456789012** string is sent to the host every second. The command ATA is used to accept the connection and ATH to reject it.

On connection, if the S0 Register is >=0 and S504=0 then confirmation to the host is in the form:-

CONNECT <bd_addr>,<uuid>,<

Where '<uuid>' is the uuid of the profile that accepted the connection.

## Dropping Connections

In a conventional telephony modem, a call is normally terminated by first sending a +++ escape sequence enveloped by an escape sequence guard time (of the order of 100 to 1000 milliseconds) to enter local command + connected mode and then the ATH command to force a disconnection.

The Laird modules provide a couple of ways of dropping a connection. One method is similar to the above, but instead a ^^^ character sequence is used. This eliminates ambiguity when a data call is in progress via a mobile phone which was established using the mobile phone's Bluetooth AT modem. The second method involves the host deasserting the DTR modem control line (DSR modem status line from the module's viewpoint) for longer than 500 milliseconds.

The escape sequence to force the module from pass-through mode and into command is as follows:-

<Guard time><Esc Chr><Guard time><Esc Chr><Guard time><Esc Chr><Guard time>

where <Guard time> is 100 milliseconds.

The four guard time means that even when a file transfer is occurring and it happens to be full of <Esc Chr> characters then it is not going to drop into command mode because, when transferring a file it is going to happen as fast as possible and so the inter character gap is going to be significantly shorter than the <Guard time>.

The <Esc Chr> character can be changed via the S2 register.

## Profiles

This section describes all the profiles that the module is capable of making and accepting connection when in AT mode.

## Serial Port Profile (SPP)

UUID : 1101

S Register 9003 bit 0 must be set to make this profile active.

Outgoing connections are initiated using the command: "**ATD<bd_addr>**"

Incoming connections will result in at least one "RING <bd_addr>" response to the host. If S Register 0 is a non-zero value then after the appropriate number of RING responses the connection will be automatically accepted and a "**CONNECT <bd_addr>,1101,<**" response will be sent to the host. If S Register 0 is 0, then the incoming connection is accepted by the host using the command ATA or rejected using ATH.

On connection, depending on the value of S Register 531, the module will enter pass through mode (data transparently exchanged between UART and air-side) or in command+online mode. In the latter, data is sent to the peer using the "ATX<string>" command and any data from the peer is either dumped silently (S531=1) or send to the host in an "RX<string>" asynchronous response (S531>1).

When in pass-through mode, the escape sequence ^^^ is used to put the module into command and online mode so that disconnection can be initiated. Or a disconnection can be initiated by deasserting the DSR input line of the module for more than 500 milliseconds.

On disconnection a "NO CARRIER" async response is sent to the host.

## HID Device Profile (HID)

UUID : 1124

S Register 9003 bit 1 must be set to make this profile active. In addition S register 9039 must be set to 0 and above to enable a DEVICE HID profile and a negative value to enable a HOST HID profile.

Outgoing connections are initiated using the command: "**ATD<bd_addr>,1124**"

Incoming connections will be automatically accepted and a "**CONNECT <bd_addr>,1124,<**" will be sent to the host.

With the HID Device profile a built in standard keyboard HID descriptor is supplied along with a canned mode of operation which enables a legacy device generating ascii characters to be presented to a host as a compliant hid keyboard.

In canned mode each ascii character (ascii characters 128 and above are silently discarded) results in two INPUT reports to the host. The first being a corresponding key press and the second being a corresponding key unpress. When the host sends the 1 byte OUTPUT report it is sent to the host as-is.

In non-canned mode (S Reg 531 > 0) the host has to send the raw 8 byte INPUT reports in the ATX<String> command and conversely any OUTPUT reports from the host are sent to the host in RX<string> asynchronous responses.

Disconnections from the module end are initiated via DSR deassertion. If however the module is in non-canned mode (S Register 531 > 0) then it is also possible to initiate a disconnection using the ATH command.

On disconnection a "NO CARRIER" async response is sent to the host.

## HID Descriptors

Hid devices present their capabilities to a host in a HID descriptor which is basically a block of octets which describe the device's capability and more importantly how events are conveyed back and forth. This concept was originally developed by the USB organisation and has been adopted by the Bluetooth SIG virtually intact.

The HID descriptor contains information about INPUT and OUTPUT reports. They are both just blocks of octets described to contain various bit fields describing the event that needs to be conveyed to the peer.

Hence, at the end of the day, if a HID implementation were to be viewed as a communications black box between a device and host, then it could be viewed as the device generating an INPUT report consisting of X bytes which is presented to the host and conversely an OUTPUT report consisting of Y bytes is sent by the host to the device.

In this module's HID implementation, the module does not care about the content of those INPUT and OUTPUT reports.

An INPUT report is presented to the module by the UART host in a ATX<string> which is then de-escaped and sent as a single atomic packet to the remote host. Similarly each OUTPUT package arrives atomically in a single packet from the remote host which is then sent to the UART host in a single RX<string> message.

It has been mentioned above that by default a standard keyboard hid descriptor is built into the firmware and the default value of S Register 9039 makes the module connectable via a HID Device profile.

It is possible to download up to two custom HID device descriptors and be stored in the module's non-volatile memory. These custom HID device descriptors are then identified via a number in the range 0 to N. If S Register 9039 is changed to a value 1 to N+1, then on power up, if S Reg 9003 indicates that HID profile is to be made available, then it will implement the appropriate custom HID descriptor in the service discovery database.

When custom HID descriptors are downloaded and stored there is no validation performed on the block of data. This is because the module has no context to perform such validation.

In MP mode, to download a custom HID descriptor, you can use the utility MpBtHost.exe. Right click on the window to invoke a pop-up menu and select "Upload Hid Descriptor" and in the new dialog box enter the blob id (recommend leave at 0) and Hid Id to use. Then to use that descriptor update S Register 9039 with a value which is HidId+1.

## HDP Profile (Health Device Profile)

UUID : 1400,1401,1402

## Background

Health Device Profile (HDP) is available on the module in both Agent and Manager roles as defined by the Continua Alliance (see www.continua.org). There are two aspects to HDP, one is the transport layer, for which only Bluetooth is catered for by this module (although the Continua Alliance has also ratified others, for example USB), and the other aspect is IEEE data encapsulation.

The Laird module provides a tightly coupled integrated solution for a Weigh Scale Specialization Agent. More specializations will be provided in the future as and when there is demand via a firmware update.

It is assumed that the reader is familiar with all the HDP and IEEE documentation and relevant guidelines published by the Continua Alliance. For HDP it is assumed that the reader has access to the specification from the Bluetooth SIG and for IEEE it is assumed that the reader has access to the IEEE11073-20601 Optimised Exchange Protocol specification and the device specializations specifications 11073-10401 through to 10499. For the Weigher Scale specialization embedded in the module the specification is 11073-10415. The IEEE standards can be obtained from their website standards.ieee.org and the Bluetooth HDP specification can be obtained from www.bluetooth.org

The IEEE data specialization along with the Bluetooth physical transport as defined in the appropriate specifications is very dry and difficult to understand and it would be pointless to reproduce that information here verbatim. However, an attempt will be made to describe it from the module's usage point of view where the module and the functionality it provides is treated like a black box manner.

## IEEE ‘Black Box’ Model

In a traditional health related environment, typical actors and props are the patient, instruments that measure appropriate parameters, health professionals and the (manual and/opr automatic) archiving of the records.

Over the years there have been many suppliers of the “instruments that measure appropriate parameters” who have all provided for proprietary methods for getting the data stored in records.

It has always been the role of the health professionals to ‘transcribe’ the data from the various instruments into the archive records. This manual process presents risks associated with errors in the transcribing process and so manufacturers have provided yet more proprietary solutions to automating that task.

The Continua Alliance came about to address that confused picture with guidelines and a certification process to ensure that a consistent inter-operable picture emerges with regards to the “instruments that measure appropriate parameters” and the “method for getting the data stored in records”,

The last thing the Continua Alliance would want to do is dictate how any individual instrument (referred to as an Agent) is physically designed as that is best left to the engineers who know how best to design them.

Instead they have specified abstract data models for the various types of instruments, which they refer to as Data Specializations, and how they shall convey the data to an entity called a ‘Manager’ that can be used to consistently store the data in an archive.

Examples of data specialization abstract models exist for Weigh Scales, Thermometers, Glucose meters, Blood Pressure meters, ECGs and many more will become available as they progress through various stages in appropriate working groups. Any Continua Alliance member is free to recommend creation of data specializations as needed. Once ratified, the end result is always an abstract data model which defines what data is pertinent for that instrument and how it shall be presented to the real world.

## Abstract Data Model

From a software engineer’s perspective, an abstract data model for an IEEE data specialization can best be described as a collection of arrays of different types of data (which the specifications refer to as attributes).

Each attribute is unambiguously defined to consist of a tag, a type and the actual value. There is no reliance on any programming language in the definition. It is purely a data model.

As a minimum there shall be one array of attributes called the ‘Medical Device System’, henceforth referred to as an MDS which represents the properties and services of the device, independent of its health data capabilities and its status and there shall be one array of attributes called the Numeric, henceforth referred to as NU which contains episodic measurements. There is also an RT-SA collection which is used to represent continuous samples or waveforms.

Other collections exist and the reader is advised to refer to the IEEE11073-20601 standard for a definitive list which will be described under the general heading of “Domain Information Model”.

At this point, you should be visualising an HDP agent as just a collection of data records that can be read and written to locally under program control and each data point is identified by its tag and will publish its data type. This is analogous to a database table with 4 fields in each record. Three fields called ‘Tag’, ‘Type’ and ‘Value’ and the fourth field being called ‘Collection Name’ such as MDS or NU or RT-SA.

You should further visualise this ‘database’ as being accessible from a Manager over a physical transport media such as Bluetooth or USB. The procedure a Manager shall use to gain access to that database of attributes is rigidly defined and standardised via a ‘Service Model’ using an association state machine defined in the IEEE11073-20601 standard. This ‘Service Model’ is encapsulated in the Laird Module and the user is encouraged to think of it in terms of a black box whose internal details are not relevant.

The picture that should emerge for the Laird module user who requires a data specialization is that of a black box consisting of that conceptual database with a 4 field table and an ‘engine’ that implements the association service model over Bluetooth so that it facilitates the mirroring of that said database at the HDP manager end.

This picture then vastly simplifies the design and development of a health instrument that has a requirement to be Continua Alliance certified. The hope is that, the user is only required to have a general idea about the content of the IEEE 11073-20601 and the data specialization IEEE11073-104xx standards.

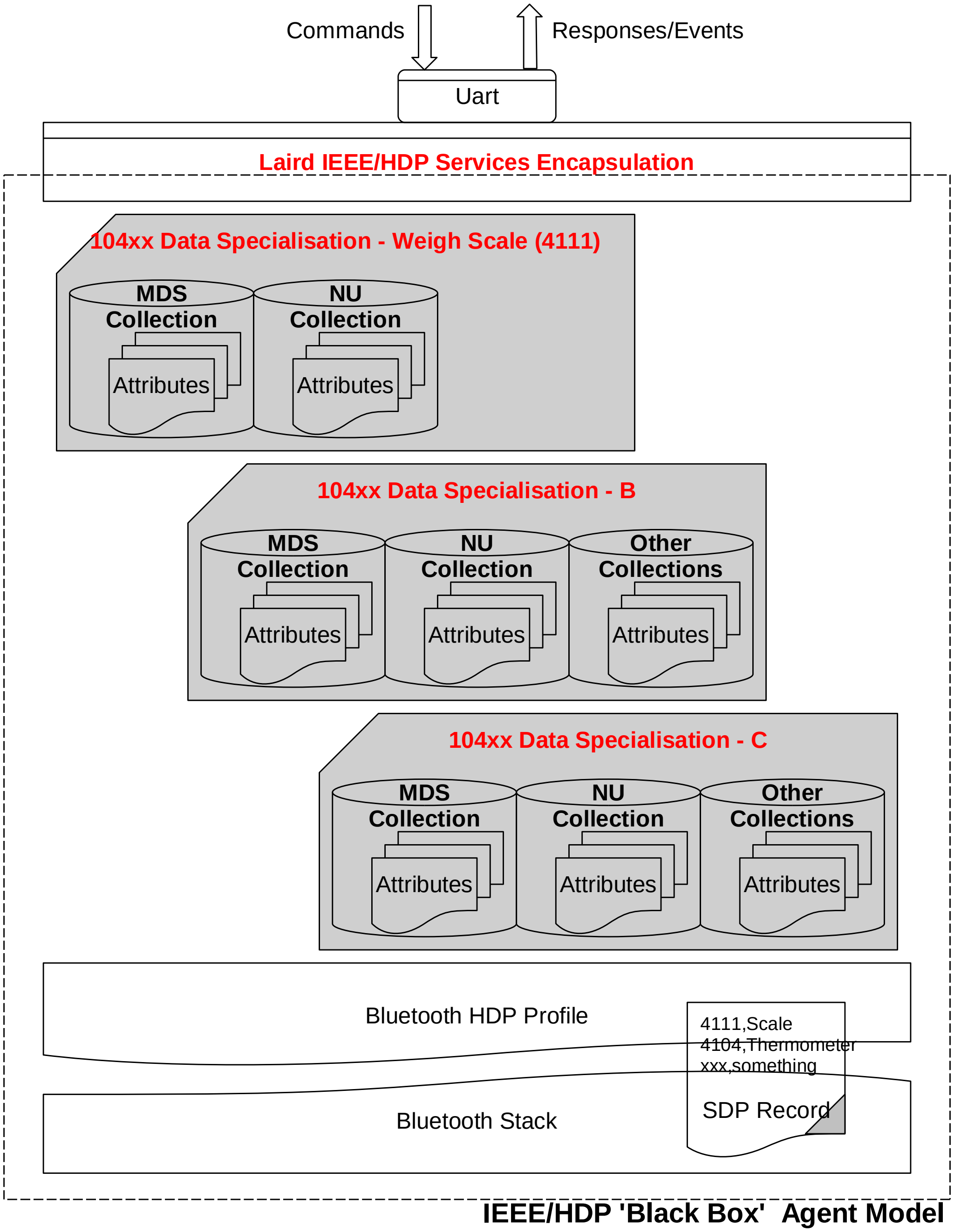
The following subsections provide more details as to how the black box can be controlled and manipulated. Please note that initially only a Weigh Scale data specialization as defined in 11073-10415 has been made available **embedded** inside the black box. By ‘embedded’ it is implied that the MDS and NU collections are pre-defined as per the standard and the attributes that can have the values changed are exposed to the user for manipulation.

In future it is hoped that a generic api will be exposed which will allow any data specialization to be downloaded and tested. Until that generic API is made available, if a user requires a specific data specialization then they are encouraged to contact Laird with that request.

It is also pertinent to note here that once a user has a working instrument using this module, it is up to the user to obtain Bluetooth Listing (quoting the QDID of the Laird module) prior to Continua Alliance testing and certification.

## HDP Agent Model

From a software perspective the HDP Agent implementation is as shown in the diagram below.

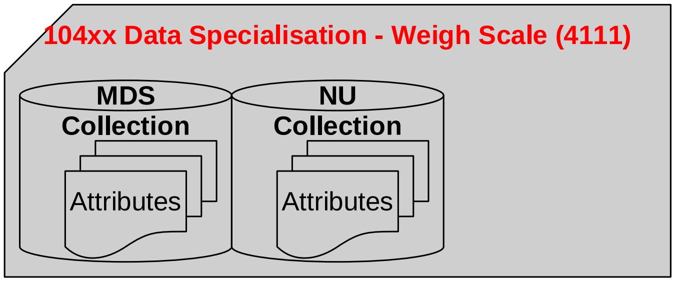


The diagram shows that the Agent model consists of the Bluetooth communications stack which consists of an SDP record that exposes to the outside world the data specializations it is capable of, a ‘Laird IEEE/HDP Service Encapsulation’ layer which is used to relay commands and responses to the host and 0 or more instances of Data Specializations. At the time of the first release of the firmware only a Weigh Scale specialization is offered.

All UART commands available to the host are provided so that the various entities in the black box can be controlled or interrogated. Given there can be many agent specializations embedded in the firmware, they are identified in various commands using a 16bit handle.

## Weigh Scale Data Specialization

The Weigh Scale Specialization (nominal code 4111) is embedded in the firmware is shown as below and it contains a MDS and an NU object.



The MDS object is defined in the firmware with the following attributes:-

| **Attribute Tag** | | **Data Type** | **Comments** |
|---|---|---|---|
| MDC_ATTR_ID_HANDLE | 2337 | HANDLE | Always: 0 |
| MDC_ATTR_SYS_TYPE_SPEC_LIST | 2650 | TYPE_SPEC_LIST | Const |
| MDC_ATTR_ID_MODEL | 2344 | SYSTEM_MODEL | Var:SystemModel |
| MDC_ATTR_SYS_ID | 2436 | OCTET_STRING | Var:SystemId |
| MDC_ATTR_DEV_CONFIG_ID | 2628 | CONFIG_ID | 1500 (0x05DC) |
| MDC_ATTR_ATTRIBUTE_VAL_MAP | 2645 | ATTR_VAL_MAP | Const |
| MDC_ATTR_ID_PROD_SPECN | 2349 | PROD_SPEC | Const |
| MDC_ATTR_TIME_ABS | 2439 | ABSOLUTE_TIME | Var:Time |
| MDC_ATTR_MDS_TIME_INFO | 2629 | MDS_TIME_INFO | Const |
| MDC_ATTR_POWER_STAT | 2389 | POWER_STATUS | Var:PowerStatus |

The NU object is defined with the following attributes

| **Attribute Tag** | | **Data Type** | **Comments** |
|---|---|---|---|
| MDC_ATTR_ID_HANDLE | 2337 | HANDLE | Always: 1 |
| MDC_ATTR_ID_TYPE | 2351 | TYPE | Const |
| MDC_ATTR_METRIC_SPEC_SMALL | 2630 | METRIC_SPEC_SMALL | Const |
| MDC_ATTR_UNIT_CODE | 2454 | OID_TYPE | Var:Weight Units |
| MDC_ATTR_ATTRIBUTE_VAL_MAP | 2645 | ATTR_VAL_MAP | 2646,2448 |
| MDC_ATTR_TIME_STAMP_ABS | 2448 | ABSOLUTE_TIME | Var:Time |
| MDC_ATTR_NU_VAL_OBS_SIMP | 2646 | SIMPLE_NU_OBS_VAL | Var:Weight |
| MDC_ATTR_NU_ACCUR_MSMT | 2378 | FLOAT_TYPE | Always: 1 |
| MDC_ATTR_MSMT_STAT | 2375 | MEASUREMENT_STATUS | Var:Measurement Status |

The attributes commented as ‘variables’ are exposed to the host for reading and writing via the UART interface using AT+HAG and AT+HAS commands respectively and are described in detail elsewhere in this document.

The variable attributes mentioned above are identified by the host on the UART interface using an attribute ID and an additional sub ID. The concept of ‘sub ID’ is a Laird artefact and is not part of any IEEE standard, but the attribute ID will in most cases be the same as those defined in the IEEE standard. The complete list for the Weigh Scale specialization is as per the table below and should be used with the agent attribute read/write commands AT+HAG and AT+HAS.

**PLEASE NOTE**: the attribute values passed back and forth from the host are NOT validated in any way by the firmware in the Laird module. It is up to the host to ensure that the correct data is written into an attribute. Any illegal values will be picked up at time of Continua Alliance certification testing which will prevent certification.

Table: Variable Attributes in Weigh Scale Specialization

| **Attribute Name**<br>**(See IEEE spec for format)** | **Atr Id** | **Sub Id** | **Size in bytes** |
|---|---|---|---|
| Weight | 2646 | 0 | 4 |
| Weight Units | 2454 | 0 | 2 |
| Time | 2448 | 0 | 8 |
| Power Status | 2389 | 0 | 2 |
| Measurement Status | 2375 | 0 | 2 |
| System ID | 2436 | 0 | 8 |
| System Model – product name | 2344 | 0 | 12 |
| System Model – model name | 2344 | 1 | 16 |
| Serial Number | 2349 | 0 | 8 |

## Agent Related AT Commands

This section describes all the commands used to manage the Agent role for HDP.

### Connection To A HDP Manager

Command: **AT+HAAhhhh**

Response: *<cr,lf> OK<cr,lf>*

*Or*
*<cr,lf>ERROR nn<cr,lf>*

*Or*
*<cr,lf>HAD:ASSOCIATE xxxx<cr,lf>OK<cr,lf>*

*Or*
*<cr,lf> HAD:DISASSOCIATE xxxx<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to establish a connection to a hdp manager and associate the agent with it so that attribute data can be exchanged. The Bluetooth address of the hdp manager and the agent specialization that needs to be associated is defined by the handle 'hhhh' which is pre-obtained using command AT+HAB.

This command will wait for the procedure to complete successfully or otherwise before responding with OK or ERROR. If the agent is already associated then an immediate OK will be the response.

If the agent is not already associated then a Bluetooth connection will be initiated and as soon as a connection is established the association state machine will progress through to negotiating a configuration and then ultimate confirmation of association. If association is successful then a "HDA:ASSOCIATE ….." asynchronous response will be sent to the host and if association fails (because BT connection failed or configuration could not be negotiated) then the async response "HDA:DISASOCIATE …" is sent to the host. An OK or ERROR response terminates the procedure.

Note if S Reg 9071 is non-zero, then there will be an automatic disassociation after a time specified by that register. The timer is restarted every time a scan report is sent to the manager.

SReg Required Settings**:** Bit 2 set in S9003 and S9070=0

## Bind A Data Specialization

Command: **AT+HAB<bd_addr>,iiii**

Response: *<cr,lf>hhhh<cr,lf>OK<cr,lf>*

Or

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to bind a data specialization identified by the nominal code 'iiii' (for example 4111 = Weigh Scale) with a HDP manager identified by the Bluetooth address '<bd_addr>'. For this command to be successful, the data specialization identified by 'iiii' has to be pre-embedded in the firmware. Although the firmware will allow multiple bindings to the same specialization, please be aware that it is the same object and each instance will not have unique set of attributes.

If the binding is successful, then a 16 bit handle 'hhhh' (a decimal number) will be returned which is then used as a parameter in many subsequent commands.

SReg Required Settings**:** Bit 2 set in S9003 and S9070=0

## Disassociate An Agent

Command: **AT+HADhhhh**

Response: *<cr,lf>OK<cr,lf>*
*Or*
*<cr,lf>> HAD:DISASSOCIATE xxxx<cr,lf>OK<cr,lf>*
*Or*
*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to disassociate an agent identified by the handle hhhh, from a manager.

This command will wait for the procedure to complete successfully or otherwise before responding with OK or ERROR. If the agent is already disassociated then an immediate OK will be the response.

SReg Required Settings: Bit 2 set in S9003 and S9070=0

Command: **AT+HAEiiii,"name"**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command must be issued prior to sending the AT+HAL commands and is used to include the data specialization endpoint 'iiii' with the string "name" as a HDP source (agent) in the SDP record that will be exposed to potential peers. This in effect tells potential peers that the device offers a 'iiii' data specialization source.

Without this entry in the SDP record, a manager is not able to make a connection to the HDP agent, although it is rare for a manager to initiate connections in typical usage scenarios.

SReg Required Settings**:** Bit 2 set in S9003 and S9070=0

Command: **AT+HAGhhhh,aaaa,ssss**

Response: *<cr,lf>hhhhhhhhhhhhhh<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to read (**g**et) the value of one of the variable attributes identified by 'aaaa' and sub id 'ssss' in the attribute collections for that agent identified by the handle 'hhhh'. For the embedded weigh scale data specialization this command is used to read the value of an attribute listed in the table identified as "Table: Variable Attributes in Weigh Scale Specialization" above.

The value is always returned as a string of hexadecimal digits representing the binary value, and the size of that string will be even. Different attributes will have different sizes.

SReg Required Settings: Bit 2 set in S9003 and S9070=0

Command: **AT+HAL**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command must be issued after sending at least one AT+HAE command and is used to register and activate the SDP record with information supplied in the AT+HAE commands.

Without the SDP record, a manager is not able to make a connection to the HDP agent, although it is rare for a manager to initiate connections in typical usage scenarios.

SReg Required Settings: Bit 2 set in S9003 and S9070=0

Command: **AT+HARhhhh,pppp[,aaaa[,aaaa[…]]]**

Response: *<cr,lf> OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to trigger a scan **r**eport for person id 'pppp' (16 bit decimal number) from the agent identified by handle 'hhhh'.

If the agent is not associated with the bound manager (see AT+HAB) then it will first trigger the start of an association.

Once association exists, a scan report will be sent.

If [,aaaa[,aaaa[…]]] is absent from the command, then the standard scan report as identified by the attribute MDC_ATTR_ATTRIBUTE_VAL_MAP in the NU collection is sent to the manager. Otherwise, the [,aaaa[,aaaa[…]]] can be a list of any attribute mentioned in the NU collection. For example, to send a scan report with just the measurement status, just specify the single value 2375.

Please note that only attributes mentioned in the NU collection are allowed. Any 'aaaa' value not found in the NU collection will be silently ignored.

SReg Required Settings: Bit 2 set in S9003 and S9070=0

Command: **AT+HAShhhh,aaaa,ssss,hhhhhhhhhhhh**

Response: *<cr,lf> OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to write (**s**et)a new value 'hhhhhhhhhhhh' to one of the variable attributes (identified by 'aaaa' and sub id 'ssss') in the attribute collections for the agent identified by the handle 'hhhh'. For the embedded weigh scale data specialization this command is used to write to an attribute listed in the table identified as "Table: Variable Attributes in Weigh Scale Specialization"

The value 'hhhhhhhhh' is in hexadecimal and is NOT validated in any way apart from the requirement that it shall be twice the size in bytes as specified for that attribute.

SReg Required Settings: Bit 2 set in S9003 and S9070=0

## Agent Related AT Asynchronous Responses

This section describes all the asynchronous responses sent to the host by the HDP Agent. Each response is framed by a <cr,lf> at the start and end.

Command: No Command. This is a status message.

Response: *HDA:DISASSOCIATED hhhh*

Description: This response is sent to the host when an association attempt fails (as a result of AT+HAA or AT+HAR commands) or when an association is terminated by either AT+HAD or due to loss of Bluetooth connectivity. The parameter 'hhhh' which is a 16 bit decimal number identifies the agent.

Command: No Command. This is a status message.

Response: *HDA:ASSOCIATED hhhh,iiii,cccc,ssssssssssss*

Description: This response is sent to the host when a successful association happens for the agent identified by 'hhhh' (16 bit decimal number). For completeness, the data specialization nominal code 'iiii' (16 bit decimal) and the device configuration id 'cccc' (16 bit decimal), that got negotiated with the manager, is also provided. The 16 character parameter 'ssssssss' specifies the system id of the manager.

Note: An agent data specialization can be basic as specified in the relevant spec, or one of the more enhanced ones which have more capabilities. For example, with the weigh scale, Laird have provided for the basic MDS collection, but it is possible to specify multiple MDS collection which expose more functionality (e.g. body mass index attributes). These extended collections are identified by configuration IDs and are offered to a manager during the association phase in descending order of complexity until the manager accepts one that it can work with.

Command: No Command. This is a status message.

Response: *HDA:TIME hhhh,ccyymmddhhmmssaa*

Description: This response is sent to the host when a manager sends new time information by writing to the MDC_ATTR_TIME_ABS attribute in the MDS collection of the agent identified by ‘hhhh’. The time ccyymmddhhmmssaa is a 16 character hexadecimal value which is encoded as follows:-

CC Century (e.g. 14==20)

YY Year (e.g. 0B==11)

MM Month (e.g. 0C==12)

DD Day (e.g 1F==31)

HH Hour (e.g. 17==23)

MM Minutes (e.g 32==50)

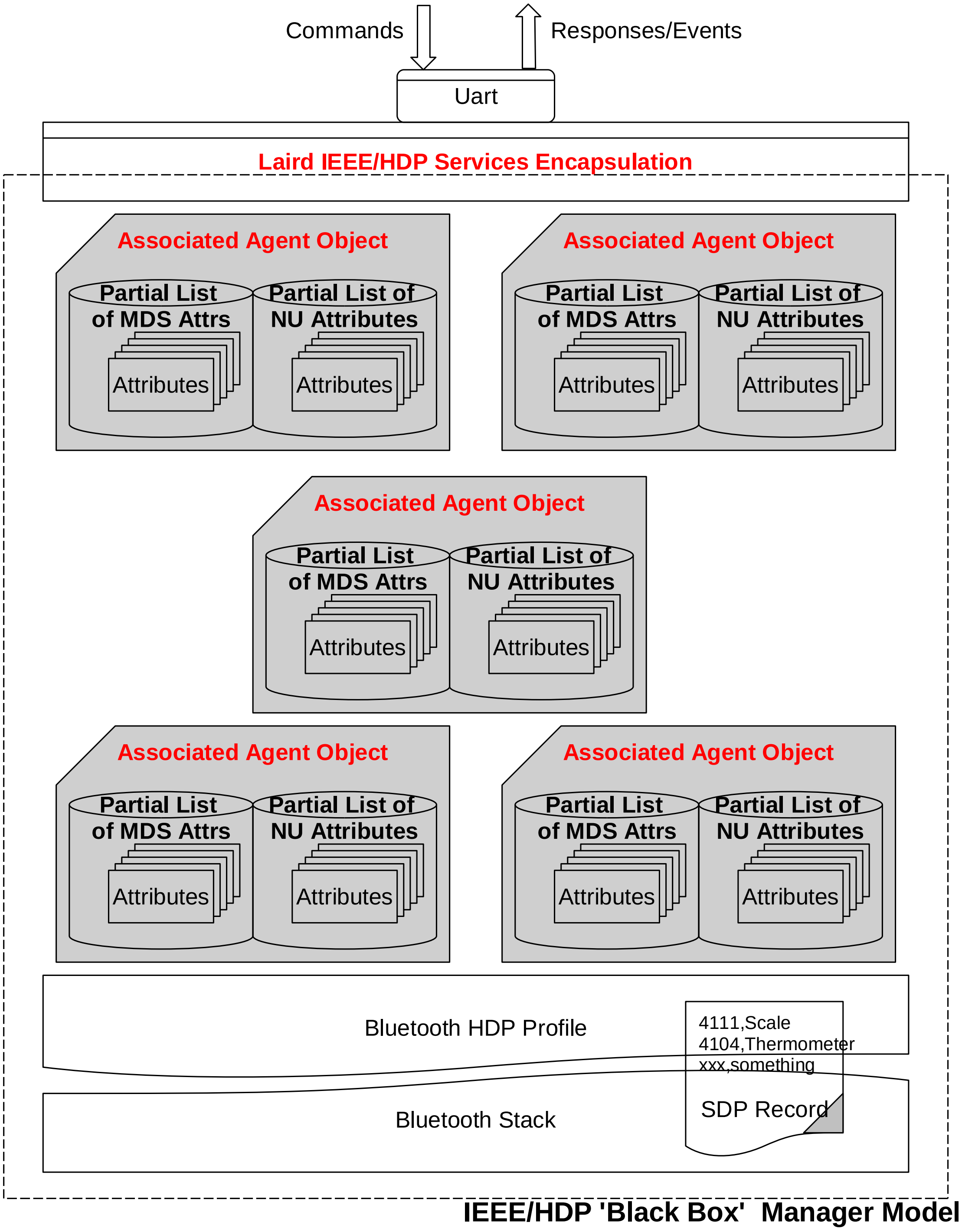
SS Seconds (e.g. 2F==47)

AA Hundreths of seconds (e.g 63==99)

For example, the data and time 12 Feb 2011, 16:45:33.78 will be sent as 140B020C102D214E

## HDP Manager Model

From a software perspective the HDP Manager implementation is as shown in the diagram below and the functionality is provided mainly to enable prototyping and regression testing of agent specializations. There are many far more capable HDP Managers available which are hosted on a PC. For example, the latest Toshiba Bluetooth Stack is HDP capable and there are imminent BlueZ releases for Linux PCs.



The diagram above shows that the Manager model consists of the Bluetooth communications stack which consists of an SDP record that exposes to the outside world the data specializations it is capable of accepting as sinks, a ‘Laird IEEE/HDP Service Encapsulation’ layer which is used to relay commands and responses to the host and 0 or more instances of Specialization Associations. These ‘Associated

Agent Objects' are transient and come into existence only when an agent successfully associates. The associated process results in the top few attributes in the MDS and NU collections being cached in the Manager which can be read by the host using the AT+HMG command after an association.

Just like the Agent end, the Manager will indicate via it's SDP record which Data Specializations it is capable of sinking, and commands analogous to the ones provided for the Agent model have been provided to manipulate that SDP record content.

## Manager Related AT Commands

This section describes all the commands used to manage the Agent role for HDP.

Command: **AT+HMEiiii,"name"**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command must be issued prior to sending the AT+HML commands and is used to include the data specialization endpoint 'iiii' with the string "name" as a HDP sink in the SDP record that will be exposed to potential agents. This in effect tells potential peers that the device offers a 'iiii' data specialization sink.

Without this entry in the SDP record, an appropriate agent is not able to make a connection to the HDP Manager.

SReg Required Settings: Bit 2 set in S9003 and S9070=1

Command: **AT+HMGhhhh,oooo,aaaa**

Response: *<cr,lf>hhhhhhhhhhhhhh<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command is used to read the value of one of the cached attributes for an agent identified by 'hhhh'. The parameter 'oooo' shall be 0 for attributes in the MDS collection cache and '1' for the NU collection cache and 'aaaa' is the id of the attribute to read.

The value is always returned as a string of hexadecimal digits representing the binary value, and the size of that string will be even. Different attributes will have different sizes.

SReg Required Settings: Bit 2 set in S9003 and S9070=1

Command: **AT+HML**

Response: *<cr,lf>OK<cr,lf>*

*Or*

*<cr,lf>ERROR nn<cr,lf>*

Description: This command must be issued after sending at least one AT+HME command and is used to register and activate the SDP record with information supplied in the AT+HME commands.

Without the SDP record, an agent is not able to make a connection to the HDP manager.

SReg Required Settings**:** Bit 2 set in S9003 and S9070=1

Command: AT+HMThhhh, ccyymmddhhmmssaa

Response: <cr,lf> OK<cr,lf>

Or

<cr,lf>ERROR nn<cr,lf>

Description: This command is used to send the current data and time to the associated agent identified by handle ‘hhhh’.

The time ccyymmddhhmmssaa is a 16 character hexadecimal value which is encoded as follows:-

| | |
|---|---|
| CC | Century (e.g. 14==20) |
| YY | Year (e.g. 0B==11) |
| MM | Month (e.g. 0C==12) |
| DD | Day (e.g 1F==31) |
| HH | Hour (e.g. 17==23) |
| MM | Minutes (e.g 32==50) |
| SS | Seconds (e.g. 2F==47) |
| AA | Hundreths of seconds (e.g 63==99) |

For example, the data and time 12 Feb 2011, 16:45:33.78 will be sent as 140B020C102D214E

SReg Required Settings: Bit 2 set in S9003 and S9070=1

## Manager Related AT Asynchronous Responses

This section describes all the asynchronous responses sent to the host by the HDP Manager. Each response is framed by a <cr,lf> at the start and end.

Command: No Command. This is a status message.

Response: *HDM:DISASSOCIATED hhhh*

Description: This response is sent to the host when an association is terminated by an agent or due to loss of Bluetooth connectivity. The parameter 'hhhh' which is a 16 bit decimal number identifies the agent.

Command: No Command. This is a status message.

Response: *HDM:ASSOCIATED hhhh,iiii,cccc,ssssssssssss*

Description: This response is sent to the host when a successful association happens for the agent identified by 'hhhh' (16 bit decimal number). For completeness, the data specialization nominal code 'iiii' (16 bit decimal) and the device configuration id 'cccc' (16 bit decimal), that got negotiated with the manager, is also provided. The 16 character parameter 'ssssssss' specifies the system id of the agent.

Note: An agent data specialization can be basic as specified in the relevant spec, or one of the more enhanced ones which have more capabilities. For example, with the weigh scale, Laird have provided for the basic MDS collection, but it is possible to specify multiple MDS collection which expose more functionality (e.g. body mass index attributes). These extended collections are identified by configuration ids and are offered to a manager during the association phase in descending order of complexity until the manager accepts one that it can work with.

Command: No Command. This is a status message.

Response: *HDM:SCANPERIOD hhhh<more>*

Description: This response is sent to the host when a scan report is received from agent identified by handle 'hhhh'. Since a scan report can consist of data values for many attributes, this report is formatted with embedded <lf> characters as follows:-

**<**cr><lf>HDM:SCANREPORT hhhh:pppp

<lf>O:oooo

<lf>A:aaaa,ddd..ddd

<lf>A:aaaa,ddd..ddd

…

<lf>O:oooo

<lf>A:aaaa,ddd..ddd

<lf>A:aaaa,ddd..ddd
…
…

<cr><lf>

Where <lf>O: identifies the collection object (oooo == 1 for NU etc) and then subsequence <lf>A: lines consist of value pairs aaaa,ddd.ddd where 'aaaa' is the attribute nominal code and ddd...ddd is its value in hexadecimal. The size of the value for each attribute is specified in the IEEE standards.

## Sample Host/Module Message Sequence

In a typical weigh scale usage, the sequence of commands (red) and responses (blue) from the module will be as follows, where it assumed that the both ends start of from factory default state:-

```
=====================================    ========================================
AGENT                                      MANAGER
=====================================    ========================================
                                     ats9003=5
                                     OK
                                     ats9070=1
                                     OK
                                     at&w
                                     OK
ats9003=5
OK
ats9070=0
OK
at&w
OK
                         ============================
                                  PAIRING
                         ============================

AT+BTW0016a4fef005
OK
PAIR 0 0016A4FEF005                    PAIR 0 0016A4FEF004


                         ============================
                            REGISTER SDP RECORDS
                         ============================
                                     AT+HME4111,"scale"
                                     OK
                                     AT+HML
                                     OK
AT+HAE4111,"scale"
OK
AT+HAB0016a4fef005,4111
46492
OK
AT+HAL
OK
                         ============================
                                 ASSOCIATE
                         ============================
AT+HAA46392
HDA:ASSOCIATED 46392,4111,1500,4C414952444D4752
                                     HDM:ASSOCIATED 2936,4111,1500,0016A4FEF004B538
OK

                         ============================
                          READ ATTIBUTES @ MANAGER
                         ============================
                                     AT+HMG29364,0,2628
                                     05DC
                                     OK

                         ============================
                           READ ATTIBUTES @ AGENT
                         ============================
AT+HAG46392,2646,0
0000004B
OK
```

```
                        ============================
                          WRITE ATTIBUTES @ AGENT
                        ============================
AT+HAS46392,2646,0,00000123
OK
AT+HAG46392,2646,0
00000123
OK
                        ============================
                                SEND TIME
                        ============================
                                    AT+HMT29364,140B020C102D214E
                                    OK
HDA:TIME 46392,140B020C102D214E

                        ============================
                           SEND FIXED SCAN REPORT
                        ============================
AT+HAR46392,1234
                                    HDM:SCANREPORT 29364:1234
                                    O:1
                                    A:2646,00000123
                                    A:2448,2011021216453378
OK

                        ============================
                           SEND VARIABLE SCAN REPORT
                        ============================
AT+HAR46392,1234,2454,2448,2646,2375
                                    HDM:SCANREPORT 29364:1234
                                    O:1
                                    A:2454,06C3
                                    A:2448,2011021216453378
                                    A:2646,0000004B
                                    A:2375,8000
OK

                        ============================
                               DISASSOCIATE
                        ============================
AT+HAD46392
OK
HDA:DISASSOCIATED 46392                 HDM:DISASSOCIATED 29364
```

## Authentication and Encryption

The module and firmware is BT v2.1 compliant so it uses Simple Secure Pairing (SSP) to authenticate devices to trust and will only invoke a legacy pairing procedure when a peer device is v2.0 or older.

It is not possible to configure the unit to be only capable of legacy pairing and still have v2.1 approvals.

The purpose of pairing, whether legacy or SSP, is to generate the same random 16 byte key at both ends which is then used in subsequent connections for authentication and encryption.

## Legacy Pairing

A legacy pairing procedure is automatically used when pairing is initiated (by either end) and the peer is approved to v2.0 and below.

**Outgoing**

To initiate a pairing, the host shall submit the command “AT+BTW<bd_addr>” to which it will get an immediate OK or ERROR response. The host then shall wait for a “PIN ? <bd_addr>” response to which it shall respond with the command AT+BTK=”pincode”.

When the pairing procedure is complete, the module will send to the host the following asynchronous response “PAIR N <bd_addr>”. Where N is 0 when the pairing is successful, 1 for a timeout and 2 for a generic failure (for example, mismatching pincode).

**Incoming**

The module has to be in at least connectable mode for it to participate in a pairing initiated from a legacy peer. The first indication the host will get that an incoming legacy pairing has been initiated is when it receives the asynchronous response “PIN ? <bd_addr>”. To this, the host shall respond with a shared pincode conveyed in the command AT+BTK=”pincode”.

When the pairing procedure is complete, the module will send to the host the following asynchronous response “PAIR N <bd_addr>”. Where N is 0 when the pairing is successful, 1 for a timeout and 2 for a generic failure (for example, mismatching pincode).

## Simple Secure Pairing

Simple secure pairing was introduced in v2.1 of the Bluetooth specification to simplify the pairing procedure so it was not reliant on a pre-shared pincode so that all connections are forced to be encrypted. Unlike pre v2.1 devices, it is not possible to create connections without encryption.

Simple secure pairing uses the Diffie-Hellman public/private encryption methodology to expedite a common 128 bit key at both ends. This eliminates the need for pre-shared pincodes but introduces the ‘man in the middle’ (MITM) attack vulnerability.

To address the MITM vulnerability the concept of verification via a 6 digit passcode was also added. A 6 digit passcode was selected, as that reduces the probability of a random MITM attack being successful to 1 in a million.

The 6 digit passcode is NOT a pre shared code, but is a random 6 digit artefact derived from the Diffie-Hellman calculations such that knowledge of that 6 digit number by an attacker cannot result in back-calculation of the 128 bit key that was generated.

For a user to interact and process the 6 digit passcode the SSP procedure requires that each Bluetooth device have an I/O capability which is one of : None, Display Only, Display with a Yes/No button and Keyboard only. This i/o capability is exchanged by the two peers going through a pairing procedure so that the optimal user interaction is selected at both ends. For example, if one end admits to keyboard only and the other to Display only, then the two will negotiate that the display end will display the passcode with an appropriate prompt to get the user at the other end to type in the passcode.

When either has ‘none’ capability, then pairing procedure will complete without any MITM protection by both ends automatically accepting the passcode generated by the pairing algorithm.

### I/O Capability

The I/O capability of the module is set via S Register 6 (9006 in AT mode)where the value to set is as follows:-

12 = No I/O capability

13 = Display with Yes/No

14 = Keyboard only

15 = Display Only

When both ends are keyboard only, a pre-shared 6 digit number can be entered.

When both ends are Display Only it is unlikely the two devices will have services that are of use to either and so it could be contrived combination.

### Outgoing

To initiate a pairing, the host shall submit the command “AT+BTW<bd_addr>” to which it will get an immediate OK or ERROR response. The host then shall wait for a “PASSKEY? N <bd_addr>” response to which it shall respond with the command AT+BTK=”passcode” or “AT+BTKY” or “AT+BTKN” depending on the value of N in the “PASSKEY?” message.

When the pairing procedure is complete, the module will send to the host the following asynchronous response “PAIR N <bd_addr>”. Where N is 0 when the pairing is successful, 1 for a timeout and 2 for a generic failure (for example, mismatching pincode).

As you can see, the host is able to determine if SSP or legacy pairing is in progress because in the former the challenge message is “PASSKEY?” where as in the latter it is “PIN ?”

### Incoming

The module has to be in at least connectable mode for it to participate in a pairing initiated from a SSP capable peer. The first indication the host may get that an incoming pairing has been initiated is when it receives the asynchronous response “PASSKEY? N <bd_addr>”. To this, the host shall respond with the command AT+BTK=”passcode” or “AT+BTKY” or “AT+BTKN” depending on the value of N in the “PASSKEY?” message.

When the pairing procedure is complete and for ‘just works’, the module will send to the host the following asynchronous response “PAIR N <bd_addr>”. Where N is 0 when the pairing is successful, 1 for a timeout and 2 for a generic failure (for example, mismatching pincode).

As you can see, the host is able to determine if SSP or legacy pairing is in progress because in the former the challenge message is “PASSKEY?” where as in the latter it is “PIN ?”. In the ‘Just Works’ scenario the PAIR 0 message informs the host that a pairing has completed.

### Host processing of the “PASSKEY? N” response

The full format of the PASSKEY? message to the host is

PASSKEY? N <bd_addr>[,passcode]

Where N is 1,2 or 3 and “,passcode” is not present when N=3.

When N=1, this message requires the host to just display the passcode. The module is not expecting any confirmation from the host.

When N=2, this message requires the host to display the passcode so that the user can accept or reject the pairing. To accept the pairing the host shall send the AT+BTKY command and to reject it shall send AT+BTKN.

When N=3, the passcode is not provided and the host shall submit the AT+BTK=”passcode” command. To reject, it can send any value no matching the passcode displayed at the remote end OR send AT+BTK=””

In the case of pairing where the i/o capability is none, the pairing will occur with ‘just works’ procedure where both ends automatically accept the pairing. In this case the module will become aware that the pairing has happened when it receives the “PAIR N <bd_addr>” response.

## GPIO Access via SReg 619 and 620

The GPIO can be read an written to in AT mode via S registers 619 and 620.

SReg 620 is used to read the current states of all gpio pins and is displayed as hex valuewiht a '&' prefix.

To write to output pins a nonzero mask MUST be first written to S register 619.

Subsequently any value written to S Reg 620 will only affect GPIO pins which has a corresponding bit in S Reg 619 set to 1.

## GPIO Exchange via Rfcomm Modem Signalling

In a SPP connection, there is a modem signalling message that is exchanged between peers which is used to convey the status of 4 bits called RTR, RTC, DV and IC, which normally map to DTR/DSR, RTS/CTS, DCD and RI respectively.

This on-air message is transparent and happens in the 'background' as and when required.

The firmware in the module allows GPIO to be mapped to those 4 bits. In total 8 GPIO pins can be mapped. Four for inputs which are mapped to the bits that are sent to the peer and four for outputs which are updated when a modem signalling message arrives from the peer.

It is not necessary to map all 8 bits and it is perfectly acceptable to have no pins mapped (which is the default).

S Registers 651 to 654 inclusive are used to specify OUTPUT pins which get updated when a modem signal message arrives from a peer.

S Registers 661 to 664 inclusive are used to specify INPUT pins which are monitored for changes of state and when detected, result in a modem signal to be sent to the peer.

As a result, it is possible to convey the state of a digital pin to the peer with some inherent latency. The latency will be dependent on the quality of the rf connection and even with the best of connection the user must test actual timings to check that the latency is acceptable for the use case.

## Enhanced Inquiry Responses

Bluetooth 2.1 specification allows up to 240 Bytes of extended inquiry data. On BTM44x modules, this data is limited to a maximum length based on firmware builds due to internal memory restrictions. Extended inquiry data can be used to transmit e.g. the friendly name, UUIDs of supported profiles or user defined data within the inquiry process and without a Bluetooth connection.

The architecture for managing EIR data is composed of a blob buffer and a set of AT commands around them and:

- Baseband (EIR data visible to inquiring devices)
- RAM buffer (allows accumulation of data)
- EIR persistent store (non-volatile buffer, copied to baseband at boot time)

As the input buffer length for one AT command is limited, there is a RAM buffer to accumulate several short data packets. The accumulated data of the RAM buffer can be copied to the Baseband where it will become visible to other inquiring devices immediately.

The content of the RAM buffer can also be copied to the EIR persistent store. If the EIR persistent store contains data, it will be copied to the Baseband automatically at boot time.

This allows a flexible usage of extended inquiry data. For example, data with a low data rate (e.g. temperature) can be transmitted without creating a connection between Bluetooth devices, however without the benefits of encryption and authentication.

The command AT+BTB has been provided to manage EIR data in AT mode.

### EIR Data Format

When passing EIR data (“<data>”) to AT commands (AT+BTB=”<data>” / AT+BTB+”<data>”), each byte should be presented by its ASCII representation whenever it is a printable character.

Each non-printable ASCII character must be presented as 2 hex digits with a preceding ‘\’. For example, a byte of decimal value 5 would be presented as “\05” because the ASCII character of 05d is not printable.

A decimal value of 43 should be presented as ‘+’ because ‘+’ is the ASCII character representing 43d. The module would also accept “\2B” (the hexadecimal presentation of 43d) but at the price of two redundant characters.

Exceptions:

‘“’ (quotation mark) must be presented as \22

‘\’ (backslash) must be presented as \5C

When querying the content of the blob, non-printable ASCII characters will be presented by 2 hex digits with preceding ‘\’.

Exceptions:

‘“’ (quotation mark) is presented as \22

‘\’ (backslash) is presented as \5C

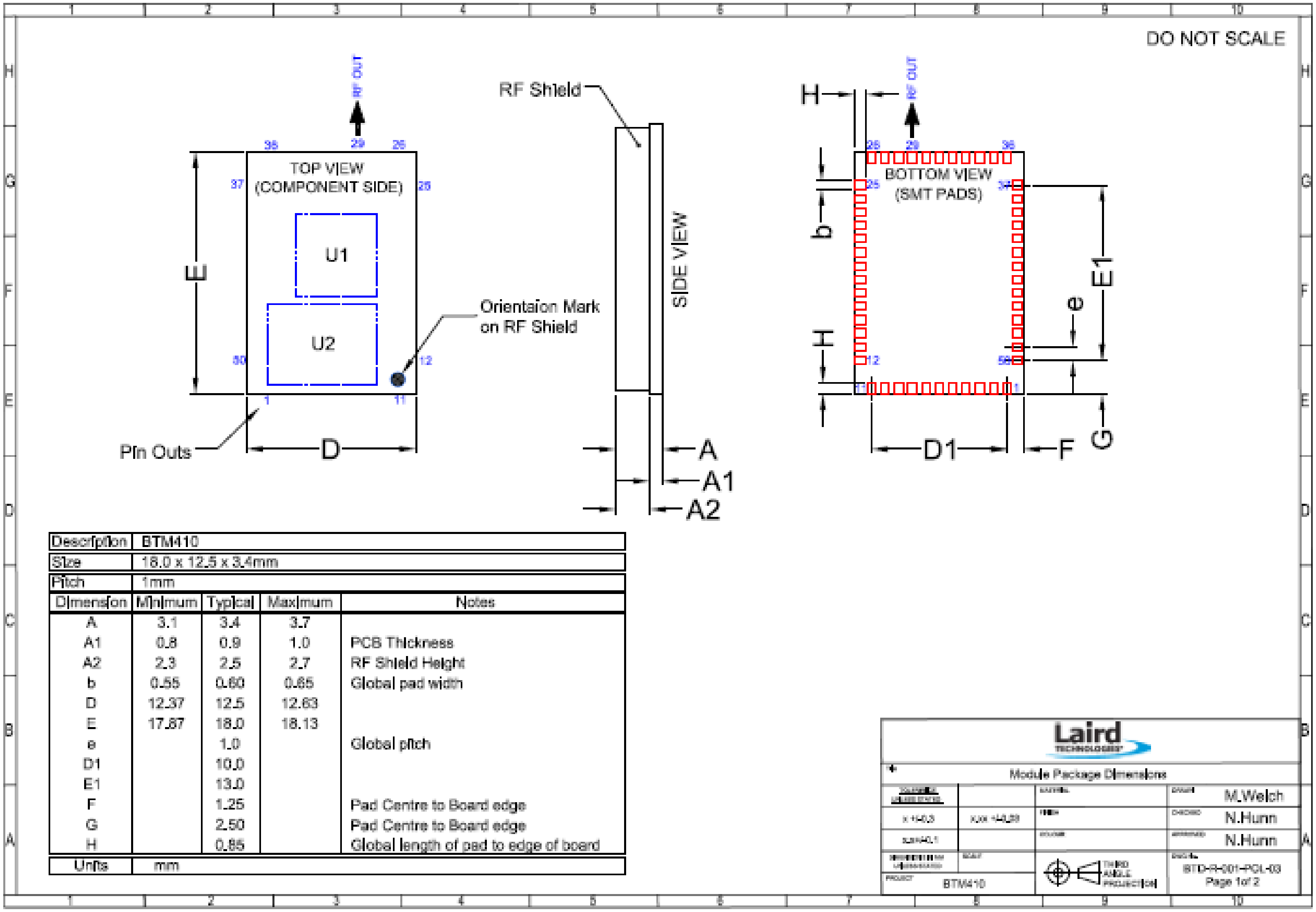
‘,’ (comma) is presented as \2C

Any data passed to the baseband must match the format defined in the Bluetooth Specification Version 2.1 + EDR [1], vol3, Part C – Generic Access Profile, 8 Extended Inquiry Response Data Format (page 1305 in the *.pdf file).

The AT command interpreter does not perform any checks on the baseband data format.

Mechanical layout

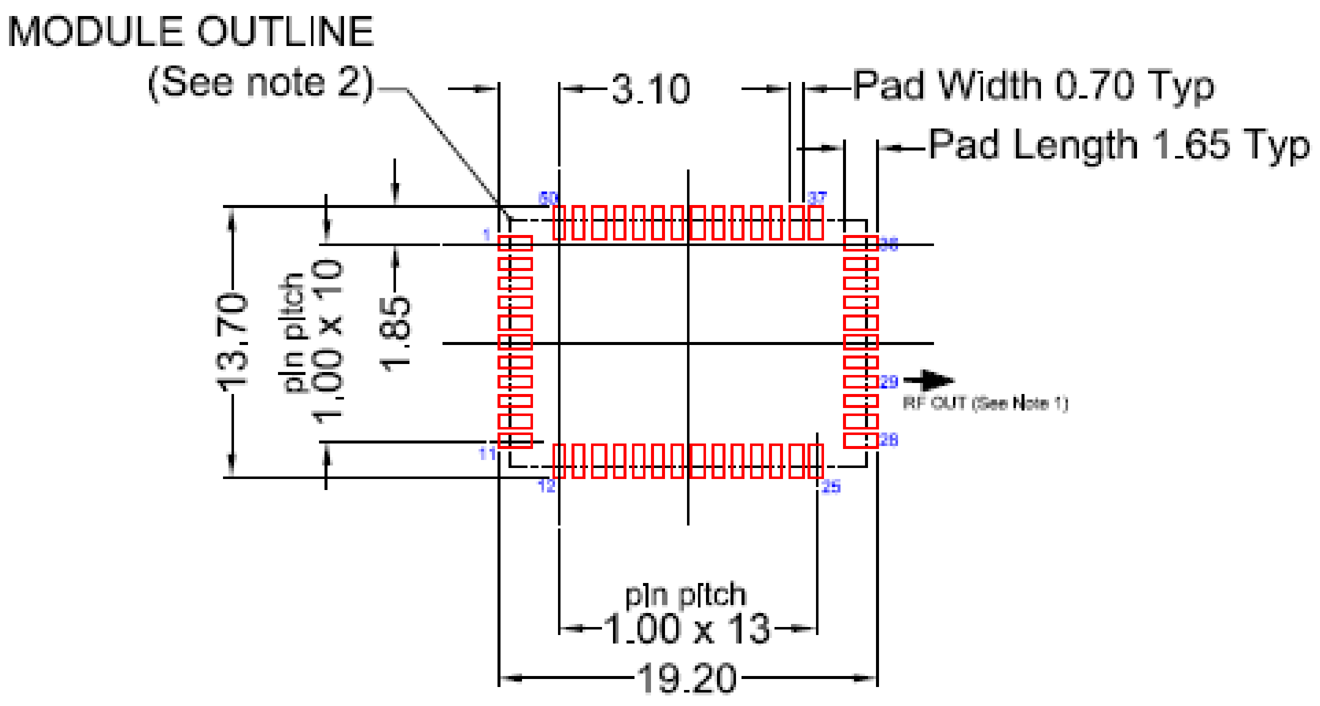
Figure 9-1: BTM440/442 Mechanical Details



| Description | BTM410 | | | |
|---|---|---|---|---|
| Size | 18.0 x 12.5 x 3.4mm | | | |
| Pitch | 1mm | | | |
| Dimension | Minimum | Typical | Maximum | Notes |
| A | 3.1 | 3.4 | 3.7 | |
| A1 | 0.8 | 0.9 | 1.0 | PCB Thickness |
| A2 | 2.3 | 2.5 | 2.7 | RF Shield Height |
| b | 0.55 | 0.60 | 0.65 | Global pad width |
| D | 12.37 | 12.5 | 12.63 | |
| E | 17.87 | 18.0 | 18.13 | |
| e | | 1.0 | | Global pitch |
| D1 | | 10.0 | | |
| E1 | | 13.0 | | |
| F | | 1.25 | | Pad Centre to Board edge |
| G | | 2.50 | | Pad Centre to Board edge |
| H | | 0.85 | | Global length of pad to edge of board |
| Units | mm | | | |

# PCB LAND PATTERN/DECAL DIMENSIONS

DO NOT SCALE



Notes

1 Connect External Antenna to RF I/O pin 29 with 50ohm microstrip or coplaner waveguide.

2: Ensure no exposed copper under module to avoid shorting to test points on underside of module.

3: The user may modify the PCB land pattern dimensions based on their experience and/or process capability.

Laird TECHNOLOGIES

Recommended Land Pattern/Decal

M.Welch

N.Hunn

N.Hunn

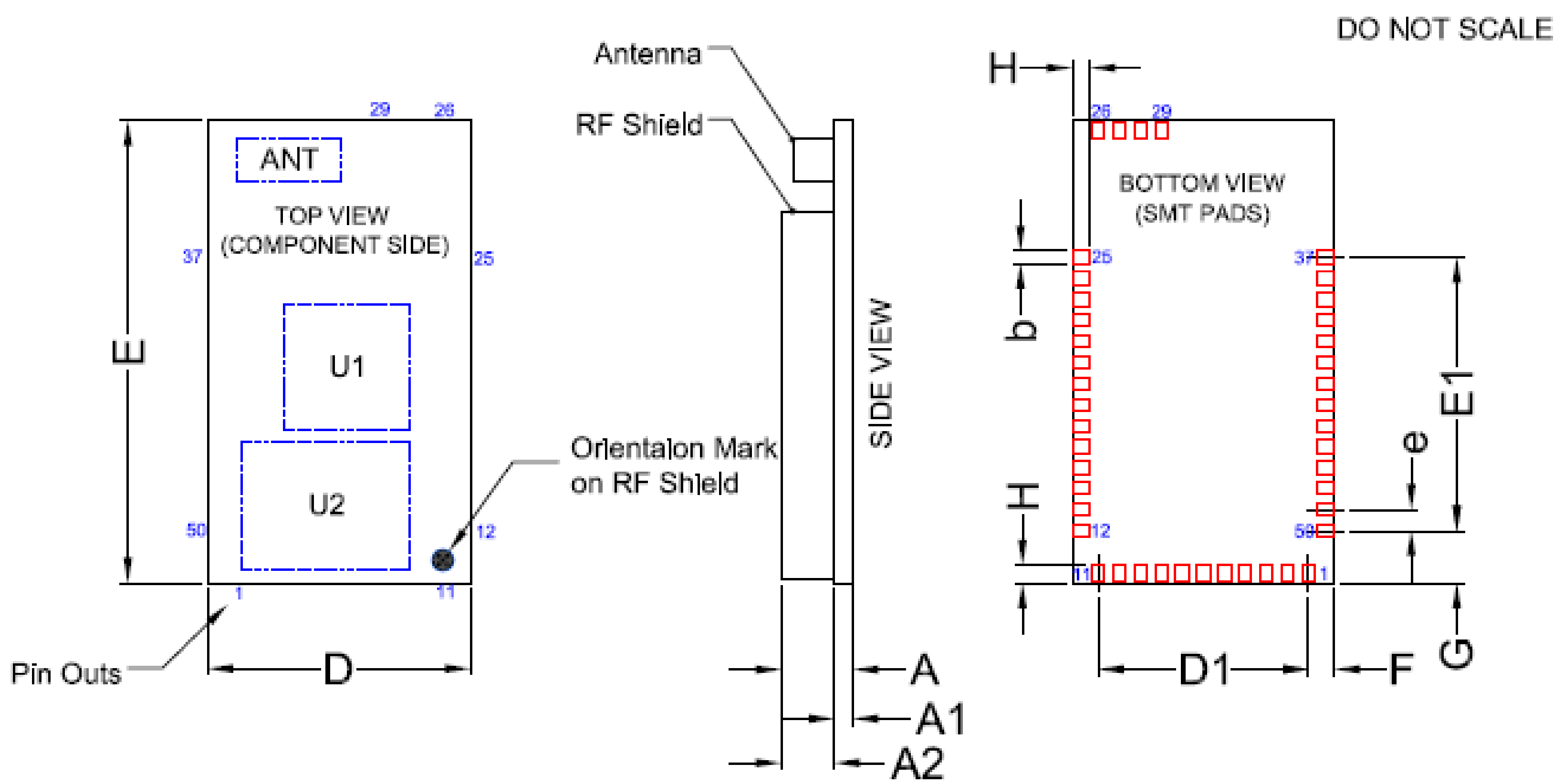
BTD-R-001-POL-03
Page 2 of 2

BTM410

THIRD ANGLE PROJECTION

Recommended Land Pattern/Decal

DO NOT SCALE

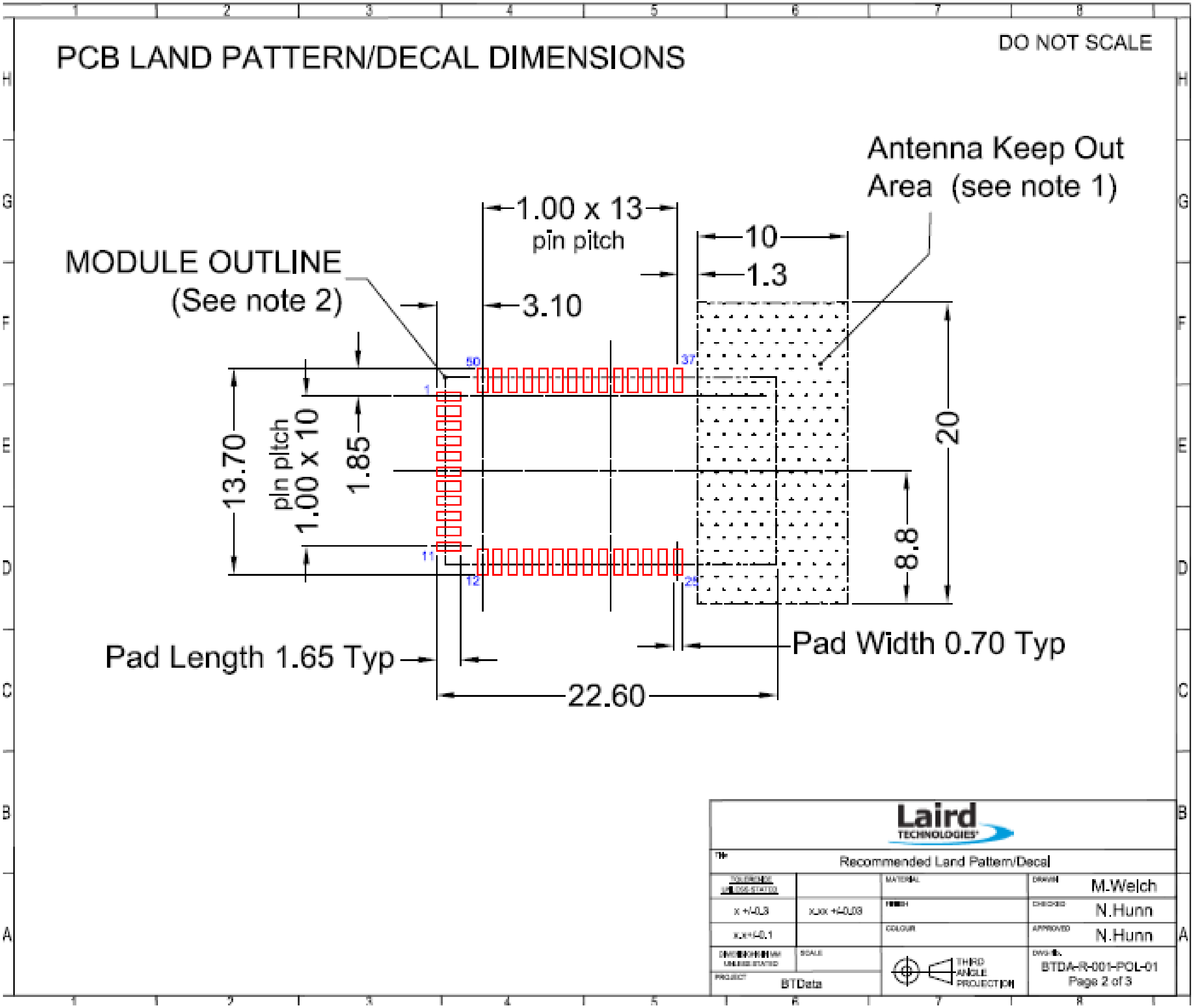


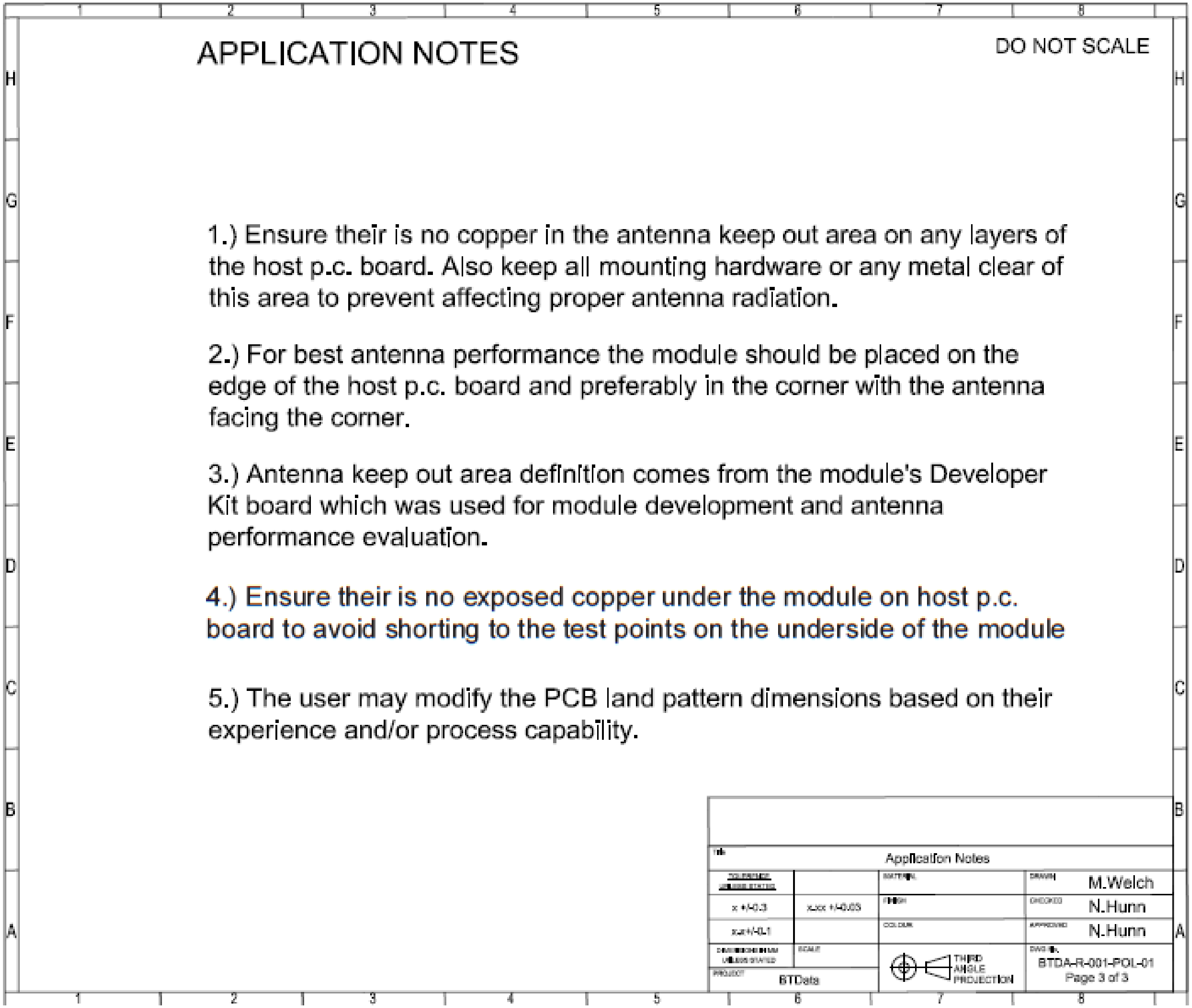
| Description | BTM411 | | | |
|---|---|---|---|---|
| Size | 22.0 x 12.5 x 2.9mm | | | |
| Pitch | 1mm | | | |
| Dimension | Minimum | Typical | Maximum | Notes |
| A | 3.1 | 3.4 | 3.7 | |
| A1 | 0.8 | 0.9 | 1.0 | PCB Thickness |
| A2 | 2.3 | 2.5 | 2.7 | RF Shield Height |
| b | 0.55 | 0.60 | 0.65 | Global pad width |
| D | 12.37 | 12.5 | 12.63 | |
| E | 21.37 | 22.0 | 22.63 | |
| e | | 1.0 | | Global pitch |
| D1 | | 10.0 | | |
| E1 | | 12.0 | | |
| F | | 1.25 | | Pad Center to Board edge |
| G | | 2.50 | | Pad Center to Board edge |
| H | | 0.85 | | Global length of pad to edge of board |
| Units | mm | | | |

Laird TECHNOLOGIES

Title: Module Package Dimensions

| | | | |
|---|---|---|---|
| TOLERANCE UNLESS STATED | | MATERIAL | DRAWN M.Welch |
| x +/-0.3 | x.xx +/-0.03 | FINISH | CHECKED N.Hunn |
| x.x+/-0.1 | | COLOUR | APPROVED N.Hunn |
| DIMENSIONS IN MM UNLESS STATED | SCALE | THIRD ANGLE PROJECTION | DWG No. BTDA-R-001-POL-01<br>Page 1of 3 |
| PROJECT BTData | | | |


PCB LAND PATTERN/DECAL DIMENSIONS
DO NOT SCALE
Antenna Keep Out Area (see note 1)
MODULE OUTLINE (See note 2)
1.00 x 13 pin pitch
3.10
10
1.3
20
8.8
13.70
1.00 x 10 pin pitch
1.85
50
37
1
11
12
25
Pad Length 1.65 Typ
Pad Width 0.70 Typ
22.60
Laird TECHNOLOGIES
Recommended Land Pattern/Decal
TOLERANCE UNLESS STATED
x +/-0.3
x.x +/-0.1
x.xx +/-0.03
DIMENSIONS IN MM UNLESS STATED
SCALE
PROJECT
BTData
MATERIAL
FINISH
COLOUR
THIRD ANGLE PROJECTION
DRAWN
M.Welch
CHECKED
N.Hunn
APPROVED
N.Hunn
DWG No.
BTDA-R-001-POL-01
Page 2 of 3
1
2
3
4
5
6
7
8
H
G
F
E
D
C
B
A



## APPLICATION NOTES

DO NOT SCALE

1.) Ensure their is no copper in the antenna keep out area on any layers of the host p.c. board. Also keep all mounting hardware or any metal clear of this area to prevent affecting proper antenna radiation.

2.) For best antenna performance the module should be placed on the edge of the host p.c. board and preferably in the corner with the antenna facing the corner.

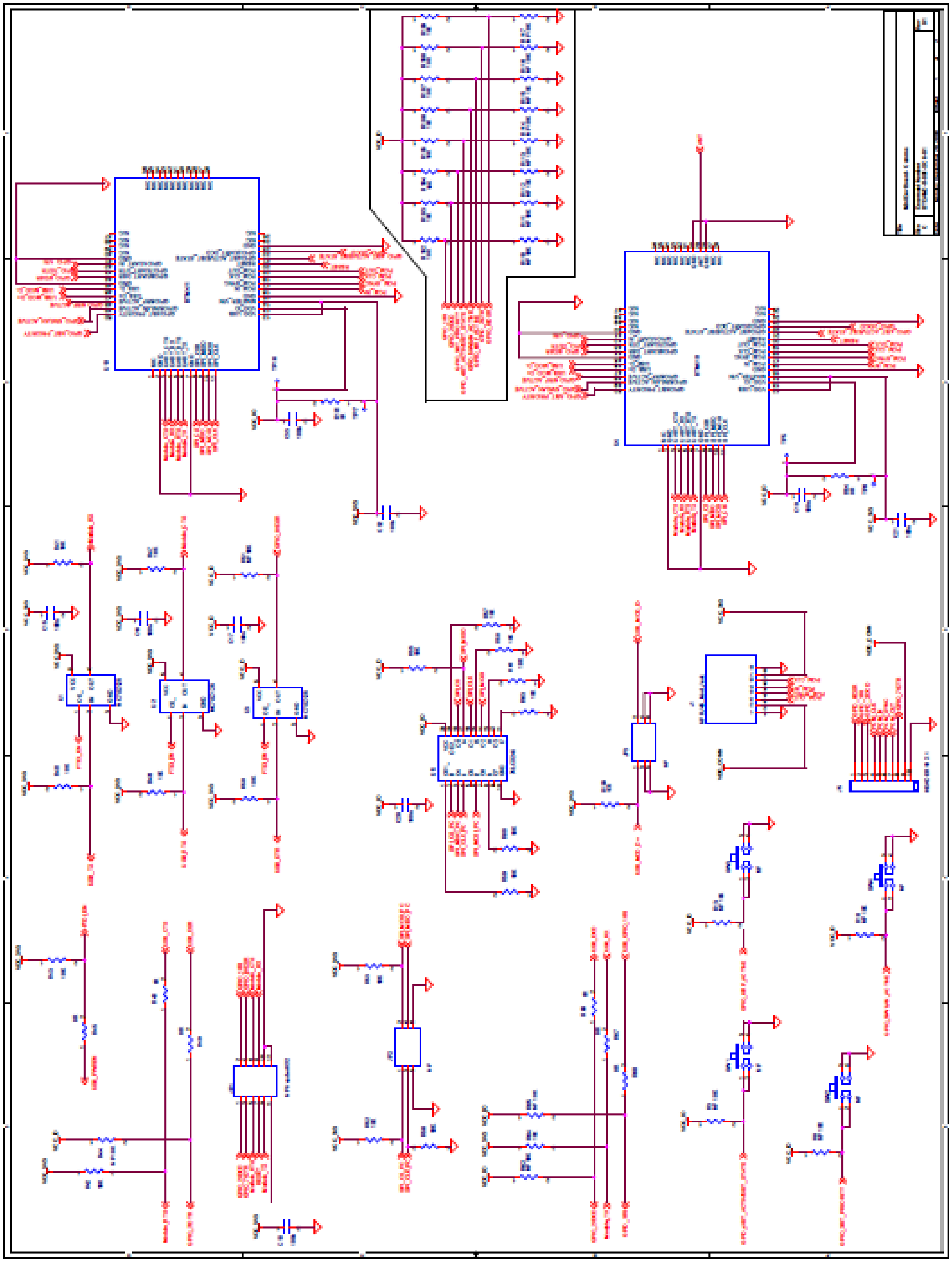
3.) Antenna keep out area definition comes from the module's Developer Kit board which was used for module development and antenna performance evaluation.

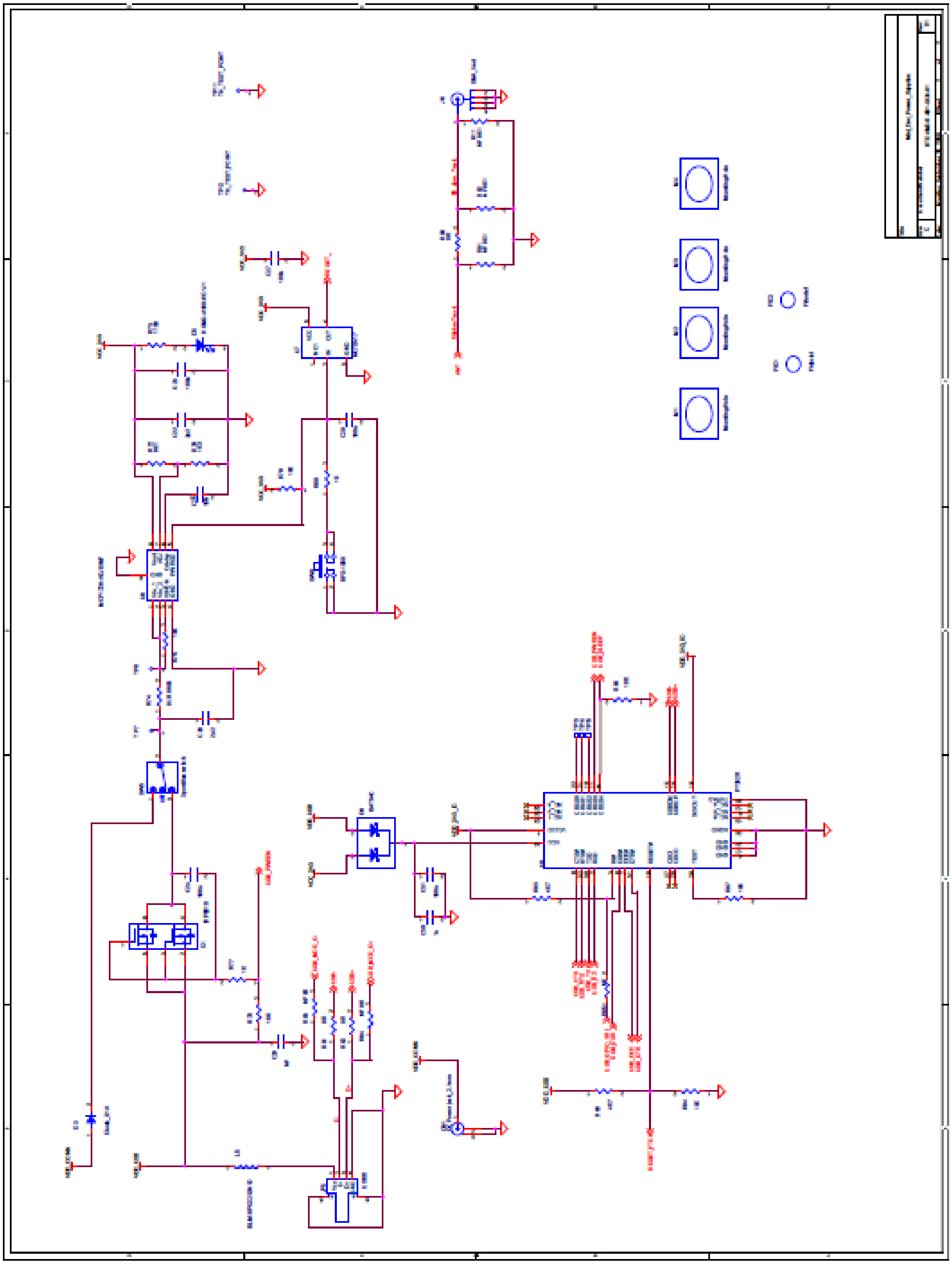
4.) Ensure their is no exposed copper under the module on host p.c. board to avoid shorting to the test points on the underside of the module

5.) The user may modify the PCB land pattern dimensions based on their experience and/or process capability.

| | | | |
|---|---|---|---|
| Title Application Notes | | | |
| TOLERANCE UNLESS STATED | | MATERIAL | DRAWN M.Welch |
| x +/-0.3 | x.xx +/-0.03 | FINISH | CHECKED N.Hunn |
| x.x +/-0.1 | | COLOUR | APPROVED N.Hunn |
| DIMENSIONS IN MM UNLESS STATED | SCALE | THIRD ANGLE PROJECTION | DWG No. BTDA-R-001-POL-01 Page 3 of 3 |
| PROJECT BTData | | | |

# Reference Schematic
# Development Kit





GPIO Pull Up/Down Information

| Categories | Feature | Implementation |
|---|---|---|
| General Purpose Interface | I/O | 8 general purpose I/O pins |

Table 2-4. GPIO Pull Up/Down Information

# Chapter 10
# FCC Regulatory Statements

## FCC and Industry Canada Statements

The OEM’s final equipment user manual must show the following statement:

***This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.***

## Considerations for OEM Integration

Changes or modifications not expressly approved by Laird Technologies could void the user’s authority to operate the equipment.

Designers should note the distinction that the FCC makes regarding portable and mobile devices. Mobile devices are defined as products that are not used closer than 20cm to the human body, whereas portable devices can be used closer that 20cm to the body. A device may be used in portable exposure conditions with no restrictions on host platforms when the averaged output power is less than the low power threshold for an uncontrolled environment ≤ 60/f(GHz) i.e. 25mW for a 2.4Ghz device. The Maximum Power Exposure for the BTM44x has been evaluated and found to comply with the low power threshold for an uncontrolled environment. Refer to FCC document KDB 447498 for more information on RF exposure procedures and equipment authorization policies for mobile and portable devices.

The BTM44x comply with the FCC RF radiation exposure limits set forth for an uncontrolled environment. This device and its antenna must not be co-located or operating in conjunction with any other antenna or transmitter except in accordance with FCC multi-transmitter product procedures. These procedures could require RF exposure evaluation to be re-evaluated on the complete product. The installer of the this module into host equipment must ensure his equipment complies with the FCC RF Exposure requirements set forth in 47 CFR 2.1091 or 2.1093

The BTM44x comply with the RSS-102RF radiation exposure limits set forth for an uncontrolled environment. If this device and its antenna is co-located or operating in conjunction with any other antenna or transmitter RF exposure evaluation should be re-evaluated on the complete product by a qualified test house.
The installer of this module into host equipment must ensure his equipment complies with the Industry Canada RSS-102 RF Exposure requirements.

### BTM440 / BTM442

These modules hold a limited modular approval. Approval with any other antenna configuration or layout other than that approved will necessitate additional radiated emission testing to be performed.

The modules were approved with the following antenna:

- RF Solutions: ANT-24G-WHJ-SMA 0dBi

## FCC Labeling requirement

### BTM440 / BTM442

If the FCC ID is not visible when the module is installed inside another device, then the outside of the device into which the module is installed must also display a label referring to the enclosed module. This exterior label can use wording such as the following: “Contains Transmitter Module FCC ID: PI4410B” or “Contains FCC ID: PI4410B”. Any similar wording that expresses the same meaning may be used.

### BTM441 / BTM443

If the FCC ID is not visible when the module is installed inside another device, then the outside of the device into which the module is installed must also display a label referring to the enclosed module. This exterior label can use wording such as the following: “Contains Transmitter Module FCC ID: PI4411B” or “Contains FCC ID: PI4411B.” Any similar wording that expresses the same meaning may be used.

# Chapter 11
# Declarations of Compliance

## EU Declaration Of Conformity

| | |
|---|---|
| Manufacturer: | Ezurio Ltd |
| Product: | BTM440/2 |
| EU Directive: | RTTE 1995/5/EC |
| Conformity Assessment: | Annex IV |

Reference standards used for presumption of conformity:

| Article Number: | Requirement | Reference standard(s): |
|---|---|---|
| 3.1a | Health and Safety | EN 60950-1:2006 |
| 3.1b | Protection requirements with respect to electromagnetic Compatibility | EN 301 489-1 V1.8.1<br>EN 301 489-17 2.1.1<br>Emissions:<br>EN55022:2006/A1:2000/A2:2006(ClassB)<br>Immunity:<br>EN61000-4-2:1995/A1:1998/A2:2001<br>EN61000-4-3:2002/A1:2002 |
| 3.2 | Means of the efficient use of the radio frequency spectrum | EN 300 328 V1.7.1 (2006-10) |

Declaration:

We, Ezurio Ltd, declare under our sole responsibility that the essential radio test suites have been carried out and that the above product to which this declaration relates is in conformity with all the applicable essential requirements of Article 3 of the EU Directive 1995/5/EC, when used for its intended purpose.

| | |
|---|---|
| Place of Issue: | Ezurio Ltd dba Laird Technologies<br>Saturn House, Mercury Park<br>Wooburn Green<br>HP100HH,<br>United Kingdom<br>tel: +44 (0)1628 858 940<br>fax: +44 (0)1628 528 382 |
| Date of Issue: | October 2009 |
| Name of Authorised Person: | Tim Wheatley, Director of Engineering |
| Signature: | |

EU Declaration Of Conformity

| | |
|---|---|
| Manufacturer: | Ezurio Ltd |
| Product: | BTM441/3 |
| EU Directive: | RTTE 1995/5/EC |
| Conformity Assessment: | Annex IV |

Reference standards used for presumption of conformity:

| Article Number: | Requirement | Reference standard(s): |
|---|---|---|
| 3.1a | Health and Safety | EN 60950-1:2006 |
| 3.1b | Protection requirements with respect to electromagnetic Compatibility | EN 301 489-1 V1.8.1<br>EN 301 489-17 2.1.1<br>Emissions:<br>EN55022:2006/A1:2000/A2:2006(ClassB)<br>Immunity:<br>EN61000-4-2:1995/A1:1998/A2:2001<br>EN61000-4-3:2002/A1:2002 |
| 3.2 | Means of the efficient use of the radio frequency spectrum | EN 300 328 V1.7.1 (2006-10) |

Declaration:

We, Ezurio Ltd, declare under our sole responsibility that the essential radio test suites have been carried out and that the above product to which this declaration relates is in conformity with all the applicable essential requirements of Article 3 of the EU Directive 1995/5/EC, when used for its intended purpose.

| | |
|---|---|
| Place of Issue: | Ezurio Ltd dba Laird Technologies Saturn House, Mercury Park<br>Wooburn Green<br>HP100HH,<br>United Kingdom<br>tel: +44 (0)1628 858 940<br>fax: +44 (0)1628 528 382 |
| Date of Issue: | October 2009 |
| Name of Authorised Person: | Tim Wheatley, Director of Engineering |
| Signature: | |

# Chapter 12
# Bluetooth Approvals

## Subsystem Combinations

This section covers the procedure for generating a new EPL (End Product Listing) on the Bluetooth SIG website. In the instance of subsystems, a member can combine two or more subsystems to create a complete Bluetooth End Product.

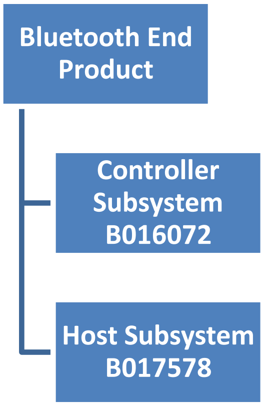
### Subsystem listings referenced as an example:

| Design Name | Owner | QDID number | Link to listing on the SIG website |
|---|---|---|---|
| BTM44x | Laird | B016072 | https://www.bluetooth.org/tpg/QLI_viewQDL.cfm?qid=16072 |
| Interface Express Subsystem | Cambridge Consultants Limited | B017578 | http://bluetooth.org/tpg/QLI_viewQDL.cfm?qid=17578 |

### Assumptions

This procedure assumes that the member is simply combining two subsystems to create a complete End Product. This is achieved by using the EPL (End Product Listing) interface on the Bluetooth SIG website. Diagram 1 shows the basic subsystem combination of a controller and host subsystem. The Controller provides the RF/BB/LM and HCI layers, with the Host providing L2CAP, SDP, GAP, RFCOMM/SPP and any other specific protocols and profiles existing in the Host subsystem listing.

Diagram 1



The following link provides an overview of the EPL system;

https://www.bluetooth.org/technical/qualification/eploverview.htm

For a detailed procedure of how to make an EPL entry, please refer to the following SIG document;

https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=71880

In the case of the Laird and Cambridge Consultants Limited subsystem combination, please search for QDID B016072 in step 4 and then 'Create New EPL' as per step 5. Under step 7, enter QDID B017578 in the box 'Link additional QDLs'.

***Note:***
***Alternatively the member can choose to have a new QDID for the subsystem combination. However it should be noted that this would incur a listing fee and require the subsystem combination to be assessed against the latest qualification requirements.***

## Useful FAQ links

https://www.bluetooth.org/ticketing/view_article.cfm?action=article_comment&aid=284

https://www.bluetooth.org/ticketing/view_article.cfm?action=article_comment&aid=300

https://www.bluetooth.org/ticketing/view_article.cfm?action=article_comment&aid=140

## Additional Assistance

Please contact your local sales representative for further assistance

# Chapter 13
# Ordering Information

| Part Number | Description |
|---|---|
| BTM440 | Bluetooth MP Data Module (external antenna) |
| BTM441 | Bluetooth MP Data Module (with integrated antenna) |
| BTM442 | Bluetooth AT Data Module (external antenna) |
| BTM443 | Bluetooth AT Data Module (with integrated antenna) |
| DVK – BTM440 | Development board with BTM440 module soldered in place |
| DVK – BTM441 | Development board with BTM441 module soldered in place Q2-2009 |
| DVK – BTM442 | Development board with BTM442 module soldered in place |
| DVK – BTM443 | Development board with BTM443 module soldered in place Q2-2009 |

# Chapter 14
# References

[1] “Bluetooth Specification Version 2.1 + EDR [vol3]”, 26 July 2007

http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/

(click on “Core Specification v2.1 + EDR”)

[2] “Serial Port Profile“ Specification

http://www.bluetooth.com/Bluetooth/Technology/Works/SPP.htm

(link at the bottom of page “Need more? View the Serial Port Profile (SPP)”)

[3] “Bluetooth Assigned Numbers”

http://www.bluetooth.com/Bluetooth/Technology/Building/Specifications/

select “Items per page: ALL”, go to end of page, there click on “Assigned Numbers – Baseband”, for a complete list of Profile UUIDs: click on “Assigned Numbers – Service Discovery”

[4] Class of Device Generator: this link might be helpful for creating a particular CoD

http://bluetooth-pentest.narod.ru/software/bluetooth_class_of_device-service_generator.html

Caution: this tool allows selection of more than one minor device classes, so make sure that only one minor device class is select and verify the result with [3] anyway.

[5] “Bluecore 4 External” Data Sheet, Cambridge Silicon Radio (CSR)

http://www.csrsupport.com (log in or new account required)

[6] “Winbond 681360 Codec Board User Guide”, Ezurio Application Note

# Chapter 15
# Glossary of Terms

| Term | | Description |
|---|---|---|
| A2DP | : | Advanced Audio Distribution Profile |
| ACL | : | Asynchronous Connection-Oriented Link |
| ADC | : | Analogue to Digital Converter |
| AGHFP | : | Audio Gateway Hands-Free Profile |
| AT | : | Command prefix, ‘Attention’ |
| AVRCP | : | Audio/Video Remote Control Profile |
| BISM | : | Bluetooth Intelligent Serial Module |
| CoD | : | Class Of Device (also referred to as “device class”) |
| Codec | : | Device capable of encoding / decoding an analogue / digital signal |
| DAC | : | Digital to Analogue Converter |
| DSP | : | Digital Signal Processor |
| DUN | : | Dial-Up Network Profile |
| EIR | : | Extended Inquiry Response |
| eSCO | : | Enhanced Synchronous Connection Oriented Link (used for Audio) |
| FTP | : | File Transfer Profile |
| GOEP | : | Generic Object Access Exchange Profile |
| GPIO | : | General Purpose Input Output |
| HF | : | Hands-free Role of Hands-free Profile (“Hands-free Unit”) |
| HFG | : | Audio Gateway Role of Hands-free Profile (“Hands-free Gateway”) |
| HFP | : | Hands Free Profile |
| HID | : | Human Interface Device Profile |
| HS | : | Headset Role of Headset Profile (“Headset”) |
| HSG | : | Audio Gateway Role of Headset Profile (“Headset Gateway”) |
| HSP | : | Headset Profile |
| I/O (IO) | : | Input/Output |
| Mic | : | Microphone |
| MITM | : | Man In The Middle |
| OPP | : | Object Push Profile |
| PBAP | : | Phone Book Access Profile |
| PT | : | PASS THROUGH Command |
| PWM | : | Pulse Width Modulation |
| SBC | : | Sub Band Codec |
| SCO | : | Synchronous Connection Oriented Link (used for Audio) |
| SLC | : | Service Level Connection |
| SPP | : | Serial Port Profile |
| SSO | : | Serial Stream Oriented |
| SSP | : | Secure Simple Pairing |
| SUI | : | SUBUNIT INFO Command |
| Sxxx | : | S-Register No. xxx |
| TDL | : | Trusted Device List |
| UART | : | Universal Asynchronous Receiver / Transmitter |
| UI | : | UNIT INFO Command |

# Index