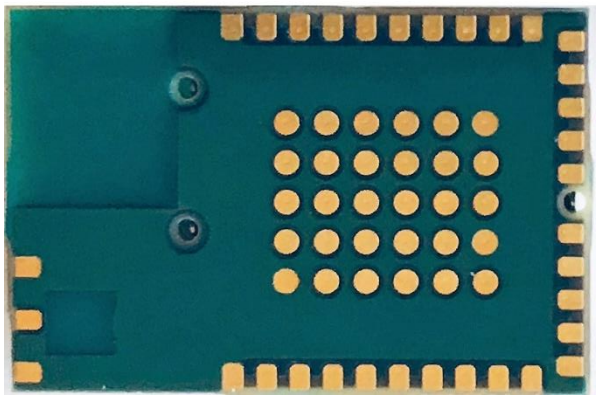
*n*Blue™ BT5.0
AT.s LE Command Set v5.1.1b



BR-LE5.0-S1A Single Mode Low Energy Module
(Actual Size Not Shown)

AT HOME. AT WORK. ON THE ROAD. USING BT5.0 WIRELESS TECHNOLOGY MEANS TOTAL FREEDOM FROM THE CONSTRAINTS AND CLUTTER OF WIRES IN YOUR LIFE.

Table of Contents

Revision History

Rev #	Date	Description
5.0.6.0	11/27/2018	**New Features:** ▪ BT5.0 support – 2Mbps and Long Range PHY support, extended advertising packets, LE Secure Connections pairing. ▪ Up to 8dBm output power now supported. ▪ BRSP throughput up to 50kBps. ▪ Single mode modules can now be connected to 5 devices simultaneously (1 as a peripheral and 4 as a central). ▪ Configurable MTU up to 247 bytes. ▪ USB and NFC support. ▪ Dedicated Over the Air Firmware Update (DFU) service. Firmware updates no longer done through BRSP service. **New Commands:** ▪ AT? – Used to display a list of all available commands. ▪ ATSCFG – Allows reading/writing the entire config in only a few commands. ▪ ATSDPL – Replaces ATSPL to set default output power level. TX_Power is set separately for Advertising and Scanning, RX_Sensitivity is fixed so that parameter has been removed. ▪ ATSCPL – Sets the current power level for an active connection. ▪ ATSUSB – Allows USB serial ports to be enabled/disabled. ▪ ATSNFC – Allows NFC pairing to be enabled/disabled. ▪ ATSPIOL – Allows temporary changing of PIO output level. ▪ ATSDMLE – Allows Channel_Map to be set for ATDMLE and default BRSP mode to be set. ▪ ATSDMTU – Allows the default MTU (Maximum Transmission Unit) to be set. ▪ ATSSPHY – Sets the PHYs the module will accept if a connection PHY update is requested by another device. ▪ ATSCPHY – Requests to change the current PHY for a connection. ▪ ATCRSSI – Allows the RSSI to be continuously sampled when in a connection. ▪ ATPKC – Allows module to respond to a PK_REQ event that requires confirmation to complete a numeric comparison as part of LESC pairing. ▪ ATGRM - GATT Read Multiple. Allows multiple characteristics to be read at a time. This feature is only supported in the client role, so it is not possible to use ATGRM to another BR-LE5.0-S1 or PAN1780-AT module. ▪ ATSBASL – Used to set the BAS battery level when in manual mode. ▪ ATBRSPW – Allows BRSP data to be sent via commands instead of a stream. ▪ ATDTM – Allows device to enter Direct Test Mode for RF testing. ▪ ATTEST – Puts the device into an AT command test mode where ATTXT and ATRXT can be used. ▪ ATBOOT – Used to run the bootloader. ▪ ATSBOOT – Used to configure the bootloader. ▪ ATSDFU – Used to enable/disable the Over the Air Firmware Update service. **New Events:** ▪ EXT_DISCOVERY – Added to support BT5.0 extended advertising. ▪ MTU – Added to support changing the connection MTU (maximum transmission unit). By default, this will now print after a CONNECT event

		once the MTU has been negotiated. Can be disabled using the ATSRM command. ▪ PHYU – Added to support the different BT5.0 PHYs and being able to change the current PHY during a connection. ▪ BRSPD – A new BRSP data event that can be enabled in ATSBRSP to have brsp data be delivered in events instead of a raw stream of data. ▪ RSSI – Similar behavior as before, but now considered an event. **Document Format Changes:** ▪ All dual mode commands have been removed to simplify the document. ▪ The location of some of the events/commands in the document has changed to better group related events/commands together. **Response Format Changes:** ▪ Leading zeroes have been removed from response values. **Optional Parameters:** ▪ All optional parameters will use the default value if not specified. In version 3.5 a change was made so that for commands that store in flash with optional parameters, if an optional parameter was not specified then the current value was used. This has not been implemented in this version. **Address Format Changes:** ▪ The BT5.0 Address and address type have now been combined into a single parameter for all commands/events using an address. See BT5.0 Address Format for more details. **PIO Functionality Changes:** ▪ Pulsing PIO_14 can be used to start/stop advertising. ▪ PIO_7 is now an LED output that can be configured to blink at a desired rate or to show sleep status using ATSZ. ▪ PIO_8 is now an LED output that by default will flash once when a command is received and can also be configured to blink at a desired rate. ▪ ATRFO used to enable RF RX/TX debugging on PIO 8/9 and now defaults to 19/20. The PIOs can also be specified through the command now. ▪ Unused PIOs will now default to a disconnected state instead of a pulled down input state. **Event Format Changes:** ▪ RESET - Reset reason added. FACTORY_RESET will no longer be printed on a factory reset. Factory_Reset is now a value in the RESET event. ▪ DISCOVERY - COD/Addr_Type value removed. ▪ CONNECT – Conn_Type replaced with Conn_Role value. Pair_Status changed to Paired. ▪ PAIR_REQ – Added Conn_Handle value and removed Pairing_Type value. ▪ PAIRED – Added Conn_Handle, Bonded and Security_Level values, removed Pairing_Status value. ▪ PAIR_FAIL – Added Conn_Handle value. ▪ PK_REQ – Added Conn_Handle value. ▪ PK_DIS – Added Conn_Handle and Confirmation_Required values. ▪ GATT_VAL – Added Value_Format value. ▪ BRSP event now reports TX_Mode and RX_Mode. **Command Changes:**

		▪ ATSRM - Individual events can now be disabled using the Response_Event_Flags parameter. ▪ ATCFG? – The order of the response for the command has been changed to match the order of the commands in the document. ▪ ATSFC - Storing of last connected address can now be disabled to prevent writing to flash after every connection to a different device. If automatically storing to flash has been disabled through ATSFC, an individual command change can be stored without saving other modifications by adding the '#' character to the end of the command string, for example ATSN#,NewModuleName<cr>. ▪ ATZ - Module can now be put into a shutdown state using new Shutdown parameter. ▪ ATSZ - Added Shutdown_PIO_State parameter to set the state of the IO in shutdown mode. ▪ ATSAT – Added Privacy_Type and Private_Addr_Cycle_Time parameters. Resolvable private random address no longer considered an address type but can be enabled through the Privacy_Type parameter. ▪ ATSDBLE – Removed the Bridge parameter as this is not supported (was only supported on the dual mode module previously). ▪ ATSPL – Replaced by ATSDPL and now TX_Power is set separately for Advertising and Scanning, RX_Sensitivity is fixed so that parameter has been removed. ▪ ATSPIO – Added Pull_Drive_Strength parameter to allow pull ups/downs for inputs and drive strength for outputs. ▪ ATSLED – 2 more LEDs can now be configured: PIO_7, PIO_8. ▪ ATADC? – Added support for changing Resolution, Gain and Acquisition Time. ▪ ATDSLE/ATDSDLE – ATDSDLE removed, direct advertising can now be done using ATDSLE with new Direct_Addr and Direct_Duty_Cycle parameters. ▪ ATSDSLE – Added new Ad_Types and new params to support extended advertising. ▪ ATSDSTLE – Connected_Ad_Interval can now be set as low as 20ms. ▪ ATSDSDLE – Can now set data for Extended Advertising Data. ▪ ATDILE – Added Extended_Primary_Phys parameter to allow scanning for extended advertising packets. By default, if not specified the module will discover both non-extended and extended ad packets on the 1Mbps PHY. ▪ ATSDILE – Max_Devices can now be set up to 50, Channel_Map parameter added. ▪ ATSDITLE – Timeout now specified in seconds. ▪ ATSDIF – Added Discovery_Type_Format to allow Discovery_Type to be formatted in the standard format, as a bitfield, or as a string. Also added Global_Format parameter to allow DISCOVERY and EXT_DISCOVERY events to be formatted the same instead of as 2 separate events. ▪ ATDMLE – Added Extended_Primary_Phys parameter to allow connecting to a device that is advertising using extended ad packets. ▪ ATSDMTLE – Timeout now specified in seconds. ▪ ATCS? – Response changed: Replaced Conn_Type with Conn_Role, Pairing_Status with Security_Level. Added Conn_Interval, Slave_Latency, Supervision_Timeout, MTU, RX_PHY, TX_PHY, and TX_Power. ▪ ATSPLE – Added LESC parameter to enable/disable LE Secure Connections. Sync_White_List now enabled by default. ▪ ATPKR – Address can no longer be specified, only Conn_Handle.

		▪ ATSDIS – Disabled characteristics will now actually be removed from the service. Any changes to this command now require a reset to take effect. ▪ ATSBAS – When in manual mode the BATT event will no longer fire when a client has registered/deregistered for notifications/indications. Manual mode level is no longer specified in this command and can be set using ATSBASL. ▪ ATSBRSP – Removed OTA Firmware Update Mode from Features as OTA updates are no longer done through BRSP, but through a dedicated DFU service. Added LESC option to Security_Mode. Added TX_Mode and RX_Mode to allow user to set whether to use Serial Port streaming mode or command/event based BRSP.
5.0.6.1	05/29/2019	**Document Changes:** ▪ Updated contact information.
5.0.6.2	01/17/2020	**Document Changes:** ▪ Fixed document formatting issues.
5.0.6.3	06/24/2020	**Document Changes:** ▪ Fixed Sync_White_List default value set to 1 in Pairing Configuration (ATSPLE).
5.0.11.11	10/21/2020	**Document Changes:** ▪ Added support for PAN1780-AT.
5.1.0.0	12/04/2020	**New Features:** ▪ Custom Profile – allows a host connected via UART/USB to add custom GATT services. **New Events:** ▪ CP_READ – used to signal the host that a client is trying to read a custom characteristic or descriptor. ▪ CP_WRITE – used to signal the host that a client is trying to write to a custom characteristic or descriptor. **New Commands:** ▪ ATSCP – Used to set custom profile settings. ▪ ATSCPS – Used to add a custom profile service to the Attribute Table. ▪ ATCPUV – Used to update characteristic and descriptor value in custom profile. ▪ ATCPRR – Used to respond to a CP_READ event. ▪ ATCPWR – Used to respond to a CP_WRITE event. **Changes:** ▪ Upgraded to Nordic nRF5 SDK 17.0.2 and SoftDevice 7.2.0 **Bug Fixes:** ▪ Fixed ATGDC not working for unknown UUIDs ▪ Fixed Buttons sometimes not working after first reset ▪ Fixed module sometimes going into DTM mode on awaking from shutdown via PIO_3 ▪ Fixed bug in Gatt Read Long that would cause buffer overrun and would overwrite OK responses value ▪ Fixed bootloader clearing restart reason, preventing RESET reason 5

		▪ Fixed GATT Write Prepared offset not declared as unit16 **Documentation Changes:** ▪ ATRST - added Delay as optional with default value of 0=Reset Immediately ▪ ATSFC - marked Store_Last_Connected_Address as optional with default value 1=Enabled ▪ ATADC? - Added Oversampling Mode with default value of 0=Disabled
5.1.0.0a	01/21/2021	**Documentation Changes:** ▪ ATSCPS - Fixed characteristic/descriptor Maximum Attribute Value Lengths to be uint16 in little-endian format.
5.1.1.0	02/18/2021	**Changes:** ▪ Added hardware version checks in bootloader to prevent updating to firmware built for wrong module. **Bug Fixes:** ▪ Fixed Device Information Model Number to be PAN1780 for PAN1780 module.
5.1.1.0a	03/23/2021	**Documentation Changes:** ▪ Changed to BT5.0.
5.1.1.0b	03/31/2021	**Documentation Changes:** ▪ Added notes that LEDs (PIO_2/5/7/8) and buttons (PIO_3/4/6/14) are inverted for PAN1780. ▪ Updated URLs.

1 Introduction

"Our clients buy our products because they are reliable and easy to integrate, enabling them to quickly deploy cost-effective wireless solutions."

Mark J. Kramer – CEO of ***Blue*****Radios**

1.1 Scope

This document describes the protocol used to control and configure ***Blue*****Radios *n*Blue™** modules. The protocol is similar to the industry standard Hayes AT protocol used in telephone modems due to the fact that both types of devices are connection oriented. The command set is similar to the one used in ***Blue*****Radios** BT Version 4.0 modules, but many changes have been made to improve the user experience. Users already familiar with the AT.s Command Set v3.X command set will want to read the new command set documentation carefully, paying attention to all the changes that have been made.

Just like telephone modems, the serial modules power up into an unconnected state and will respond to inquiry and connection requests. Then, just like controlling a modem, the host or client can issue AT commands which map to various BT5.0 activities. The command set is extensive enough to allow a host to make connections which are authenticated and encrypted or not. The ***Blue*****Radios *n*Blue™** modules can be configured, commanded, and controlled through simple ASCII strings through the hardware serial UART or over a remote BT5.0 RF connection.

1.2 Background

BT5.0 low energy was designed to enable the development of low complexity, low-cost wireless devices that require minimal power consumption, such as sensors and watches. These devices typically transmit very small data packets at a time, while consuming as little power as possible. BT5.0 specifies two types of implementation for BLE devices: single-mode and dual-mode. Single-mode chips implement the low energy specification and consume just a fraction of the power of classic BT (BR/EDR), allowing the short-range wireless standard to extend to coin cell battery applications. Dual mode chips combine low energy with the power of classic BT and are a standard feature in almost all new BT enabled cellular phones and computers (i.e., gateway devices).

The ***Blue*****Radios *n*Blue™** modules are BT5.0 compliant. The modules are designed to be built into an embedded device and to provide a simple, reliable, and low-cost API interface. The module is designed to integrate with a wide range of applications and platforms.

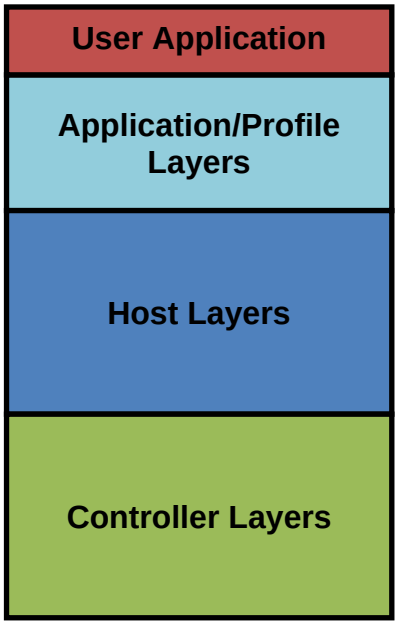
The ***n*Blue™** modules use a proprietary GATT profile developed by BlueRadios to stream data; it is not an official BT profile. BlueRadios BRSP serial port implementation simplifies the user experience, allowing users to stream data similar to the way the official BT Serial Port Profile (SPP) works on BR/EDR devices. It allows the ***n*Blue™** modules to behave very similar to the legacy BlueRadios ATMP modules.

1.3 LE Protocol Stack

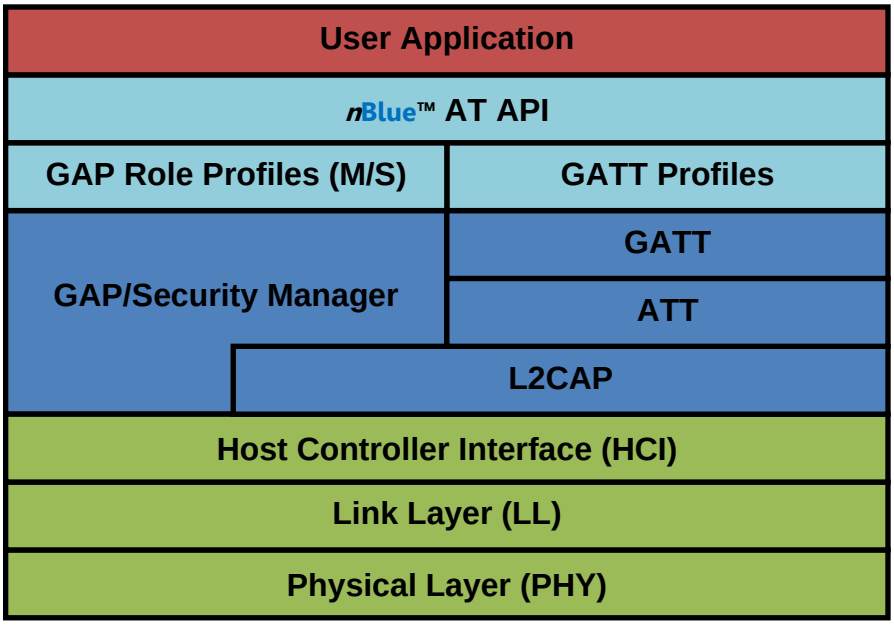
The BT5.0 low energy protocol stack is built similarly to the BR/EDR stack, with a controller layer at the bottom, followed by a host layer, and finally the profile and application layers. The GAP and GATT layers and above are the only layers that should concern the user. The GAP layer controls device discovery, advertising, and connections. The GATT layer handles the exchanging of data.

The *n*Blue™ modules contain a full stack all the way up to the application layer as seen below. The ATBLE API contains all the necessary commands to enable the user to quickly and easily develop a fully featured BLE application.

Simplified Stack



Detailed Stack



BT5.0 New Features

1.3.1 New LE Physical Layers (PHYs)

BT5.0 supports 3 different LE physical layers (PHYs): 1Mbps Standard (1M), 2Mbps High Speed (2M) and 125kbps Coded Long Range (LR). Non-extended advertising (4.0 advertising) is done on the 1M PHY only, whereas the new 2M and LR PHYs can be used for extended advertising. Once connected any of the 3 PHYs can be used if both sides of the connection support it.

1M: The 1Mbps PHY is the standard PHY used by BT4.0 and must be supported by a BT5.0 device.

2M: The 2Mbps PHY is a new optional PHY that provides double the speed of the 1M PHY. This increase in speed may not be noticeable when streaming over BRSP, as our testing has shown both 1M and 2M PHYs maxed out at ~50kBps when streaming in one direction. But if streaming large amounts of data bi-directionally the 2M PHY will provide the highest throughput (~45kBps vs ~35kBps). Regardless of the speed increase, the 2M PHY can be a better choice in a short distance environment. Due to the increased symbol rate of the 2M PHY the radio on time will be reduced since the data is being sent in a shorter amount of time, but this will also decrease the link budget so it will have shorter range than either the 1M or LR PHYs.

LR: The 125kbps Coded PHY is a new optional PHY that provides 4x the range by using Forward Error Correction (FEC). The increased range comes at the cost of throughput since the coding scheme replaces each bit with 8 bits. When streaming over BRSP the maximum data rate in one direction will be ~5kBps and ~4kBps when streaming bi-directionally.

1.3.1.1 BRSP Throughput for Different PHYs

Measurements were taken when streaming either with the UART at 921600 baud or through USB with a default MTU of 247 and default connection interval of 20ms. Using a lower baud rate for the UART will limit the throughput to the baud rate itself.

	Single Direction	Bi-Directional
1M	50 kBps	35 kBps
2M	50 kBps	45 kBps
LR	5 kBps	4 kBps

1.3.2 Extended Advertising

When advertising using the new extended advertising format up to 255 bytes can be sent in one advertising packet. Extended advertising takes place in 2 stages and a different PHY can be used for each stage. First ADV_EXT_IND packets are sent out on the primary PHY on the standard advertising channels (37, 38 and 39). These packets do not include any advertising data but indicate to the receiver what secondary PHY and channel the extended advertising packet will be sent on. The actual extended advertising packet will go out on one of channels 0-36, which are the same channels used for data during a connection. For the primary PHY either 1M or LR can be selected, but the 2M PHY cannot be used as the primary PHY. For the secondary PHY any PHY may be selected. If a connection is made to a device that is advertising using a connectable extended advertisement, then the secondary PHY is the PHY that the connection will be established on.

1.3.3 LE Secure Connections Pairing (LESC)

LE Secure Connections Pairing uses the Elliptic-Curve Diffie-Hellman (ECDH) algorithm for key generation during the pairing process to secure it from passive eavesdroppers in all instances. Both devices that are pairing must support LESC for it to be used. For protection against Man in the Middle attacks authentication should still be enabled. When authentication is enabled and the IO_Capabilities support it, a passkey will need to be confirmed on both sides during the pairing process when using LESC. This is a slightly different process than the passkey display and passkey response used when LESC is not enabled.

2 Important Notes – Please Read Prior To Continuing

2.1 Important Notes

- **The module can be powered in normal voltage mode (1.7-3.6V) or high voltage mode (2.5-5.5V). In normal voltage mode, 1.7-3.6V is connected to both VDD and VDDH. In high voltage mode, 2.5-5.5V is connected only to VDDH and in this case VDD will become the output of the internal VDDH regulator (AT.s firmware sets the VDDH output to 3.3V, but the 52840 defaults to 1.8V). In either power mode the voltage on the VDD pin will be the IO voltage, so the IO voltage can never be higher than 3.6V and is not 5V tolerant even when using VDDH. Due to an issue with Rev 1 of the nRF52840, high voltage mode can only be used under the following conditions: the VDDH internal DCDC converter cannot not enabled (default LDO mode only), no current is drawn from the VDD pin during power up and the VDDH rise time to 3V is < 1ms. This will be fixed in the next nRF52840 revision. See Errata 197 and 202 of the nRF52840 Rev 1 Errata for more info:** https://infocenter.nordicsemi.com/index.jsp?topic=%2Ferrata_nRF52840_Rev1%2FERR%2FnRF52840%2FRev1%2Flatest%2Ferr_840.html

- **If migrating from a pre-5.0 module, please thoroughly read the Revision History to learn about the changes made since the previous AT.s Command Set 3.5.**

- **Over the air updates are enabled by default through the DFU service. If you do not want to allow updating the module over the air, this feature can be disabled through ATSDFU.**

- **Remote command mode is enabled by default, allowing AT commands to be sent to the device over the air through the BRSP service. If you do not want remote command mode enabled, it can be disabled using ATSBRSP.**

- Before proceeding, use the ATV? command to read the firmware version of your modules. Make sure the first two digits of the module's firmware version matches the first two digits of the version of this document. If the module is running older firmware, newer firmware or the command set document corresponding to the older firmware can be found on blueradios.com/forum.

- The BT5.0 Address and address type have now been combined into a single parameter for all commands/events using an address. See BT5.0 Address Format for more details.

- BRSP throughput can now reach speeds of up to 50kBps between two BR-LE5.0-S1 or PAN1780-AT modules when the MTU is set to 247 bytes. When connecting a BR-LE5.0-S1 or PAN1780-AT to an older BR-LE4.0 module the MTU will be 23 and the maximum BRSP throughput will be 1.3kBps.

- To achieve higher data throughput the BR-LE5.0-S1 and PAN1780-AT module supports higher MTUs (Maximum Transmission Units) than the default value of 23 used by our older modules. To use a higher than default MTU two connecting devices must perform an MTU exchange to find out what MTU both sides can support. Once this exchange is complete and an MTU has been set an MTU event will be sent. An MTU exchange is a GATT request, so GATT commands cannot be sent until the MTU event has occurred. If both devices being connected have their MTU set to 23 which is the minimum, then an MTU exchange will not be attempted and the MTU event will never occur.

- Any events that are unused or unwanted can now be disabled using the ATSRM command.

2.2 Backwards Compatibility

- To provide the best firmware architecture, design, and future profile support 100% backwards compatibility between releases cannot be guaranteed.

2.3 Known Issues

Version 5.0.2.0:

- When pairing with another device for the first time, the internal pairing database does a GATT read of the Central Address Resolution characteristic. This GATT read occurs at the time the PAIRED event is sent and completes 30-50ms later (when using the default connection interval of 20ms). If a GATT command is sent in this time frame it will fail with ERROR,12 indicating a GATT request is already pending.

2.4 Related Applications

- *n*Blue Programmer - *n*Blue™ Programmer (*n*BP) is a Windows application that allows firmware to be updated on all *Blue*Radios® *n*Blue™ BT4.0/5.0 modules. Updates can be performed through the module's UART, USB and Over the Air (OTA) through a BLE connection (see the OTA updates section for additional requirements.)

- *n*Blue iBeacon Configuration Tool - *n*Blue™ iBeacon Configuration Tool (*n*BiBCT) is a Windows application that allows *Blue*Radios® *n*Blue™ BT4.0/5.0 modules to be configured as iBeacons.

- *n*BlueTerm – *n*BlueTerm is an iOS application that allows an iOS device to communicate with all *Blue*Radios® *n*Blue™ BT4.0/5.0 modules over BRSP in both data and remote command modes. It is available to download for free from the Apple App Store.

2.5 Related Documents

- *n*Blue Module User's Guide
- *n*Blue BR-EVAL-5.0-S1 Quick Start Guide
- *n*Blue Programmer User's Guide
- *n*Blue iBeacon Configuration Tool User's Guide

3 Hardware Notes

This section is meant to provide a summary of the hardware specifications and details of specific AT.s hardware behavior. See the nBlue Module User's Guide for detailed hardware specifications.

The BR-LE5.0-S1 utilizes the Nordic nRF52840 SoC, for detailed specifications see the product specification: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fps_nrf52840%2Fkeyfeatures_html5.html

3.1 Power Modes

**** IMPORTANT **: The module can be powered in normal voltage mode (1.7-3.6V) or high voltage mode (2.5-5.5V). In normal voltage mode, 1.7-3.6V must be connected to both VDD and VDDH. In high voltage mode, 2.5-5.5V is connected only to VDDH and in this case VDD will be the output of the internal VDDH regulator (AT.s firmware sets the VDDH output to 3.3V, but the 52840 defaults to 1.8V). In either power mode the voltage on the VDD pin will be the IO voltage, so the IO voltage can never be higher than 3.6V and is never 5V tolerant even when using VDDH. See the Internal Regulator Diagram on the next page for a visual on how the regulators connect.**

**** IMPORTANT **: Due to an issue with Rev 1 of the nRF52840, high voltage mode can only be used under the following conditions: the VDDH internal DCDC converter cannot not enabled (default LDO mode only), no current is drawn from the VDD pin during power up and the VDDH rise time to 3V is < 1ms. This will be fixed in the next nRF52840 revision. See Errata 197 and 202 of the nRF52840 Rev 1 Errata for more info: https://infocenter.nordicsemi.com/pdf/nRF52840_Rev_1_Errata_v1.1.pdf**

*** VBUS *:** To use the BR-LE5.0-S1A as a USB peripheral, 5V must be supplied on the VBUS pin. The VBUS supply is internally regulated to 3.3V but is only used for the USB signaling interface and USB detection. The rest of the USB peripheral is powered through the main power supply, so power must still be supplied through VDDH or VDD depending on what power mode is being used. When supplying power from a USB source only, VBUS must be connected to VDDH if USB is to be used.

3.2 Electrical Specifications Summary

Item	Specifications
Supply voltage (VDD)	1.7-3.6 V
VDD Supply rise time (0V to 1.7V)	60ms
Supply voltage (VDDH – Optional)	2.5-5.5 V
VDDH Supply rise time (0V to 3.7V)	100ms
Supply voltage (VBUS - Optional)	4.35-5.5 V
Supply ripple	100 mV Max
Max I/O pin voltage	VDD + .3V, 3.9V Max **(Not 5V Tolerant)**
Ambient Temperature Range	-40 – 85 °C

3.3 Power-up and Reset

There are no strict requirements for power up timing. The module is ready to receive commands once the RESET event is sent out of the UART or USB Serial Port. This event is sent over the UART approximately 850ms after power on. To reset the module, the RESET line must be pulsed low for at least 1µS.

3.4 Default IO States

PIOs 2,5,7,8,16 (RTS) and 17(TX) default to outputs. PIOs 3,4,6,14,15 (CTS) and 18 (RX) default to inputs with 3,4,6 and 14 pulled low for the BR-LE5.0-S1 or high for the PAN1780-AT. All other inputs default to the disconnected state.

3.5 PIO Functions

NOTE: ***The LEDs (PIO_2/5/7/8) and buttons (PIO_3/4/6/14) on the PAN1780-AT are inverted compared to the BR-LE5.0-S1A.***

3.5.1 PIO_2/5/7/8 Status Indicators Outputs

PIO_2 and PIO_5 are defaulted to outputs used to indicate the current status of the module. By default, they will behave in the following manner. PIO_2 will pulse at a configurable duty cycle and rate when the module is connected to another device. PIO_5 will pulse at a configurable duty cycle and rate when the module is advertising, discovering, or connecting. When the module is idle both PIOs will output logic low for the BR-LE5.0-S1 or logic high for the PAN1780-AT. Both can be configured using the ATSLED/ATSPIO commands.

PIO_7 can be configured via the ATSZ command to indicate when the module is in sleep mode or low power mode. PIO_8 will flash when a command is received to confirm communication with the dev board. PIO_7/8 can also be overridden to blink at a user specified rate using ATSLED.

3.5.2 PIO_3 - Sleep Mode Toggle Input

PIO_3 is used to toggle the module in and out of sleep mode. PIO_3 needs to be pulsed high for the BR-LE5.0-S1 or low for the PAN1780-AT for at least 1ms. Since the UART is shutdown when the device enters sleep mode, RTS will be set high when sleep mode is enabled, and upon waking up RTS will go low. ATSZ can be used to also configure PIO_5 to go low on the BR-LE5.0-S1 or high on the PAN1780-AT when the module is sleeping and high on the BR-LE5.0-S1 or low for the PAN1780-AT when it is awake.

If PIO_3 is held high on the BR-LE5.0-S1 or low for the PAN1780-AT for 500ms, shutdown will be enabled, and the module will enter its lowest power state where it can only be woken back up by a reset or by pulsing PIO_3 (which will trigger a reset). In this state the UART and all BT5.0 functionality will be disabled, but the current state of the PIOs at the time of the shutdown will be maintained.

3.5.3 PIO_4 - Factory Reset / Connection Control Input

PIO_4 has multiple purposes. First, it can be used to factory reset the module by setting it to VDD on the BR-LE5.0-S1 or GND on the PAN1780-AT during power up or reset and holding it at VDD on the BR-LE5.0-S1 or GND on the PAN1780-AT until “<cr_lf>FACTORY RESET<cr_lf>” is printed from the UART. Secondly, it can be used as a hardware shortcut to reconnect and disconnect when pulsed high on the BR-LE5.0-S1 or low for the PAN1780-AT for at least 1ms.

- Pulsing PIO_4 while not connected will perform an ATDMLLE to connect to the last connected device.
- If already connecting, pulsing PIO_4 will perform an ATDC to cancel connecting.
- Pulsing PIO_4 while connected will perform an ATDH to disconnect.
- The module will respond with the same response as if the command was entered through the UART.

3.5.4 PIO_6 - BRSP Comm Mode Toggle Input

PIO_6 can be used as a hardware shortcut for toggling between command mode and data mode when connected by pulsing it high on the BR-LE5.0-S1 or low for the PAN1780-AT for at least 1ms. The module will respond with the same response as if the command was entered through the UART.

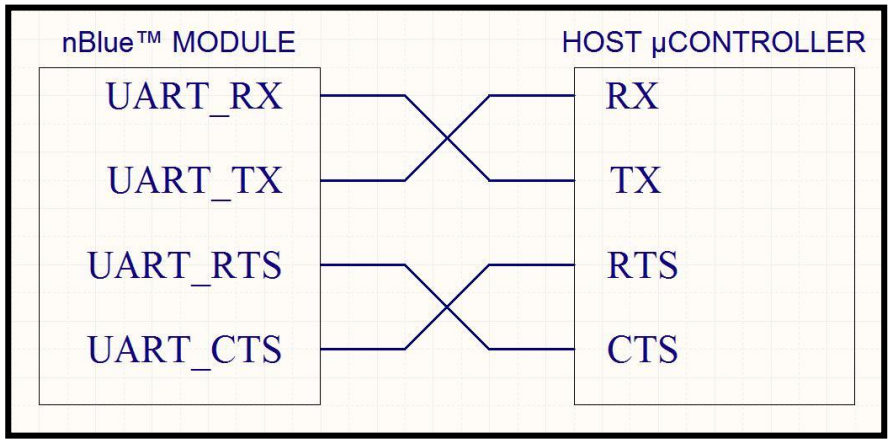
3.5.5 PIO_14 - Firmware Upgrade Mode / Advertising Control Input

PIO_14 has multiple purposes. First, it can be used to manually put the module into firmware upgrade mode by setting it to VDD on the BR-LE5.0-S1 or GND on the PAN1780-AT during power up or reset by holding it at VDD on the BR-LE5.0-S1 or GND on the PAN1780-AT until the "nBoot" message is sent from the UART. Secondly, it can be used as a hardware shortcut to control advertising when pulsed high on the BR-LE5.0-S1 or low for the PAN1780-AT for at least 1ms.

- Pulsing PIO_14 while not advertising will perform an ATDSLE to start advertising.
- If already advertising, pulsing PIO_14 will perform an ATDC to cancel advertising.
- The module will respond with the same response as if the command was entered through the UART.

3.6 UART Interface

UART_TX, UART_RX, UART_RTS and UART_CTS form a conventional asynchronous serial data port. Two-way hardware flow control is implemented by UART_RTS and UART_CTS. These signals operate according to normal industry convention. The signaling levels are nominal 0V and VDD and are inverted with respect to the signaling on an RS232 cable. The interface is programmable; the baud rate, stop-bits, parity, and flow control can all be configured through the ATSUART command. If the module is not connected to another UART, RX should be pulled high to keep it from floating.



3.7 USB Interface

In addition to the UART, the module can also be controlled using AT commands through a USB CDC ACM virtual serial port. The module can be controlled through this port through AT commands and events in the same way it can over the UART.

4 Lowering Power Consumption

4.1 Sleep Mode

AT.s modules are controlled through AT commands sent to the module's UART (although they can also receive commands over the air using BRSP Remote Command Mode). To receive commands and data via the UART, the UART needs to be active and for the UART to be active the module's microcontroller needs to be in the active state. Sleep mode can be used to reduce the amount of time the module's microcontroller is in the active state by allowing the user to tell the module when the UART needs to be active.

When sleep mode is enabled the module will enter the lowest possible power state it can based on the current state and configuration of the module. By default, the module will not be in sleep mode upon power up, but for maximum power savings it should be put into sleep mode as soon as possible after all configuration is complete and kept in sleep mode as often as possible per the application.

While in sleep mode the module will not be able to receive data on the UART but will still be able to handle RF tasks. This allows any active connection requests, discovery request, advertising requests, or connections to be maintained in the background, with the module sleeping in between RF events. The module can still receive incoming data over the air and output it on the UART while it is in sleep mode.

Sleep mode will not affect the operation of the USB serial port.

4.1.1 Enabling/Disabling Sleep Mode

Sleep mode can be enabled by executing the ATZ command or by pulsing PIO_3 high on the BR-LE5.0-S1 or low for the PAN1780-AT while the module is awake. ATSZ can also be used to configure the module to automatically enter sleep mode on power up or after a reset.

Sleep mode can be disabled by pulsing PIO_3 high on the BR-LE5.0-S1 or low for the PAN1780-AT while the module is asleep. ATSZ can be used to configure the module to wake up in response to a high to low transition on the UART_RX line. This allows the module to be woken up by sending a character over the UART, but this character will not be received by the module's UART, as it cannot wake up fast enough to receive this character.

PIO_3 needs to be pulsed high on the BR-LE5.0-S1 or low for the PAN1780-AT for at least 1ms to toggle in and out of sleep mode.

4.1.2 UART and Sleep Mode

When the module is in sleep mode it will not be able to receive data on the UART, so it will not be able to accept any data or AT commands. Since the UART receiver is shutdown when the device enters sleep mode, RTS will be set high when entering sleep mode, and upon waking up it will go low.

Keep in mind that only the UART receiver is disabled by sleep mode, not the UART transmitter. The module can still receive incoming data over the air and output it on the UART while it is in sleep mode. Events will also be sent while in sleep mode, so for example an ATDILE command could be executed followed by an ATZ and the DISCOVERY events will still be sent, even though the module is in sleep mode.

4.1.3 Remote Commands and Sleep Mode

Since connections are maintained in the background while sleep mode is enabled, modules can be controlled through AT commands sent over BRSP in Remote Command Mode. See the ATMRC command for more information.

4.1.4 Shutdown Mode

If PIO_3 is held high on the BR-LE5.0-S1 or low for the PAN1780-AT for 500ms, shutdown will be enabled, and the module will enter its lowest power state where it can only be woken back up by a reset or by pulsing PIO_3 (which will trigger a reset). In this state the UART and all BT5.0 functionality will be disabled, but the current state of the PIOs at the time of the shutdown will be maintained.

4.1.5 Power Consumption Scenarios

Shutdown
When shutdown is enabled, the module will enter the SYSTEM OFF mode where all internal clocks are disabled, and the module will consume the least amount of current possible: ~0.4 µA.

Sleep Mode Enabled and Idle
When sleep mode is enabled the UART will be disabled and the module will consume ~3.4 µA.

Sleep Mode Enabled and Not Idle
~140 µA when advertising at a 100ms interval at 0dB.
~240 µA when advertising at a 100ms interval at 8dB.

Sleep Mode Disabled and Idle
When sleep mode is disabled the UART will be enabled and the module will consume ~625 µA.

Sleep Mode Disabled and Not Idle
~750 µA when advertising at a 100ms interval at 0dB.
~850 µA when advertising at a 100ms interval at 8dB.

4.2 IO Considerations

Make sure no current is being leaked through any IO. Unused IO can be set to a disconnected state using the ATSPIO command.

If LEDs are connected to PIO 2,5,7,8 (as they are on the dev boards), their blinking rates can be decreased, or they can be completely disabled using ATSLED to lower current consumption.

4.3 Idle State

Keep the module in the idle state whenever RF activity (advertising, discovering, connecting, connected) is not necessary. When not connected, the idle state can be entered by using the ATDC command to cancel any advertising, discovery, or connection requests. If connected an ATDH should be performed first to disconnect, followed by an ATDC. The ATSDBLE command can also be used to make the module default to the idle state instead of the advertising state.

4.4 Output Power / Receive Sensitivity

Use ATSDPL to lower the output power / receive sensitivity to the lowest possible values acceptable for the intended application.

4.5 Advertising Interval

Use the ATSDSTLE command to set the advertising interval to the highest possible values acceptable for the intended application. Be aware that increasing the advertising interval will lower the rate of advertising which will increase the amount of time it may take to connect.

4.6 Higher Connection Interval

Use the ATSDCP and ATSCCP commands to set the connection interval to the highest possible values acceptable for the intended application. Be aware that increasing the connection interval will lower the throughput rate and increase the data latency of the connection.

4.7 Slave Latency

Use the ATSDCP and ATSCCP commands to increase the slave latency. This gives the slave device the option to skip connection events and continue sleeping if it has no data to be sent. Be aware that this will lower the throughput and increase the data latency from master to slave.

4.8 Whitelist

The ATSWL command can be used to add device addresses to a whitelist that can be enabled with the ATSDSLE and ATSDMLE commands to only allow connections to specific devices. This can prevent unwanted connections from unknown devices on the slave side and allow the master side to connect to the first connectable device it sees from a specific set without needing to do a separate discovery first.

4.9 Disable Auto Configuration Flashing

By default, all configuration changes will be automatically stored in flash. Using ATSCF this functionality can be disabled to save the time and energy spent storing the configuration after each change. Configuration can still be manually stored using the ATFC command.

4.10 Steps for Entering the Lowest Power State (Shutdown .4 uA)

These instructions apply to a factory default module. If any settings have been changed, factory reset the module using ATFRST or by setting PIO_4 high on the BR-LE5.0-S1 or low for the PAN1780-AT during reset.

Make sure no current is being leaked through any IO.

1. Put the module in the idle state using the ATDC command.
2. Enable shutdown mode using ATZ,1 or my holding PIO_3 high on the BR-LE5.0-S1 or low for the PAN1780-AT for 500ms.
3. The module will now be drawing ~.4uA.

5 Command Usage Guidelines

5.1 Command Usage

- All commands are entered in the following format: “COMMAND”<cr>.
- All commands are typed exactly as shown in the examples. The commands themselves are not case sensitive, but the arguments may be, depending on the command.
- Commands ending in LE can be entered without the LE since this is a single mode LE command set and there are no classic BT commands. For example, ATDMLE can be entered as ATDM.
- All commands that set configuration parameters will start with the ATS prefix, for example ATSN which sets the module’s name. The configuration command parameters can be read back by adding a “?” to the end of the command, for example ATSN?,<cr>.
- By default, the module configuration will be automatically stored to flash any time an ATS prefixed command is sent. This behavior can be changed using the ATSFC command, allowing automatic flash storage to be disabled. When disabled the ATFC command can be used to manually store the module configuration in flash – this will store all changes to the configuration. To store individual command settings without saving other modifications the ‘#’ character can be added to the end of the command string, for example ATSN#,NewModuleName<cr>.
- All response lines come back in the following format <cr_lf>“RESPONSE”<cr_lf>.
- All parameters are in decimal format, unless otherwise noted.
- The factory default setting of any stored parameters will be identified by a blue background like this.
- Some command parameters are optional and will be identified by a gray background like this. The default value that the parameter will take if not specified will also be identified by a gray background.

5.2 Common Parameter/Response Value Descriptions

- <cr> = 0x0D (carriage return)
- <cr_lf> = 0x0D0A (carriage return, linefeed)
- <Conn_Handle> = Connection Handle, 0-9.
- <Att_Handle> = Attribute Handle, specific to the GATT commands and events, 1-65535.

5.3 BT5.0 Address Format

<Addr> = <BT5.0 Address>:<Address Type>

The BT5.0 Address and address type have now been combined into a single parameter for all commands/events using an address. Within the parameter the address will consist of 12 characters and 1 address type character separated by a ':' character, for example ECFE7E123456:0 for a public address of ECFE7E123456. An address may be entered without the ':' and address type and it will be treated as a public address.

- **Address Types:**
  - 0 = Public Address
  - 1 = Static Random Address
  - 2 = Non-resolvable Private Random Address
  - 3 = Resolvable Private Random Address
  - 4 = Anonymous (address is not advertised)

6 Command Status Responses

All commands with valid syntax will respond immediately with either an OK or an ERROR response. All commands with invalid syntax will not respond with anything. After an OK, any additional response is command specific.

6.1 OK

OK STATUS RESPONSE

Function: All successful commands will respond immediately with an OK response, except for ATRST and ATFRST.

Example(s):

```
RESPONSE: <cr_lf>
          OK<cr_lf>
```

6.2 Error

ERROR STATUS RESPONSE

Function: All unsuccessful commands will respond immediately with an ERROR response.

Response Format: ERROR,<Error Code>

Response Values:

- **Error Code:**
  - 1 = Invalid Parameters
  - 2 = Invalid Role
  - 3 = Invalid State
  - 4 = Invalid Password
  - 5 = Invalid Connection Handle
  - 6 = Configuration Locked
  - 7 = List Error
  - 8 = Hardware Error
  - 9 = No Address Stored
  - 10 = BT5.0 Error
  - 11 = Memory Allocation Error
  - 12 = GATT Request Pending
  - 13 = Not Supported
  - 128 = Unknown Error

Example(s):

```
RESPONSE: <cr_lf>
          ERROR,1<cr_lf>
```

7 Events

7.1 General

7.1.1 Reset (RESET)

RESET EVENT

Function: The reset event will be printed as soon as the module is initialized and ready to receive commands after powering up or being reset.

Event Format: RESET,<Factory_Reset>,<Reason>,<Module_Type>

Event Values:

- **Factory_Reset:**
  0 = Module was not factory reset
  1 = Module was factory reset

- **Reason:**
  0 = Software Reset (ATRST / ATFRST)
  1 = Reset Pin
  2 = Power on Reset
  3 = Watchdog Reset
  4 = CPU Lock-Up Detected
  5 = Shutdown Wake-Up

- **Module_Type:**

BR-LE5.0-S1	S1 Single Mode LE 5.0 Module
PAN1780	Panasonic Bluetooth® Low Energy RF Module

Example(s):

1. Following a reset on an S1 module, the Module_Type is sent:

```
EVENT: <cr_lf>
       RESET,0,0,BR-LE5.0-S1<cr_lf>
```

Note(s):

- *This event will be sent over the UART ~850ms after reset on the S1.*

7.1.2 Bootloader (NBOOT)

BOOTLOADER EVENT

Function: The bootloader event will occur when the module enters the bootloader. The reason will let the user know why the module entered the bootloader. For example, if an Over the Air update is started through the DFU service the bootloader event will occur with reason 2. In this case nothing should be sent over the UART or USB until a RESET event has occurred indicating the firmware update is complete. If USB was being used the port will close when the module enters the bootloader.

Event Format: NBOOT,<Reason>,<Version>,<Module_Type>

Event Values:

- **Reason:**
  - 0 = Module entered the bootloader via the ATBOOT command.
  - 1 = Module entered the bootloader via PIO_14.
  - 2 = Module entered the bootloader via the DFU service.
  - 3 = Firmware Update Continuation – Depending on what components of the firmware are being updated, the update can take place in multiple steps and the bootloader can run multiple times. If this reason is seen, the firmware update is still taking place.
  - 4 = No Valid Application

- **Version:** Bootloader version.

- **Module_Type:**

BR-LE5.0-S1	S1 Single Mode LE 5.0 Module
PAN1780	Panasonic Bluetooth® Low Energy RF Module

Example(s):

```
COMMAND:  ATBOOT<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          NBOOT,0,3,BR-LE5.0-S1<cr_lf>
```

7.1.3 Done (DONE)

DONE EVENT

Function: The done event will be sent when a command that runs in the background times out. If a command is cancelled with ATDC, the DONE event will not be sent.

Event Format: DONE,<Command_Type>,<Completed_Command>

Event Values:

- **Command_Type:**
  1 = Low Energy (LE)
  2 = General
  3 = Continuous RSSI

- **Completed_Command:**
  Command_Type = 1 (LE):
  0 = ATDSLE
  1 = ATDILE
  2 = ATDMLE

  Command_Type = 2 (General):
  0 = Command Complete

  Command_Type = 3 (Continuous RSSI):
  Completed_Command = Conn_Handle of connection continuous RSSI was cancelled on.

Example(s):

1. The advertising timeout is set to 1 second using ATSDSTLE. Advertising is then started using ATDSLE. After 1 second advertising stops which causes the DONE event to print, signaling that the module is no longer advertising:

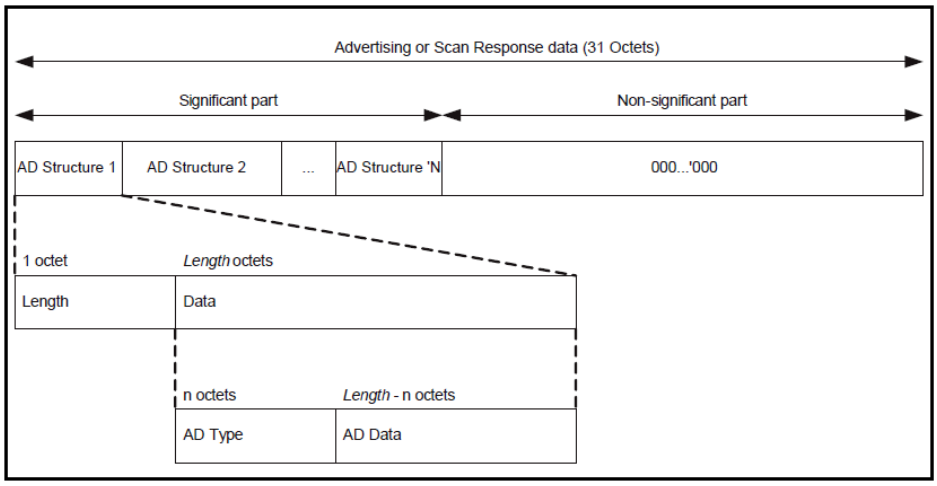
```
COMMAND:  ATSDSTLE,1,160,2048<cr>
RESPONSE: <cr_lf>
          OK<cr_lf
COMMAND:  ATDSLE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
(1 Second Delay)
EVENT:    <cr_lf>
          DONE,1,0<cr_lf>
```

Note(s):

- *A module transitioning from the advertising/connecting (ATDSLE/ATDMLE) state to the connected state will not send a DONE event but will just send the CONNECT event instead.*

7.2 Discovery

Advertising and scan response data is made up of one or more data structures, as seen in the figure below. Each one consists of a different type of data which is defined by the AD/EIR Type. Non-extended advertising and scan response data can be up to 31 bytes long. Extended advertising packets can be up to 255 bytes long.



The full list of AD/EIR Types can be found here:
https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

Descriptions of each AD/EIR Type can be found in the following document:
https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421047

7.2.1 Non-Extended Discovery (DISCOVERY)

NON-EXTENDED DISCOVERY EVENT

Function: The non-extended discovery event will occur when a device is discovered using non-extended advertising packets after the ATDILE command has been issued.

Event Format: DISCOVERY**,**<Discovery_Type>,<Addr>,<RSSI>,
<Data_Structure_Count>,<Data_Structures>

Event Values:

- **Discovery_Type:**
  2 = Connectable, Scannable, Undirected Advertisement
  3 = Connectable, Non-Scannable, Directed Advertisement
  4 = Non-Connectable, Scannable, Undirected Advertisement
  5 = Non-connectable, Non-Scannable Undirected Advertisement
  6 = Scan Response

- **Addr:** The address and address type of the discovered device.

- **RSSI:** The RSSI of the packet received from the device. [-127 – +20 dBm]

- **Data_Structure_Count:** Number of data structures found.

- **Data Structures:** The data structures are returned in the format specified by ATSDIF. This value will be 0, if the Data_Structure_Count is 0.

Example(s):

1. The ATDILE command is used to start an LE discovery and two devices are found, ECFE7E000001 and ECFE7E000002. ECFE7E000001 is advertising Connectable + Scannable, so two DISCOVERY events are sent for it, the first for the advertising data and the second for the scan response data. The ad data contains 4 ad data structures (Flags, TX Power, Slave Connection Interval Range, and BRSP Service), and the scan data contains 1 ad structure (Complete Local Name). ECFE7E000002 is advertising Connectable only, so only one DISCOVERY event is sent for advertising data:

```
COMMAND:  ATDILE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,2,ECFE7E000001:0,-45,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,6,ECFE7E000001:0,-45,1,
          1109426C7565526164696F73303030303031<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,3,ECFE7E000002:0,-45,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DONE,1,1<cr_lf>
```

7.2.2 Extended Discovery (EXT_DISCOVERY)

EXTENDED DISCOVERY EVENT

Function: The extended discovery event will occur when a device is discovered using extended advertising packets after the ATDILE command has been issued.

Event Format:
EXT_DISCOVERY,<Discovery_Type>,<Addr>,<RSSI>,<Primary_PHY>,<Secondary_PHY>,<Sed_ID>,
<Data_ID>,<Data_Structure_Count>,<Data_Structures>

Event Values:

- **Discovery_Type:**
  7 = Extended Connectable, Non-Scannable, Undirected Advertisement
  8 = Extended Connectable, Non-Scannable, Directed Advertisement
  9 = Extended Non-Connectable, Scannable, Undirected Advertisement Scan Response
  10 = Extended Non-Connectable, Scannable, Directed Advertisement Scan Response
  11 = Extended Non-Connectable, Non-Scannable, Undirected Advertisement
  12 = Extended Non-Connectable, Non-Scannable, Directed Advertisement

- **Addr:** The address and address type of the discovered device.

- **RSSI:** The RSSI of the packet received from the device. [-127 – +20 dBm]

- **Primary_PHY:**
  1 = 1M (1Mbps Standard)
  4 = LR (125Kbps Coded Long Range)

- **Secondary_PHY:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  4 = LR (125Kbps Coded Long Range)

- **Set_ID:** Set by the advertiser to distinguish between different advertising sets transmitted by the device. [0-254, 255 if not present]

- **Data_ID:** Set by the advertiser to indicate to the scanner whether the data has changed. [0-4095, not valid if Set_ID is set to 255]

- **Data_Structure_Count:** Number of data structures found.

- **Data Structures:** The data structures are returned in the format specified by ATSDIF. This value will be 0, if the Data_Structure_Count is 0.

Example(s):

1. The ATDILE command is used to start an extended LE discovery on the LR PHY and one device is found, ECFE7E000001. ECFE7E000001 is advertising extended connectable. The ad data contains 6 ad data structures: Flags, TX Power, Slave Connection Interval Range, BRSP Service, Manufacturer Specific Data and Complete Local Name.

```
COMMAND:  ATDILE,4<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

```
EVENT:    <cr_lf>
          EXT_DISCOVERY,7,ECFE7E000001:0,-45,4,2,0,295,6,020106-020A08-
          051208001000-1107796022A0BEAFC0BDDE487962F1842BDA-05FF85000005-
          1109BlueRadios000001<cr_lf>
EVENT:    <cr_lf>
          DONE,1,1<cr_lf>
```

Note(s):

- *An extended advertising event could be up to 600 characters long if the device is advertising the maximum length of 255 bytes with multiple ad data structures and hyphens are enabled in ATSDIF. With hyphens disabled the maximum length is 550.*

7.3 Connection

7.3.1 Connect (CONNECT)

CONNECT EVENT

Function: The connect event will be sent when a connection is established.

Event Format: CONNECT,<Conn_Handle>,<Conn_Role>,<Paired>,<Addr>

Event Values:

- **Conn_Handle:** Connection handle
- **Conn_Role:**
  1 = LE Peripheral
  2 = LE Central
- **Paired:**
  0 = Not Paired
  1 = Paired - PAIRED or PAIR_FAIL event will follow.
- **Addr:** The address and address type of the connected device.

Example(s):

1. The ATDMLE command is used to initiate an LE connection and once connected the CONNECT event is sent. The connected device is an LE device, that is not paired with the module and has an address of ECFE7E000001:0.

```
COMMAND:  ATDMLE,ECFE7E000001:0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
```

7.3.2 Disconnect (DISCONNECT)

DISCONNECT EVENT

Function: The disconnect event will be sent when a connection is terminated.

Event Format: DISCONNECT,<Conn_Handle>,<Disconnect_Reason>

Event Values:

- **Conn_Handle:** Connection handle

- **Disconnect_Reason:**
  0 = Local Disconnect Requested
  1 = Remote Disconnect Requested
  2 = Link Supervision Timeout
  3 = Unacceptable Connection Interval
  4 = MIC (Message Integrity Check) Failure
  5 = Connection Failed to Be Established
  6 = Connection Timing Failure
  9 = Other

Example(s):

1. The ATDH command is used to disconnect and once disconnected the DISCONNECT event is sent, with a reason of Local Disconnect Requested:

```
COMMAND:   ATDH,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           DISCONNECT,0,0<cr_lf>
```

7.3.3 Connection Parameter Update Status (SCCPS)

CONNECTION PARAMETER UPDATE STATUS EVENT

Function: The connection parameter update status event will be sent after issuing an ATSCCP command while in the slave role, letting the user know if the update request was accepted or rejected by the master. This event will also print if Slave_Auto_Update is enabled in the ATSDCP command.

Event Format: SCCPS,<Conn_Handle>,<Status>

Event Values:

- **Conn_Handle:** Connection handle
- **Status:**
  0 = Connection Parameter Update Accepted
  1 = Connection Parameter Update Rejected

Example(s):

1. A connection parameter update is requested on connection handle 0 by a slave module. The SCCPS event is sent, confirming that the update was accepted and then the CPU event is sent when the connection parameters have been updated:

```
COMMAND:  ATSCCP,0,8,8,0,400<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          SCCPS,0,0<cr_lf>
EVENT:    <cr_lf>
          CPU,0,8,0,400<cr_lf>
```

Note(s):

- *The SCCPS event is not guaranteed to always occur before the CPU event.*
- *If the module is in the master role of a connection, the SCCPS event will never be sent.*

7.3.4 Connection Parameter Update (CPU)

CONNECTION PARAMETER UPDATE EVENT

Function: The connection parameter update event will be sent when the current connection parameters have changed.

Event Format: CPU,<Conn_Handle>,<Conn_Interval>,<Slave_Latency>,<Supervision_Timeout>

Event Values:

- **Conn_Handle:** Connection handle
- **Conn_Interval:** Integer value from 6 to 3200 [1.25ms].
- **Slave_Latency:** Integer value from 0 to 499 [connection intervals].
- **Supervision_Timeout:** Integer value from 10 to 3200 [10ms].

Example(s):

1. A connection parameter update is requested on connection handle 0 by a slave module. The SCCPS event is sent, confirming that the update was accepted and then the CPU event is sent when the connection parameters have been updated:

```
COMMAND:  ATSCCP,0,8,8,0,400<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          SCCPS,0,0<cr_lf>
EVENT:    <cr_lf>
          CPU,0,8,0,400<cr_lf>
```

Note(s):

- *The SCCPS event is not guaranteed to always occur before the CPU event.*
- *If the module is in the master role of a connection, the SCCPS event will never be sent.*

7.3.5 MTU Update (MTU)

MTU UPDATE EVENT

Function: The MTU update event will be sent when an MTU exchange has completed. An MTU exchange is a GATT request, so GATT commands cannot be sent until the MTU event has occurred. If both devices being connected have their MTU set to 23 which is the minimum, then an MTU exchange will not be attempted and the MTU event will never occur. In this case GATT commands can be sent immediately after the CONNECTION event.

Event Format: MTU,<Conn_Handle>,<MTU>

Event Values:

- **Conn_Handle:** Connection handle
- **MTU:** 23-247

Example(s):

1. The ATDMLE command is used to initiate a connection and once connected the CONNECT event is sent.

```
COMMAND:  ATDMLE,ECFE7E000001:0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
```

Note(s):

- *If connecting to a paired device, depending on the timing of events after a connection is established the MTU event may occur either before or after the PAIRED event.*

7.3.6 PHY Update (PHYU)

PHY UPDATE EVENT

Function: The PHY update event will be sent when a connection PHY update request has been issued. In the case that the PHY update fails due to the receiving device not supporting one of the requested PHYs the PHYU event will still occur but the PHYs will stay the same.

Event Format: PHYU,<Conn_Handle>,<Status>,<RX_PHY>,<TX_PHY>

Event Values:

- **Conn_Handle:** Connection handle

- **RX_PHY:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  4 = LR (125Kbps Coded Long Range)

- **TX_PHY:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  4 = LR (125Kbps Coded Long Range)

Example(s):

1. A PHY update to 2M is requested on connection handle 0 and succeeds.

```
COMMAND:  ATSCPHY,0,2,2<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PHYU,0,2,2<cr_lf>
```

2. A PHY update to 2M is requested on connection handle 0 and fails, the PHYs stays at 1M.

```
COMMAND:  ATSCPHY,0,2,2<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PHYU,0,1,1<cr_lf>
```

7.3.7 RSSI (RSSI)

RSSI EVENT

Function: The RSSI event will be sent after an RSSI reading has been requested using the ATRSSI? command.

Event Format: RSSI,<Conn_Handle>,<RSSI_Value>

Event Values:

- **Conn_Handle:** Connection handle
- **RSSI_Value:** Absolute receiver RSSI value in dBm. If the RSSI cannot be read, 127. [-127 – +20 dBm, 127]

Example(s):

```
COMMAND:   ATRSSI?,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           RSSI,0,-34<cr_lf>
```

7.4 Pairing

7.4.1 Pairing Request (PAIR_REQ)

PAIRING REQUEST EVENT

Function: The pairing request event will be sent when another device requests pairing and automatic pairing request accept is not enabled. The ATP or ATPLE command is used to accept a pairing request.

Event Format: PAIR_REQ,<Conn_Handle>,<Addr>

Event Values:

- **Conn_Handle:** Connection handle
- **Addr:** The address and address type of the device requesting pairing.

Example(s):

1. An LE pairing request is received from ECFE7E000001 and ATPLE is used to accept the request:

```
EVENT:     <cr_lf>
           PAIR_REQ,0,ECFE7E000001:0<cr_lf>
COMMAND:   ATPLE,0,1<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           PAIRED,0,2,2,ECFE7E000001:0<cr_lf>
```

7.4.2 Paired (PAIRED)

PAIRED EVENT

Function: The paired event will be sent when pairing is successful. The connection is encrypted after this message is received.

Event Format: PAIRED,<Conn_Handle>,<Bonded>,<Authenticated>,<Addr>

Event Values:

- **Conn_Handle:** Connection handle
- **Bonded:**
  0 = Not Bonded
  1 = Previously Bonded Device
  2 = New Bonded Device
- **Security_Level:**
  2 = Encrypted, Not Authenticated
  3 = Encrypted, Authenticated
  4 = Encrypted, LESC Authenticated
- **Addr:** The address and address type of the paired device.

Example(s):

1. LE pairing is initiated using the ATPLE command to connection handle 0. The remote device accepts the command and pairing is completed, triggering the PAIRED event:

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,0,2,2,ECFE7E000001:0<cr_lf>
```

Note(s):

- *Unbonded Pairing States: nBlue modules always request bonding during pairing, but if the remote device does not support bonding then unbonded pairing will occur. In this case pairing is only temporary for the current connection and will not automatically be enabled the next time the remote device is connected.*

7.4.3 Pairing Failed (PAIR_FAIL)

PAIRING FAILED EVENT

Function: The pairing failed event will be sent when a pairing request has failed.

Event Format: PAIR_FAIL,<Conn_Handle>,<Addr>,<Reason>

Event Values:

- **Conn_Handle:** Connection handle
- **Addr:** The address and address type of the remote device.
- **Reason:**
  0 = Pairing Timeout
  1 = Invalid Passkey
  3 = IO Capabilities Cannot Meet Authentication Requirements
  5 = Pairing Not Supported
  6 = Encryption Key Size (If previously paired, then the remote device has unpaired, and pairing will need to be reinitiated.)
  7 = Other/Unknown

Example(s):

1. LE pairing is initiated using the ATPLE command to connection handle 0, but the remote device does not support pairing, triggering the PAIR_FAIL event:

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIR_FAIL,0,ECFE7E000001:0,5<cr_lf>
```

7.4.4 Passkey Request (PK_REQ)

PASSKEY REQUEST EVENT

Function: The passkey request event will be sent when a passkey input is needed for authentication during the pairing process. The ATPKR command is used to respond to this event.

Event Format: PK_REQ,<Conn_Handle>,<Addr>

Event Values:

- **Conn_Handle:** Connection handle
- **Addr:** The address and address type of the remote device.

Example(s):

1. LE pairing is initiated using the ATPLE command to connection handle 0. Authentication is required by one side or the other and based on the module's IO_Capabilities (ATSPLE), PK_REQ is sent to the local module. The remote device will display a passkey. This event is then responded to with an ATPKR command with a passkey of 123456. Once pairing is completed the PAIRED event is sent:

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PK_REQ,0,ECFE7E000001:0<cr_lf>
COMMAND:  ATPKR,0,123456<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,0,2,3,ECFE7E000001:0<cr_lf>
```

Note(s):

- *This event will not occur if a fixed passkey is set with ATSPK as the fixed passkey will be used instead.*

7.4.5 Passkey Display (PK_DIS)

PASSKEY DISPLAY EVENT

Function: The passkey display event will be sent when a passkey needs to be displayed for authentication. When this event is received the Passkey shall be displayed to the user on the local device.

Event Format: PK_DIS,<Conn_Handle>,<Addr>,<Passkey>,<Confirmation_Required>

Event Values:

- **Conn_Handle:** Connection handle
- **Addr:** The address and address type of the remote device.
- **Passkey:** The passkey to be displayed. 6 numeric characters.
- **Confirmation_Required:**
  0 = Passkey just needs to be displayed.
  1 = Passkey needs to be displayed and confirmed by the user. The ATPKC command is used to respond to this event.

Example(s):

1. LE pairing is initiated using the ATPLE command to connection handle 0. Authentication is required by one side or the other and based on the module's IO_Capabilities (ATSPLE), PK_DIS is sent to the local module. The remote device will need to enter a passkey. Once pairing is completed the PAIRED event is sent:

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PK_DIS,0,ECFE7E000001:0,123456,0<cr_lf>
EVENT:    <cr_lf>
          PAIRED,0,2,3,ECFE7E000001:0<cr_lf>
```

2. LE pairing is initiated using the ATPLE command to connection handle 0. Authentication is required by one side or the other and based on the module's IO_Capabilities (ATSPLE), PK_CFM is sent to the local module. This passkey is then confirmed with the ATPKC command. Once pairing is completed the PAIRED event is sent:

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PK_DIS,0,ECFE7E000001:0,123456,1<cr_lf>
COMMAND:  ATPKC,0,1<cr>
EVENT:    <cr_lf>
          PAIRED,0,2,3,ECFE7E000001:0<cr_lf>
```

Note(s):

- *This event will not occur if a fixed passkey is set with ATSPK.*

7.4.6 Address Resolved (RESOLVED)

RESOLVED EVENT

Function: This event will be sent when the module first pairs with a device that is using a private resolvable address type. It will inform the user of the device's public address, which will be used in place of the private address from that point on. Device's using a private resolvable address can change their address often, but after receiving a resolved event the module will automatically resolve any address changes and always reference the device by its public address.

Event Format: RESOLVED,<Private_Addr>,<Public_Addr>

Event Values:

- **Private_Addr:** The private resolvable address that the device is using.
- **Public_Addr:** The device's public address.

Example(s):

1. An LE discovery is performed and a device using a private resolvable address of 7CED99C0E2B3 is found.

```
COMMAND:  ATDILE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,2,7CED99C0E2B3:3,-45,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,6,7CED99C0E2B3:3,-45,1,
          1109426C7565526164696F73303030303031<cr_lf>
EVENT:    <cr_lf>
          DONE,1,1<cr_lf>
```

A connection is made to the device using ATDMLE with an address type of private resolvable (3). Once connected, pairing is initiated and RESOLVED and PAIRED events are sent. The RESOLVED event informs the user that 7CED99C0E2B3's public address is ECFE7E000001. The public address is then used to refer to this device from now on, as seen in the PAIRED event.

```
COMMAND:  ATDMLE,7CED99C0E2B3:3,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,7CED99C0E2B3:3<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          RESOLVED,7CED99C0E2B3:3,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          PAIRED,0,2,2,ECFE7E000001:0<cr_lf>
```

```
COMMAND:  ATDH,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCONNECT,0,0<cr_lf>
```

After disconnecting another discovery is performed. Now that the module is paired to the remote device and the address has been resolved, the public address is shown instead of the private address.

```
COMMAND:  ATDILE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,2,ECFE7E000001:0,-45,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,6,ECFE7E000001:0,-45,1,
          1109426C7565526164696F73303030303031<cr_lf>
EVENT:    <cr_lf>
          DONE,1,1<cr_lf>
```

A connection can now be made to the remote device using the public address.

```
COMMAND:  ATDMLE,ECFE7E000001:0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
```

7.5 GATT Client

7.5.1 GATT Done (GATT_DONE)

GATT DONE EVENT

Function: This event will be sent when a GATT request is completed and report any errors that occurred.

Event Format: GATT_DONE,<Conn_Handle>,<Completed_Command>,<Error>

Event Values:

- **Conn_Handle:** Connection handle

- **Completed_Command:**
  0 = ATGDPS/ATGDPSU
  2 = ATGDC/ATGDCU
  3 = ATGDCD
  4 = ATGR/ATGRU
  5 = ATGW
  6 = ATGRL
  7 = ATGWP
  8 = ATGWPE
  9 = ATGRM

- **Error:**
  0 = No Error
  1 = Invalid Attribute Handle
  2 = Read Not Permitted
  3 = Write Not Permitted
  4 = Invalid PDU
  5 = Insufficient Authentication
  6 = Request Not Supported
  7 = Invalid Offset
  8 = Insufficient Authorization
  9 = Prepare Queue Full
  10 = Attribute Not Found
  11 = Attribute Not Long
  12 = Insufficient Encryption Key Size
  13 = Invalid Attribute Value Length
  14 = Unlikely Error
  15 = Insufficient Encryption
  16 = Unsupported Group Type
  17 = Insufficient Resources
  >128 = Service Specific Error

Example(s):

1. A GATT read is requested on handle 100, which does not exist, so a GATT_DONE is triggered with an Error of Invalid Attribute Handle:

```
COMMAND:  ATGR,0,100<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

```
EVENT:    <cr_lf>
          GATT_DONE,0,4,1<cr_lf>
```

2. A GATT write is requested on handle 19, and after successfully completing a GATT_DONE is sent:

```
COMMAND:  ATGW,0,19,1,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,5,0<cr_lf>
```

7.5.2 GATT Discovered Primary Service (GATT_DPS)

GATT DISCOVERED PRIMARY SERVICE EVENT

Function: This event will be sent for each primary service discovered by an ATGDPS or ATGDPSU command.

Event Format: GATT_DPS,<Conn_Handle>,<Svc_Att_Handle>,<Svc_End_Att_Handle>,<Svc_UUID>

Event Values:

- **Conn_Handle:** Connection handle
- **Svc_Att_Handle:** Service declaration attribute handle. [1-65535]
- **Svc_End_Att_Handle:** Attribute handle of the last attribute in the service. If the discovered service is the last service in a devices attribute table, this value will be 65535. [1-65535]
- **Svc_UUID:** UUID of the discovered service. [16-bit UUID = 4 chars, 128-bit UUID = 32 chars]

Example(s):

1. ATGDPS is issued for connection handle 0 and 6 primary services are discovered: GAP, GATT, DIS (Device Info), BAS (Battery), DFU (Firmware Update) and BRSP. When all the primary services have been discovered a GATT_DONE event is triggered:

```
COMMAND:  ATGDPS,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,1,9,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,10,13,1801<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,14,24,180A<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,25,28,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,29,32,FE59<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,33,65535,DA2B84F1627948DEBDC0AFBEA0226079<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,0<cr_lf>
```

7.5.3 GATT Discovered Characteristic (GATT_DC)

GATT DISCOVERED CHARACTERISTIC EVENT

Function: This event will be sent for each characteristic discovered by an ATGDC or ATGDCU command.

Event Format: GATT_DC,<Conn_Handle>,<Char_Value_Att_Handle>,<Char_Properties>,<Char_UUID>

Event Values:

- **Conn_Handle:** Connection handle.
- **Char_Value_Att_Handle:** Characteristic value attribute handle. [1-65535]
- **Char_Properties: [Hex Formatted]** The properties determine how the characteristic can be used.
  - 0x01 = Broadcast
  - 0x02 = Read
  - 0x04 = Write Without Response
  - 0x08 = Write
  - 0x10 = Notify
  - 0x20 = Indicate
  - 0x40 = Authenticated Signed Writes
  - 0x80 = Extended Properties
- **Char_UUID:** UUID of the characteristic. [16-bit UUID = 4 chars, 128-bit UUID = 32 chars]

Example(s):

1. ATGDCU is used to search for a characteristic with a UUID of 2A00. A read only characteristic is discovered at handle 3, and a GATT_DONE is triggered when the search is complete:

```
COMMAND:  ATGDCU,0,2A00<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,3,2,2A00<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,0<cr_lf>
```

7.5.4 GATT Discovered Characteristic Descriptor (GATT_DCD)

GATT DISCOVER CHARACTERISTIC DESCRIPTOR EVENT

Function: This event will be sent for each characteristic descriptor discovered by an ATGDCD command.

Event Format: GATT_DCD,<Conn_Handle>,<Char_Desc_Att_Handle>,<Char_Desc_UUID>

Event Values:

- **Conn_Handle:** Connection handle
- **Char_Desc_Att_Handle:** Characteristic descriptor attribute handle. [1-65535]
- **Char_UUID:** UUID of the characteristic descriptor. [16-bit UUID = 4 chars]

Example(s):

1. ATGDCD is used to discover the characteristic descriptors of the BAS Battery Level characteristic on a remote device. First, ATGDPSU is used to find the BAS service, which has a start handle of 25 and an end handle of 28:

```
COMMAND:   ATGDPSU,0,180F<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_DPS,0,25,28,180F<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,0,0<cr_lf>
```

   Next, ATGDC is used to find the Battery Level characteristic:

```
COMMAND:   ATGDC,0,25,28<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_DC,0,27,12,2A19<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,2,0<cr_lf>
```

   Last, ATGDCD is used to find the Battery Level characteristic descriptors. The Start_Att_Handle is set to 28, which is the Battery Level characteristic value handle + 1 that was found by ATGDC. The End_Att_Handle is also set to 19, since this is the last handle in the BAS service (thus the last handle in the characteristic), which was found by ATGDPSU:

```
COMMAND:   ATGDCD,0,28,28<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_DCD,0,28,2902<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,3,0<cr_lf>
```

7.5.5 GATT Characteristic/Descriptor Value (GATT_VAL)

GATT CHARACTERISTIC/DESCRIPTOR VALUE EVENT

Function: This event will be sent when GATT data is received, either from a GATT read request or from a notification/indication.

Event Format: GATT_VAL,<Conn_Handle>,<Value_Att_Handle>,<Value_Event_Type>,
<Value_Format>,<Value_Length>,<Value>

Event Values:

- **Conn_Handle:** Connection handle.
- **Value_Att_Handle:** Value attribute handle. [1-65535, 0 if Read Multiple Response]
- **Value_Event_Type:**
  0 = Read Response
  1 = Notification
  2 = Indication
  3 = Read Multiple Response
- **Value_Format:** The format of the value.
  0 = Hex
  1 = 8-Bit Unsigned Decimal
  2 = 16-Bit Unsigned Decimal
  3 = 24-Bit Unsigned Decimal
  4 = 32-Bit Unsigned Decimal
  5 = String
- **Value_Length:** Length of characteristic value in bytes.
- **Value:** Value formatted in hex.

Example(s):

1. ATGR is used to read a value from handle 3. When the operation is complete a GATT_VAL is triggered, returning a 16-byte value:

```
COMMAND:  ATGR,0,3<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_VAL,0,3,0,0,16,426C7565526164696F73303030303031<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,4,0<cr_lf>
```

7.5.6 GATT Long Characteristic/Descriptor Value (GATT_LVAL)

GATT LONG CHARACTERISTIC/DESCRIPTOR VALUE EVENT

Function: This event will be sent when GATT data is received from a GATT read long request. A long read will come back in multiple chunks of 22 bytes at a time, with each consecutive GATT_LVAL returning at an offset 22 bytes further into the whole value.

Event Format: GATT_VAL,<Conn_Handle>,<Value_Att_Handle>,<Value_Offset>,
<Value_Length>,<Value>

Event Values:

- **Conn_Handle:** Connection handle.
- **Value_Att_Handle:** Value attribute handle. [1-65535]
- **Value_Offset:** Offset of partial data into entire value.
- **Value_Length:** Length of partial data in bytes.
- **Value:** Partial data value formatted in hex.

Example(s):

1. ATGRL is used to read a value from handle 54. The value is 33 bytes long, so 2 GATT_LVAL's are returned followed by a GATT_DONE. The first GATT_VAL returns the first portion of the data from offset 0 and the second returns the final portion of the data from offset 22.

```
COMMAND:   ATGRL,0,54<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_LVAL,0,54,0,22,
           112233445566778899001122334455667788990011 22<cr_lf>
EVENT:     <cr_lf>
           GATT_LVAL,0,54,22,11,33445566778899001122 33<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,6,0<cr_lf>
```

7.6 GATT Service

7.6.1 Battery Level Event (BATT)

BATTERY LEVEL EVENT

Function: If BAS is enabled in Automatic mode with Critical_Level set to a non-zero value, the BATT event will fire to alert the application that the battery has dropped below the critical level. It will continue to fire each time the battery percentage drops when below this level. To be notified anytime the battery level drops, Critical_Level can be set to 100. If BAS is enabled in Manual mode, this event will never fire.

Event Format: BATT,<Level>

Event Values:

- **Level:** 0-99 %

Example(s):

```
EVENT:    <cr_lf>
          BATT,99<cr_lf>
```

7.6.2 BRSP Status (BRSP)

BRSP STATUS EVENT

Function: The BRSP status event will be sent when the BRSP mode changes.

Event Format: BRSP,<Conn_Handle>,<BRSP_Status>,<TX_Mode>,<RX_Mode>

Event Values:

- **Conn_Handle:** Connection handle

- **BRSP_Status:**
  1 = Data Mode
  2 = Remote Command Mode Client
  3 = Remote Command Mode Server
  5 = BRSP Already Initiated by Remote Device
  6 = Requested Mode Not Supported
  7 = Unauthenticated Pairing Required
  8 = Authenticated Pairing Required
  9 = BRSP Service Not Found

- **TX_Mode:**
  0 = Serial Port Mode – All data received on the UART will be sent over BRSP.
  1 = Command Mode – All data received on the UART will continue to be parsed locally as commands. Data can be sent over BRSP using the ATBRSPW command.

- **RX_Mode:**
  0 = Serial Port Mode – All data received over the air through BRSP will be sent directly through the UART.
  1 = Event Mode – All data received over the air through BRSP will be sent in BRSPD events.

Example(s):

1. A connection is made to ECFE7E000001 with BRSP_Mode set to Data Mode, once BRSP has been initialized, the BRSP event is sent signaling that Data Mode is ready:

```
COMMAND:  ATDMLE,ECFE7E000001:0,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
EVENT:    <cr_lf>
          BRSP,0,1,0,0<cr_lf>
```

Note(s):

- *Using the Escape to Command Mode command (+++) to switch to command mode does not change the BRSP mode. So, when switching back and forth between command mode and data mode, or between command mode and remote command mode, no BRSP events will be sent.*

7.6.3 BRSP Data (BRSPD)

BRSP DATA EVENT

Function: The BRSP data event will be sent when BRSP data is received an BRSP RX_Mode is set to Event Mode in ATSBRSP.

Event Format: BRSPD,<Conn_Handle>,<Data_Length>,<Data>

Event Values:

- **Conn_Handle:** Connection handle
- **Data_Length:** Length of the Data field in bytes.
- **Data:** ASCII BRSP data.

Example(s):

1. A connection is made to ECFE7E000001 with BRSP_Mode set to Data Mode, once BRSP has been initialized, the BRSP event is sent signaling that Data Mode is ready. Then a BRSPD event is received with the string “hello”.

```
COMMAND:  ATDMLE,ECFE7E000001:0,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          BRSP,0,1,1,1<cr_lf>
EVENT:    <cr_lf>
          BRSPD,0,5,hello<cr_lf>
```

7.6.4 Custom Profile Read (CP_READ)

CUSTOM PROFILE READ EVENT

Function: The custom profile read event will be sent when a GATT client device initiates a read on any of the characteristics or descriptors configured in the custom profile. Use the ATCPRR command to respond to this event.

Event Format: CP_READ,<Conn_Handle><Svc_Index>,<Char_Index>,<Desc_Index>

Event Values:

- **Conn_Handle:** Connection handle
- **Svc_Index:** The service index as an Integer value from 0-14.
- **Char_Index:** The characteristic index as an Integer value from 0-62.
- **Desc_Index:** The descriptor index as an Integer value from 0-62, or 63. When set to a value of 63, this index will be ignored, thus setting the specified characteristic index value.

Example(s):

1. A client on connection handle 4 initiates a GATT read on the custom service at index 0, characteristic at index 0, and since the descriptor index is 63, the client is trying to read the characteristic value.

```
EVENT:     <cr_lf>
           CP_READ,4,0,0,63<cr_lf>
COMMAND:   ATCPUV,0,0,63,0,E2FA8CD00977<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
COMMAND:   ATCPRR,4,0,0,63,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

2. A client on connection handle 0 initiates a GATT read on the custom service at index 4, characteristic at index 4, and since the descriptor index is 2 the client is trying to read the descriptor value.

```
EVENT:     <cr_lf>
           CP_READ,0,4,4,2<cr_lf>
COMMAND:   ATCPUV,4,4,2,0,0010<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
COMMAND:   ATCPRR,0,4,4,2,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

7.6.5 Custom Profile Write (CP_WRITE)

CUSTOM PROFILE WRITE EVENT

Function: The custom profile write event will be sent when a GATT client device initiates a write on any of the characteristics or descriptors configured in the custom profile. Use the ATCPWR command to respond to this event.

Event Format: CP_WRITE,<Conn_Handle><Svc_Index>,<Char_Index>,<Desc_Index>,
<Value_Length>,<Value>

Event Values:

- **Conn_Handle:** Connection handle
- **Svc_Index:** The service index as an Integer value from 0-14.
- **Char_Index:** The characteristic index as an Integer value from 0-62.
- **Desc_Index:** The descriptor index as an Integer value from 0-62, or 63. When set to a value of 63, this index will be ignored, thus setting the specified characteristic index value.
- **Value_Length:** Length of characteristic value in bytes.
- **Value:** Value formatted in hex.

Example(s):

1. A client on connection handle 2 initiates a GATT write on the custom service at index 2, characteristic at index 3, and since the descriptor index is 63, the client is trying to write to the characteristic value.

```
EVENT:    <cr_lf>
          CP_WRITE,2,2,3,63,5,48454C4C4F<cr_lf>
COMMAND:  ATCPWR,2,2,3,63,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

8 Commands

8.1 General

8.1.1 AT

AT

Function: The AT prefix by itself can be used to check communication with the module. It will always respond with an OK.

Example(s):

```
COMMAND:  AT<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

AT?

Function: AT? will respond with a list of all available commands.

Example(s):

```
COMMAND:  AT?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          AT<cr_lf>
          AT?<cr_lf>
          ATA?<cr_lf>
          ATADC?<cr_lf>
          …
          ATV?<cr_lf>
          ATZ<cr_lf>
          <cr_lf>
          DONE,2,0<cr_lf>
```

8.1.2 Help (ATHELP)

HELP

Function: Provides the address of the BlueRadios forum.

Example(s):

```
COMMAND:  ATHELP<cr>
RESPONSE: <cr_lf>
          www.blueradios.com/forum<cr_lf>
```

8.1.3 Response Mode (ATSRM)

SET RESPONSE MODE

Function: Sets the module's response mode, which configures how verbose the module will be.

Command Format: ATSRM,<Response_Mode>,<Disconnected_Mode>,<Response_Event_Flags>

Command Parameter(s):

- **Response_Mode:** The following events cannot be disabled: DISCOVERY, RSSI, GATT_DPS, GATT_DC, GATT_DCD, GATT_VAL, GATT_LVAL, BRSPD.

  0 = Full – All responses and events will be sent.
  1 = Limited – Events and command responses will be sent. Command status responses (OK, ERROR) will not be sent.
  2 = Minimal – Only events with requested data and command responses will be sent. Command status responses and events that do not contain requested data will not be sent.
  3 = Custom – Individual responses and events can be enabled or disabled.

- **Disconnected_Mode:**
  0 = Command Mode
  1 = No AT commands are accepted and UART data is flushed.

- **Response_Event_Flags:** If Response_Mode = 3, then individual responses and events can be enabled or disabled using the Response_Event_Flags. If Response_Mode != 3, this parameter is not used.

Decimal Value	Hex Value	Response / Event
1	0x00000001	OK
2	0x00000002	ERROR
4	0x00000004	RESET
8	0x00000008	DONE,1,0 (ATDSLE)
16	0x00000010	DONE,1,1 (ATDILE)
32	0x00000020	DONE,1,2 (ATDMLE)
64	0x00000040	DONE,2,0 (General)
128	0x00000080	CONNECT
256	0x00000100	DISCONNECT
512	0x00000200	CPU_STATUS
1024	0x00000400	CPU
2048	0x00000800	MTU
4096	0x00001000	PHYU
8192	0x00002000	PAIR_REQ
16384	0x00004000	PAIRED
32768	0x00008000	PAIR_FAIL
65536	0x00010000	RESOLVED
131072	0x00020000	GATT_DONE
262144	0x00040000	BATT
524288	0x00080000	BRSP
4294967295	0xFFFFFFFF	All Responses / Events Enabled

Example(s):

```
COMMAND:  ATSRM,0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *When switching to limited or minimal response mode, the OK response will not come back after the ATSRM command.*

GET RESPONSE MODE

Function: Sets the modules response mode.

Command Format: ATSRM?

Response Format: <Response_Mode>,<Disconnected_Mode>,<Response_Event_Flags>

Example(s):

```
COMMAND:  ATSRM?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,0,4294967295<cr_lf>
```

8.2 Reset

8.2.1 Reset (ATRST)

RESET

Function: Resets the module.

Command Format: ATRST,<Delay>

- **Delay:**
  Integer value from 0 to 65535 [ms] to delay reset.
  Default: 0 (Reset immediately)

Example(s):

1. An ATRST is sent and once the module has reset, the RESET event is triggered.

```
COMMAND:  ATRST<cr>
RESPONSE: <cr_lf>
          RESET,0,0,BR-LE5.0-S1<cr_lf>
```

Note(s):

- *This command does not return an OK.*
- *During a reset, the TX line of the UART will pulse low, which may be read as a NULL character showing up before the <cr_lf> on some receivers.*
- *If using USB during a reset, the USB port will close and need to be re-opened.*

8.2.2 Factory Reset (ATFRST)

FACTORY RESET

Function: Resets the module back to its factory defaults.

Command Format: ATFRST,<Clear_Paired>

- **Clear_Paired:**
  0 = Clear paired list
  1 = Keep paired list

Example(s):

1. An ATFRST is sent and once the module has reset, the RESET event is triggered.

```
COMMAND:  ATFRST<cr>
RESPONSE: <cr_lf>
          RESET,1,0,BR-LE5.0-S1<cr_lf>
```

Note(s):

- *This command does not return an OK.*
- *During a reset, the TX line of the UART will momentarily pulse low, which may be read as a NULL character showing up before the <cr_lf> on some receivers.*
- *Setting PIO_4 high on the BR-LE5.0-S1 or low for the PAN1780-AT during reset can also be used to factory reset a module.*

8.3 Configuration Control

By default, the module configuration will be automatically stored to flash any time an ATS prefixed command is sent. This behavior can be changed using the ATSFC command, allowing automatic flash storage to be disabled. When disabled the ATFC command can be used to manually store the module configuration in flash – this will store all changes to the configuration. To store individual command settings without saving other modifications the '#' character can be added to the end of the command string, for example ATSN#,NewModuleName<cr>.

8.3.1 Configuration Lock (ATSCL)

SET CONFIGURATION LOCK

Warning: Be careful when locking the configuration. The password cannot be obtained after it has been changed and a hardware factory reset using PIO_4 is the only way to reset it.

Command Format: ATSCL,<Lock>,<Password>

Command Parameter(s):

- **Lock:**
  0 = Unlock
  1 = Lock

- **Password:** 1-20 alphanumeric characters (Case sensitive, includes spaces). The password is stored when locking the configuration. To unlock the configuration the same password that was used to lock the configuration must be entered.

Example(s):

1. The configuration is locked with the Password "good_password". It is then attempted to be unlocked using the Password "bad_password", which fails and causes an ERROR,04. The configuration is then successfully unlocked using the correct Password:

```
COMMAND:  ATSCL,1,good_password<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
COMMAND:  ATSCL,0,bad_password<cr>
RESPONSE: <cr_lf>
          ERROR,4<cr_lf>
COMMAND:  ATSCL,0,good_password<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *All configuration commands will reply ERROR,6 after the configuration has been locked.*

- *ATSCL will still work after locking the user settings, allowing them to be unlocked.*

- *ATSCL will always store in flash regardless of the ATSFC setting.*

- *ATFRST will be locked, but a hardware factory reset using PIO_4 can still be used.*

GET CONFIGURATION LOCK

Function: Gets the configuration lock setting.

Command Format: ATSCL?

Response Format: <Lock>

Example(s):

```
COMMAND:  ATSCL?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0<cr_lf>
```

8.3.2 Configure Flash Storage Configuration (ATSFC)

SET FLASH STORAGE CONFIGURATION

Function: Sets the flash storage configuration settings.

Command Format: ATSFC,<Store_Config>,<Store_Last_Connected_Address>

Command Parameter(s):

- **Store_Config:**
  0 = Disabled – Any configuration changes must be manually stored using ATFC.
  1 = Enabled – All configuration changes will be automatically stored in flash.

- **Store_Last_Connect_Address:**
  0 = Disabled – Last connected address will not be stored in flash.
  1 = Enabled – Last connected address used by ATDMLLE and ATLCALE? will be stored in flash. The address from the last CONNECT event will be used and will not be stored in flash until all active connections have been disconnected.

Example(s):

```
COMMAND:  ATSFC,1,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET FLASH STORAGE CONFIGURATION

Function: Gets the flash storage configuration settings.

Command Format: ATSFC?

Response Format: <Store_Config>,<Store_Last_Connected_Address>

Example(s):

```
COMMAND:  ATSFC?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,1<cr_lf>
```

8.3.3 Flash Configuration (ATFC)

FLASH CONFIG

Function: Stores the current configuration in flash. This command is intended to be used after all settings have been configured. If Store_Config is enabled in ATSFC, then each time a command is used to change configuration the configuration will be stored to flash, which is inefficient if multiple changes are being made. So, if multiple changes need to be made, the most efficient way to do is to disable Auto_Flash, make all the changes, then use ATFC to store them.

Command Format: ATFC

Example(s):

```
COMMAND:  ATFC<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *ATFC can still be used once the configuration has been locked.*
- *To store individual command settings without saving other modifications the '#' character can be added to the end of the command string, for example ATSN#,NewModuleName<cr>.*

8.3.4 Set Configuration (ATSCFG)

SET CONFIGURATION

Function: Can be used to set the entire module configuration with just a few commands. If the OK response is received the configuration will be stored in flash.

Command Format: ATSCFG,<Config_Index>,<Hex_Config>

- **Config_Index:**
  0 = Main Config
  1 = Extended Ad Config
  2 = DIS Config
- **Hex_Config:** The config string read using the ATSCFG? command.

Example(s):

```
COMMAND:
          ATSCFG,0,C6E600352E302E302E31322D533100010700000000000000000000000000
          000000000000000000000030808000000C000020201000001010202020202020200
          000050000020202020000020202020202020202020202020202020202020000000
          0064FFFF000AD0070064FFFF00641400000201000000000FFFFFFFF19010000000
          000000000000002B00000001008403426C7565526164696F733031303337380000
          000000100000000007000101000000008A00001070A000A00100010000700FFFF0
          107000010001000000008001000000090018902000000000000000201FF020AFF05
          12FFFFFFFF1107FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF001E05FFFFFFFFFF1509F
          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000001C0707F701000900FFFFFF
          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE8AB0020E8AC0020<cr>
RESPONSE:<cr_lf>
          OK<cr_lf>
```

Note(s):

- *After configuring the module using ATSCFG the module needs to be reset for the new configuration to take effect.*

GET CONFIGURATION

Function: Reads the current configuration in hex format.

Command Format: ATSCFG?,<Config_Index>

Command Parameter(s):

- **Config_Index:**
  0 = Main Config
  1 = Extended Ad Config
  2 = DIS Config

Example(s):

```
COMMAND:  ATSCFG?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          C6E600352E302E302E31322D533100010700000000000000000000000000000000
          0000000000000030808000000C000020201000001010202020202020200000500000
          0202020200000202020202020202020202020202020202020200000000064FFFF0
          00AD0070064FFFF006414000002010000000000FFFFFFFF19010000000000000000
          0000002B00000001008403426C7565526164696F7330313033373800000000001000
          0000000700010100000000008A00001070A000A00100010000700FFFF0107000010
          0010000000080010000000 9001B902000000000000000201FF020AFF0512FFFFFFF
          F1107FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF001E05FFFFFFFFFF1509FFFFFFFFFF
          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000001C0707F701000900FFFFFFFFFFFFFFFF
          FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
          FFFFFFFFFFFFFFFFFFFFFFFFFFFFE8AB0020E8AC0020<cr_lf>
          <cr_lf>
          DONE,2,0<cr_lf>
```

8.3.5 Configuration Dump (ATCFG?)

GET CONFIGURATION

Function: Dumps the current configuration. The configuration can be read in a readable format or a hex format that can be applied using the ATSCFG command.

Command Format: ATCFG?

Example(s):

```
COMMAND:  ATCFG?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          ATMT?        = BR-LE5.0-S1<cr_lf>
          ATV?         = 5.0.1.0-S1 (DBG)<cr_lf>
          ATA?         = C9B52CB7CF09:1<cr_lf>
          ATSRM?       = 0,0<cr_lf>
          ATSCL?       = 0<cr_lf>
          ATSFC?       = 1,1<cr_lf>
          ATSZ?        = 0,0,0<cr_lf>
          ATSN?        = BlueRadiosB7CF09,0<cr_lf>
          ATSAPP?      = 0<cr_lf>
          ATSAT?       = 1,0,900<cr_lf>
          ATSDBLE?     = 1,0<cr_lf>
          ATSDPL?      = 8,8<cr_lf>
          ATSUART?     = 7,0,0,1<cr_lf>
          ATSUSB?      = 1,0<cr_lf>
          ATSNFC?      = 0<cr_lf>
          ATSPIO?,0    = 2,0,0<cr_lf>
          ATSPIO?,1    = 2,0,0<cr_lf>
          ATSPIO?,2    = 1,0,0<cr_lf>
          ...
          ATSDMTU?     = 247<cr_lf>
          ATSSPHY?     = 7,7<cr_lf>
          ATSPLE?      = 2,0,3,1,0<cr_lf>
          ATSPK?       = 0<cr_lf>
          ATSDIS?,0    = 1,BlueRadios,Inc.<cr_lf>
          ATSDIS?,1    = 1,BR-LE5.0-S1<cr_lf>
          ATSDIS?,2    = 1,B7CF09<cr_lf>
          ATSDIS?,3    = 0,Hardware Revision<cr_lf>
          ATSDIS?,4    = 1,5.0.1.0-S1 (DBG)<cr_lf>
          ATSDIS?,5    = 0,Software Revision<cr_lf>
          ATSDIS?,6    = 0,00000000000000000000000000<cr_lf>
          ATSDIS?,7    = 0,FE006578706572696D656E74616C<cr_lf>
          ATSDIS?,8    = 1,01850000050001<cr_lf>
          ATSBAS?      = 1,0,0,0<cr_lf>
          ATSDFU?      = 1,1<cr_lf>
          ATSBRSP?     = 1,3,0,0,0<cr_lf>
          ATSSP?       = 43,0<cr_lf>
          ATSBOOT?     = 1,7,1<cr_lf>
          <cr_lf>
          DONE,2,0<cr_lf>
```

8.4 Sleep

8.4.1 Sleep (ATZ)

SLEEP

Function: Enables sleep mode. When sleep mode is enabled the module will enter a low power state when its MCU is idle. By default, the module will not be in sleep mode upon power up, but for maximum power savings it should be put into sleep mode as soon as possible after all configuration is complete.

While in sleep mode the module will not be able to receive data on the UART, but the module will still be able to handle RF tasks, so any active connection requests, discovery requests, advertising requests, or connections will be maintained in the background, with the module sleeping in between task events. The module can still receive incoming data over the air and output it on the UART while it is in sleep mode.

If shutdown is enabled the module will enter its lowest power state where it can only be woken back up by a reset or by pulsing PIO_3 (which will trigger a reset). In this state all peripherals including the UART, USB and all BT5.0 functionality will be disabled. By default, all PIOs (except PIO_3) will be set to the disconnected state during shutdown, but they can also be configured to maintain their current state using the ATSZ command.

Command Format: ATZ,<Shutdown>

Command Parameter(s):

- **Shutdown:**
  0 = Sleep
  1 = Shutdown

Example(s):

```
COMMAND:  ATZ<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- ***When sleep mode / shutdown is enabled the module will not be able to receive data on the UART, so it will not be able to accept any AT commands.***
- ***When shutdown is enabled all BT5.0 functionality will be disabled.***
- *Since the UART cannot receive data when the device enters sleep mode, RTS will be set high when sleep mode is enabled, and flow control is enabled.*
- *See section PIO_3 – Sleep Mode Toggle Input for more information on toggling the device in and out of sleep mode.*

8.4.2 Sleep Configuration (ATSZ)

SET SLEEP

Function: Configures sleep mode.

Command Format: ATSZ,<Sleep_On_Reset>,<Wake_On_Rx>,<PIO_7_Sleep_Status>,
<Shutdown_PIO_State>

Command Parameter(s):

- **Sleep_On_Reset:**
  0 = Disabled.
  1 = Sleep mode will automatically be enabled on power up or after a reset.

- **Wake_On_Uart_Rx:**
  0 = Disabled.
  1 = Sleep mode will be disabled after a high to low transition on the UART_RX line. This allows the module to be woken up by sending a character over the UART, but this character will not be received by the module's UART, as it cannot wake up fast enough to receive this character.

- **PIO_7_Sleep_Status:**
  0 = Disabled.
  1 = PIO_7 default will show sleep mode status. PIO_7 will be low on the BR-LE5.0-S1 or high on the PAN1780-AT when sleep mode is enabled, and high on the BR-LE5.0-S1 or low for the PAN1780-AT when sleep mode is disabled, and the module is awake and able to receive commands.
  2 = PIO_7 will show when the module is in its low power state. PIO_7 will be high on the BR-LE5.0-S1 or low for the PAN1780-AT when the MCU is idle and in the low power state, and low on the BR-LE5.0-S1 or high on the PAN1780-AT when it is active. This status is intended for debugging/observation only.

- **Shutdown_PIO_State:**
  0 = All PIOs (except PIO_3) will be put in the disconnected state during shutdown.
  1 = PIOs will maintain their current state at the time of shutdown during shutdown.

Example(s):

```
COMMAND:  ATSZ,1,0,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *When waking up from sleep mode, the time between a PIO_3 pulse or an RX transition and the module being ready to receive data on the UART can vary based on its current state. Monitor RTS or PIO_7 sleep status to know when the module is awake. If neither of these lines can be monitored, wait at least 2ms before sending data.*

GET SLEEP

Function: Gets the sleep configuration.

Command Format: ATSZ?

Response Format: <Sleep_On_Reset>,<Wake_On_Rx>,<PIO_7_Sleep_Status>,
<Shutdown_PIO_State>

Example(s):

```
COMMAND:  ATSZ?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,0,1,0<cr_lf>
```

8.5 Module Information

8.5.1 Module Type (ATMT?)

GET MODULE TYPE

Function: Gets the module type.

Command Format: ATMT?

Response Format: <Module_Type>

Response Values:

- **Module_Type:**

BR-LE5.0-S1	S1 Single Mode LE 5.0 Module
PAN1780	Panasonic Bluetooth® Low Energy RF Module

Example(s):

```
COMMAND:  ATMT?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          BR-LE5.0-S1<cr_lf>
```

8.5.2 Stack Type (ATST?)

GET STACK TYPE

Function: Gets the stack type.

Command Format: ATST?

Response Format: BlueRadios nBlue

Example(s):

```
COMMAND:  ATST?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          BlueRadios nBlue<cr_lf>
```

8.5.3 Firmware Version (ATV?)

GET FIRMWARE VERSION

Function: Gets the module's firmware version.

Command Format: ATV?

Response Format: <Firmware_Version>

Response Values:

- **Firmware_Version:** Module Firmware Version.

  <Major AT Version>.<Minor AT Version>.<Bug Fix Version>.0-<Module_Type_Abbr>

  The first two digits, <Major AT Version>.<Minor AT Version>, should match the command set document version and will only be incremented if changes are made to the command set. The <Bug Fix Version> is incremented if a release is issued with bug fixes only and no changes to the command set. The fourth digit is used internally and will always be a 0 for release versions. The two characters after the hyphen are an abbreviation of the module type.

Example(s):

```
COMMAND:  ATV?<cr>
RESPONSE: <cr_lf>
          OK <cr_lf>
          <cr_lf>
          5.1.0.0-S1<cr_lf>
```

8.5.4 BT5.0 Device Address (ATA?)

GET ADDRESS

Function: Gets the module's BT5.0 Device Address.

Command Format: ATA?

Response Format: <Addr>

Response Values:

- **Addr:** Local BT5.0 device address and address type.

Example(s):

```
COMMAND:  ATA?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          ECFE7E000001:0<cr_lf>
```

8.5.5 BT5.0 Device Name (ATSN)

SET NAME

Function: Sets the module's BT5.0 Device Name. This will be the GAP Device Name in the GAP Service Profile which can be read through the GATT layer. The name will also be placed in the scan response data, so when another device performs a discovery, the name will be returned in the scan response.

Command Format: ATSN,<Name>,<GATT_Write>

Command Parameter(s):

- **Name:** BT5.0 Device Name – 1-20 characters
  Default: BlueRadios######

  Part or all the BT5.0 Device Address can be automatically added into the name by including 2 or more # characters in the name. If only a portion of the address is included the least significant part of the address will be used.

- **GATT_Write:**
  0 = Disable - The name cannot be changed by a remote device.
  1 = Writeable - The name can be changed by any remote device.
  2 = Authenticated Paired Writeable - Name can only be changed by authenticated paired devices.
  3 = Paired Writeable – Name can only be changed by paired devices.

Example(s):

(BT5.0 Device Address = ECFE7E123456)

```
COMMAND:  ATSN,BlueRadios######<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET NAME

Function: Gets the module's BT5.0 Device Name.

Command Format: ATSN?

Response Format: <Name>,<GATT_Write>

Example(s):

```
COMMAND:  ATSN?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          BlueRadios123456,0<cr_lf>
```

8.5.6 Appearance (ATSAPP)

SET APPEARANCE

Function: Sets the value of the Appearance Characteristic, which represents the external appearance of the module.

Command Format: ATSAPP,<Appearance>

Command Parameters:

- **Appearance:** 16-bit integer value. Valid values can be found at: https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.gap.appearance.xml
  Default: 0

Example:

```
COMMAND:  ATSAPP,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET APPEARANCE

Function: Gets the Appearance.

Command Format: ATSAPP?

Response Format: <Appearance>

Example:

```
COMMAND:  ATSAPP?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0<cr_lf>
```

8.5.7 Address Type (ATSAT)

SET ADDRESS TYPE

Function: Sets the module's BT5.0 identity address, which identifies the module to other devices. Also allows LE Privacy to be enabled or disabled.

This address will be exchanged with a peer device during bonding.

Supported Identity Address Types:

- **Public Device Address:** Unique IEEE BT5.0 Device Address.
- **Static Random Address:** A randomly generated address that does not change until the module is reset or power cycled.

Command Format: ATSAT,<Addr_Type>,<Privacy_Type>,<Private_Addr_Cycle_Time>

Command Parameters:

- **Addr_Type:**
  0 = Public Address – The unique BT5.0 Device Address with BlueRadios OUI: ECFE7E will be used.
  1 = Static Random Address - A random address that was generated during production will be used.
- **Privacy_Type:**
  0 = No privacy.
  1 = Resolvable Private Random Address – A new resolvable random address will be automatically generated every Private_Addr_Cycle_Time seconds.
  2 = Non-Resolvable Private Random Address – A new non-resolvable random address will be automatically generated every Private_Addr_Cycle_Time seconds.
- **Private_Addr_Cycle_Time:**
  The private address cycle interval in seconds [1-65535]. The private address will also change after disconnecting.
  Default: 900 (15 minutes)

Example:

```
COMMAND:  ATSAT,0,1,900<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used if in the Discovering, Connecting or Connected states.*

GET ADDRESS TYPE

Function: Gets the current address type and address being used by the module.

Command Format: ATSAT?

Response Format: <Addr_Type>,<Privacy_Type>,<Private_Addr_Cycle_Time>

Example:

```
COMMAND:  ATSAT?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,1,900 <cr_lf>
```

8.6 Module State

8.6.1 Get State (ATSLE?)

GET STATE

Function: Gets the current LE state of the module.

Command Format: ATSLE?

Response Format: <Idle_Testing>,<Advertising>,<Discovering>,<Connecting>,<Connected>

Response Values:

- **Idle_Testing:**
  0 = Not Idle
  1 = Idle
  2 = Testing (ATTXT/ATRXT)

- **Advertising:**
  0 = Not Advertising
  1 = Advertising (ATDSLE)

- **Discovering:**
  0 = Not Discovering
  1 = Discovering (ATDILE)

- **Connecting:**
  0 = Not Connecting
  1 = Connecting (ATDMLE)

- **Connected:**
  0 = Not Connected
  >0 = Connected, number is equal to the number of active connections

Example(s):

```
COMMAND:  ATSLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,0,0,0,0<cr_lf>
```

8.6.2 Default Behavior (ATSDBLE)

SET DEFAULT BEHAVIOR

Function: Sets the module's default behavior. Allows the user to select a command to automatically execute based on a trigger.

Command Format: ATSDBLE,<Command>,<Trigger>

Command Parameter(s):

- **Command:**
  0 = No Command (Idle)
  1 = ATDSLE (Advertising)
  2 = ATDILE (Discovering) – Cannot be used with Trigger = 1
  3 = ATDMLE,WL,0 (Connecting to White List)
  4 = ATDMLE,WL,1 (Connecting to White List, BRSP Data Mode)
  5 = ATDMLE,WL,2 (Connecting to White List, BRSP Remote Command Mode)

- **Trigger:**
  0 = Idle
  1 = Idle UART Data **–** Cannot be used if ATSDB Trigger = 1

Example(s):

1. The module's default behavior is set to advertise on idle. ATSLE is used to confirm that the module is not advertising. The default behavior is then set to no command on idle, so the ATDSLE is cancelled, triggering a DONE event.

```
COMMAND:  ATSDBLE,1,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
COMMAND:  ATSLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,1,0,0,0<cr_lf>
COMMAND:  ATSDBLE,0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DONE,1,0<cr_lf>
```

Note(s):

- *At least one device must be in the whitelist (see ATSWL) to use the ATDMLE actions.*

- *When switching between different default behavior settings, the command from the previous default behavior will be cancelled.*

- *A default behavior can be temporarily disabled by executing an ATDC command or by toggling PIO_4.*

GET DEFAULT BEHAVIOR LE

Function: Gets the module's default behavior.

Command Format: ATSDBLE?

Response Format: <Command>,<Trigger>,<Bridge>

Example(s):

```
COMMAND:  ATSDBLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,0<cr_lf>
```

8.6.3 Cancel/Idle Command (ATDC)

CANCEL/IDLE

Function: This command tells the radio to cancel commands that run in the background such as ATDM, ATDS, or ATDI. This command can come in handy for a quick exit from commands like ATDI if there are no devices in the area and you do not want to wait for a timeout. Not passing any parameters will cancel all active commands, causing the module to go into the idle state.

Command Format: ATDC,<Command_Type>,<Command>

Command Parameter(s):

- **Command_Type:**
  1 = Low Energy (LE)
  2 = General
  3 = Continuous RSSI

- **Command:**
  Command_Type = 1 (LE):
  0 = ATDSLE
  1 = ATDILE
  2 = ATDMLE

  Command_Type = 3 (Continuous RSSI):
  Command = Conn_Handle of connection to cancel continuous RSSI on. Not passing this parameter will cancel continuous RSSI on all connections.

Example(s):

```
COMMAND:  ATDC<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

8.7 Hardware Configuration / Control

8.7.1 Set Default Power Levels (ATSDPL)

SET DEFAULT POWER LEVELS

Function: Sets the default transmit power levels.

Command Format: ATSDPL,<Ad_TX_Power>,<Scan_TX_Power>

Command Parameter(s):

- **Ad_TX_Power:** Transmit power level for advertising and peripheral (incoming) connections.
- **Scan_TX_Power:** Transmit power level for discovery and central (outgoing) connections. If not specified, will be set to the same value as Ad_TX_Power.

Supported Values (dBm): -40,-20,-16,-12,-8,-4,0,2,3,4,5,6,7,8

Example(s):

```
COMMAND:  ATSDPL,8,8<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *See the Module User's Guide or Module Datasheet for current consumption numbers at different transmit power level settings.*

GET DEFAULT POWER LEVELS

Function: Gets the default transmit power levels.

Command Format: ATSDPL?

Response Format: <Ad_TX_Power>,<Scan_TX_Power>

Example(s):

```
COMMAND:  ATSDPL?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          8,8<cr_lf>
```

8.7.2 Set Current Power Level (ATSCPL)

SET CURRENT POWER LEVEL

Function: Sets the current power level for an active connection.

Command Format: ATSCPL,<Conn_Handle>,<TX_Power>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to update.
- **TX_Power:** -40,-20,-16,-12,-8,-4,0,2,3,4,5,6,7,8 dBm

Example(s):

```
COMMAND:  ATSCPL,0,8<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *See the Module User's Guide or Module Datasheet for current consumption numbers at different transmit power level settings.*

GET CURRENT POWER LEVEL

Function: Gets the current transmit power level for an active connection.

Command Format: ATSCPL?,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to update.

Response Format: <TX_Power>

Example(s):

```
COMMAND:  ATSCPL?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          8<cr_lf>
```

8.7.3 UART Configuration (ATSUART)

SET UART

Function: Configures the module's UART. The new settings will take effect after the OK response has been sent.

Command Format: ATSUART,<Baud_Rate>,<Parity>,<Stop_Bits>,<Flow_Control>

Command Parameter(s):

- **Baud_Rate:** 3-10

Baud rate (bps)	Value
9600	3
19200	4
38400	5
57600	6
115200	7
230400	8
460800	9
921600	10

- **Parity:**
  0 = None
  2 = Even

- **Stop_Bits:**
  0 = One

- **Flow_Control:**
  0 = Flow Control Off
  1 = Flow Control On

Example(s):

```
COMMAND:  ATSUART,7,0,0,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *The new settings will take effect after the OK response has been sent.*
- *The RTS line of the radio will be low when the radio is ready to receive data and high when its buffer is full. When RTS goes high wait until it returns to low before sending more data to avoid losing information.*
- *The module can be factory reset by setting PIO_4 to VDD on the BR-LE5.0-S1 or to GND on the PAN1780-AT during reset.*

GET UART

Function: Gets the UART configuration.

Command Format: ATSUART?

Response Format: <Baud_Rate>,<Parity>,<Stop_Bits>,<Flow_Control>

Example(s):

```
COMMAND:  ATSUART?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          7,0,0,1<cr_lf>
```

8.7.4 USB Configuration (ATSUSB)

SET USB

Function: Configures the module's USB port.

Command Format: ATSUSB,<Enable_USB >,<Enable_Event_Port>

Command Parameter(s):

- **Enable_USB:** If enabled, the module's USB port will be enabled and a USB CDC ACM virtual serial port will be available. The module can be controlled through this port through AT commands and events in the same way it can over the UART.
  0 = Disable
  1 = Enable

- **Enable_USB_Event_Port:** If enabled and USB is enabled, a 2nd USB CDC ACM serial port will be made available that will only transmit events. When this port is enabled events will no longer be sent from the 1st USB CDC ACM port
  0 = Disable
  1 = Enable

Example(s):

```
COMMAND:  ATSUSB,1,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *Changes to these settings will require a reset to take effect.*

GET USB

Function: Gets the USB configuration.

Command Format: ATSUSB?

Response Format: <Enable_USB >,<Enable_Event_Port>

Example(s):

```
COMMAND:  ATSUSB?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,0<cr_lf>
```

8.7.5 NFC Configuration (ATSNFC)

SET NFC

Function: Enables/disables the module's NFC (Near Filed Communication) subsystem. When enabled the module can pair with another device using NFC.

Command Format: ATSNFC,<Enable>

Command Parameter(s):

- **Enable**
  0 = Disabled
  1 = Enabled

Example(s):

```
COMMAND:  ATSNFC,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *Using NFC requires an external NFC antenna to be connected to the module's NFC1/NFC2 pins.*

GET NFC

Function: Gets the NFC configuration.

Command Format: ATSNFC?

Response Format: <Enable>

Example(s):

```
COMMAND:  ATSNFC?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0<cr_lf>
```

8.7.6 PIO Configuration (ATSPIO)

SET PIO CONFIG

Warning: Applying an external voltage to a PIO assigned as an output may permanently damage the module. The maximum voltage level on any pin should not exceed 3.6V. The I/O is NOT 5V tolerant.

Function: Sets PIO configuration.

Command Format: ATSPIO,<PIO_Num>,<Direction>,<Default_Level>,<Pull_Drive_Strength>

Command Parameter(s):

- **PIO_Num:** 0-44 (Excluding 3,4,6,14,17,18,23,24)

- **Direction:**
  0 = Input
  1 = Output
  2 = Pin Disconnected

- **Default_Level:**
  Direction = 0/2: No Effect

  Direction = 1:
  0 = Low
  1 = High

- **Pull_Drive_Strength:**
  Direction = 0, Pull Direction:
  0 = Pull Down
  1 = Pull Up
  2 = No Pull

  Direction = 1, Drive Strength: (Standard = 2mA, High = 10mA)
  0 = S0S1 (Standard 0, Standard 1)
  1 = H0S1 (High 0, Standard 1)
  2 = S0H1 (Standard 0, High 1)
  3 = H0H1 (High 0, High 1)
  4 = D0S1 (Disconnect 0, Standard 1)
  5 = D0H1 (Disconnect 0, High 1)
  6 = S0D1 (Standard 0, Disconnect 1)
  7 = H0D1 (High 0, Disconnect 1)

Example(s):

```
COMMAND:  ATSPIO,7,1,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *The value of PIOs 2,5,7,8 can only be modified if Duty_Cycle is set to 0 in ATSLED.*

GET PIO CONFIG

Function: Reads PIO settings.

Command Format: ATSPIO?,<PIO_Num>

Command Parameter(s):

- **PIO Number:** 0-46 (Excluding 3,4,6,17,18,23,24)

Response Format: <Direction>,<Value>,<Pull_Drive_Strength>

Example(s):

```
COMMAND:  ATSPIO?,7<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,1,0<cr_lf>
```

8.7.7 PIO Level (ATPIOL)

SET PIO LEVEL

Warning: Applying an external voltage to a PIO assigned as an output may permanently damage the module. The maximum voltage level on any pin should not exceed 3.6V. The I/O is NOT 5V tolerant.

Function: Sets the output level of PIO's configured as outputs.

Command Format: ATPIOL,<PIO_Num>,<Level>

Command Parameter(s):

- **PIO_Num:** 0-44 (Excluding 3,4,6,14,17,18,23,24)
- **Level:** 0 = Low, 1 = High

Example(s):

```
COMMAND:  ATPIOL,7,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *The value of PIO_2 and PIO_5 can only be modified if Duty_Cycle is set to 0 in ATSLED.*
- *An error will be returned if the PIO is not configured as an output.*

GET PIO LEVEL

Function: Reads the current level of PIO's. The level can be read for both inputs and outputs.

Command Format: ATPIOL?,<PIO_Num>

Command Parameter(s):

- **PIO Number:** 0-46

Response Format: <PIO_Num>

Example(s):

```
COMMAND:  ATPIOL?,7<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1<cr_lf>
```

Note(s):

- *An error will be returned if the PIO is not configured as an input or an output.*

8.7.8 LED Configuration (ATSLED)

SET LED

Function: Sets the status LED (PIO_2/PIO_5) parameters.

Command Format: ATSLED,<Led_Num>,<Duty_Cycle>,<Period>,<Override_Default_Behavior>

Command Parameter(s):

- **Led_Num:**
  0 = PIO_2 – Will pulse when the module is connected.
  1 = PIO_5 – Will pulse when the module in not idle (advertising, discovering, connecting), excluding the connected state as it is handled by PIO_2.
  2 = PIO_7 – Will pulse if PIO_7_Sleep_Status is enabled in ATSZ.
  3 = PIO_8 – Will pulse when the module receives a valid AT command.

- **Duty_Cycle:** Integer decimal value from 0 to 100. If the Duty_Cycle is set to 0, the status LED functionality will be disabled, and the PIO can be controlled manually using the ATSPIO command.
  PIO_2 Default: 100, PIO_5 Default: 10, PIO_7 Default: 100, PIO_8 Default: 100

- **Period:** Integer decimal value from 1ms to 65,535ms
  PIO_2 Default: 65535, PIO_5 Default: 2000, PIO_7 Default: 65535, PIO_8 Default: 20

- **Override_Default_Behavior:**
  0 = Disabled - The LEDs will only pulse when in the states described above.
  1-65535 = Enabled - The led state behavior will be overridden and the led will pulse immediately 1-65534 pulses, 65535 will pulse indefinitely. If a non-zero value is used a 1 will be stored so the default behavior will be overridden, but it will not cause the led to blink after a reset. For example, if ATSLED,1,10,2000,5 is sent the PIO_5 led will immediately blink 5 times, but after a reset the led will not start blinking.

Example(s):

```
COMMAND:  ATSLED,1,10,2000,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET LED

Function: Gets the LED (PIO_2/PIO_5) parameters.

Command Format: ATSLED?,<Led_Num>

Command Parameter(s):

- **Led_Num:**
  0 = PIO_2
  1 = PIO_5

Response Format: <Duty Cycle>,<Period>,<Override_Default_Behavior>

Example(s):

```
COMMAND:  ATSLED?,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          10,2000,0<cr_lf>
```

8.7.9 Get ADC (ATADC?)

GET ADC

Function: Takes a reading from one of the module's ADCs in mV using an internal .6V reference.

Command Format: ATADC?,<ADC_Num>,<Resolution>,<Gain>,<Acquisition_Time>,
<Oversampling_Mode>

Command Parameter(s):

- **ADC_Num:** 0-8

ADC_Num	PIO_Num	Pin_Name
0	0	ADC_0
1	1	ADC_1
2	10	SPI_MISO
3	13	SPI_MOSI
4	11	SPI_CSB
5	12	SPI_CLK
6	9	PIO_9
7	29	PIO_29
8	VDD	VDD

- **Resolution:**
  0 = 8-bit
  1 = 10-bit
  2 = 12-bit
  3 = 14-bit

- **Gain:**
  0 = 1/6
  1 = 1/5
  2 = 1/4
  3 = 1/3
  4 = 1/2
  5 = 1
  6 = 2
  7 = 4

- **Acquisition_Time:** 0-5

Value	Acq Time	Max Source Resistance
0	3 µS	10 kΩ
1	5 µS	40 kΩ
2	10 µS	100 kΩ
3	15 µS	200 kΩ
4	20 µS	400 kΩ
5	40 µS	800 kΩ

- **Oversampling_Mode:** 0-8
  0 = Disabled
  1 = 2x
  2 = 4x
  3 = 8x
  4 = 16x
  5 = 32x
  6 = 64x
  7 = 128x
  8 = 256x

Response Format: <ADC_Value>

Response Value(s):
- **ADC_Value:** Voltage in mV.

Example(s):

```
COMMAND:  ATADC?,0<cr>
RESPONSE: <cr_lf>
          OK
          <cr_lf>
          1250<cr_lf>
```

8.7.10 Get Battery Level (ATBL?)

GET BATTERY LEVEL

Function: Gets the battery level as a percentage from 0-100. The level is only valid if a 3.0V coin cell is connected directly to VDD.

Command Format: ATBL?

Response Format: <Batt_Level>

Response Value(s):

- **Batt_Level:** 0-100 [%]

Example(s):

```
COMMAND:  ATBL?<cr>
RESPONSE: <cr_lf>
          OK
          <cr_lf>
          100<cr_lf>
```

Note(s):

- *This does not read the battery level from the Battery Service but reads directly from the ADC. Use ATSBASL/ATSBASL? to read the battery level for the Battery Service.*

8.7.11 Get Temperature (ATT?)

GET TEMPERATURE

Function: Get the current temperature of the module's internal temperature sensor.

Command Format: ATT?

Response Format: <Temp_Celsius>,<Temp_Fahrenheit>

Response Value(s):

- **Temp_Celsius:** Temperature in Celsius.
- **Temp_Fahrenheit:** Temperature in Fahrenheit.

Example(s):

1. The module will return floating point values with two decimal places.

```
COMMAND:  ATT?<cr>
RESPONSE: <cr_lf>
          OK
          <cr_lf>
          26.25,79.25<cr_lf>
```

Note(s):

- *The accuracy of the internal temperature sensor is ±5°C.*

8.7.12 Calibrate Temperature Sensor (ATCT)

CALIBRATE TEMPERATURE

Function: Calibrates the module's internal temperature sensor.

Command Format: ATCT,<Temp_Celsius>

Command Parameter(s):

- **Temp_Celsius:** Current temperature in Celsius.

Example(s):

```
COMMAND:  ATCT,25<cr>
RESPONSE: <cr_lf>
          OK
```

8.8 Advertising

Advertising can now be done using either legacy non-extended advertising with a maximum packet size of 31 bytes or using the new extended advertising packets with a maximum packet size of 255 bytes. Extended advertising takes place in 2 stages and a different PHY can be used for each stage. First ADV_EXT_IND packets are sent out on the primary PHY on the standard advertising channels (37, 38 and 39). These packets do not include any advertising data but indicate to the receiver what secondary PHY and channel the extended advertising packet will be sent on. The actual extended advertising packet will go out on one of channels 0-36, which are the same channels used for data during a connection. For the primary PHY either 1M or LR can be selected, but the 2M PHY cannot be used as the primary PHY. For the secondary PHY any PHY may be selected. If a connection is made to a device that is advertising using a connectable extended advertisement, then the secondary PHY is the PHY that the connection will be established on.

Extended advertising is enabled by selecting one of the extended Ad_Types in the ATSDSLE command. This command is also used to configure the Primary and Secondary PHYs to use for extended advertising. The extended advertising data can be set using the ATSDSDLE command.

With extended advertising there are no separate advertising and scan response packets, only one packet of data. Selecting Ad_Type 5 (Extended Scannable Indirect Advertisement) will use the same extended advertising data as a non-scannable Ad_Type, but it will make it so the remote device must actively request the data so that the local device could use whitelisting to filter who can receive the data.

8.8.1 Advertise (ATDSLE)

DIAL AS SLAVE (ADVERTISE)

Function: This command places the module in the advertising state, allowing it to be discovered / connected to by other LE devices. Depending on the ATSDSLE Ad_Type, this can also make the module scannable and/or connectable.

If a Direct_Addr is supplied the module will advertise using directed advertising, which will send connectable advertisements targeted at a specific master. This advertising mode is meant to be used to reconnect to a master. The master would typically be configured to always be attempting to connect to devices in its whitelist so as soon as the directed advertisement was seen the connection would be made.

The type of direct advertising used will depend on the ATSDSLE Ad_Type. If one of the non-extended advertising modes is used, then Directed Connectable Non-Scannable advertising will be used. The Duty_Cycle parameter can be used to specify whether to use high duty cycle mode. This type of advertising does not contain any data and automatically timeout after 1.28s regardless of the ATSDSTLE Ad_Timeout setting.

If one of the extended advertising modes is used, then the directed form of that Ad_Type will be used. Extended directed advertising will not timeout after 1.28s and will use the ATSDSTLE Ad_Timeout settings.

Command Format: ATDSLE,<Direct_Addr>,<Duty_Cycle>

Command Parameter(s):

- **Direct_Addr:** Master address and address type for connectable direct advertising. If not specified, then indirect advertising will be used.

- **Direct_Duty_Cycle:** (Only applies to non-extended advertising)
  0 = Low Duty Cycle – A power efficient directed advertising mode for cases where reconnection with a specific device is required, but time is not of the essence or it is not known if the central device is in range or not. A 10ms advertising interval is used in this mode.
  1 = High Duty Cycle – A power intensive directed advertising mode for cases in which fast connection setup is essential. A 3.75ms advertising interval is used in this mode.

Example(s):

1. Indirect Advertising

```
COMMAND:  ATDSLE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

2. Directed Advertising

```
COMMAND:  ATDSLE,ECFE7E000000:0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *LE modules in the slave role can only be connected to one master at a time.*

8.8.2 Advertising Configuration (ATSDSLE)

SET ADVERTISING

Function: This command is used to configure the advertising (ATDSLE) settings.

Command Format: ATSDSLE,<White_List_Filter>,<Ad_Type>,<Ad_Channel_Map>,
<Extended_Primary_PHY>,<Extended_Secondary_PHY>,
<Extended_Anonymous>

Command Parameter(s):

- **White_List_Filter:** The whitelist can be configured using the ATSWL command.
  0 = Disabled, allow scan and connect requests from all devices
  1 = Allow scan requests from White List devices only, connect requests from all devices
  2 = Allow scan requests from all devices, connect requests from White List devices only
  3 = Allow scan and connect requests from White List devices only

- **Ad_Type:**
  0 = Connectable advertisement (Connectable + Scannable)
  2 = Scannable advertisement (Scannable Only)
  3 = Non-connectable advertisement (Neither Connectable nor Scannable)
  4 = Extended connectable advertisement (Connectable Only)
  5 = Extended scannable advertisement (Scannable Only)
  6 = Extended non-connectable advertisement (Neither Connectable nor Scannable)

- **Channel_Map:**
  1 = 37
  2 = 38
  3 = 37,38
  4 = 39
  5 = 37,39
  6 = 38,39
  7 = 37,38,39

- **Extended_Primary_PHY:** (Only applies if using an extended Ad_Type)
  1 = 1M (1Mbps Standard)
  4 = LR (125Kbps Coded Long Range)

- **Extended_Secondary_PHY:** (Only applies if using an extended Ad_Type)
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  4 = LR (125Kbps Coded Long Range)

- **Extended_Anonymous:** (Only applies if using an extended Ad_Type)
  0 = Device address will be included in the advertising packet.
  1 = Device address will not be included in the advertising packet.

Example(s):

```
COMMAND:  ATSDSLE,0,0,7,1,1,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *To use connectable direct advertising, use ATSDSDLE.*
- *This command cannot be used when the module is in the Advertising State.*

GET ADVERTISING

Function: Gets the advertising (ATDSLE) settings.

Command Format: ATSDSLE?

Response Format: <White_List_Filter>,<Ad_Type>,<Ad_Channel_Map>,
<Extended_Primary_PHY>,<Extended_Secondary_PHY>,<Extended_Anonymous>

Example(s):

```
COMMAND:  ATSDSLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,0,7,1,1,0<cr_lf>
```

8.8.3 Advertising Timing Configuration (ATSDSTLE)

SET ADVERTISING TIMING

Function: Sets a module's advertising (ATDSLE) timing parameters.

Command Format: ATSDSTLE,<Ad_Timeout>,<Unconnected_Ad_Interval>,
<Connected_Ad_Interval>

Command Parameter(s):

- **Ad_Timeout:** Integer value from 0 to 180 [s]. A value of 0 will disable the timeout and set the General Discoverable Flag in the advertising flags structure. A non-zero value will set the Limited Discoverable Flag.
  Default: 0 (No Timeout/General Discoverable Mode)

- **Unconnected_Ad_Interval:** Integer value from 32 to 16384 [.625ms]. This is the interval that advertisements will be sent when the module is not connected.
  Default: 160 (100ms)

- **Connected_Ad_Interval:** Integer value from 32 to 16384 [.625ms]. This is the interval that advertisements will be sent when the module is connected.
  Default: 2048 (1280ms)

Example(s):

```
COMMAND:  ATSDSTLE,0,160,2048<cr>
RESPONSE: <cr_lf>
          OK<cr_lf
```

Note(s):

- *The Flags Ad_Structure in the advertising data will automatically be updated to reflect the discoverable mode based on the value of Ad_Timeout.*

- *This command cannot be used when the module is in the Advertising State.*

GET ADVERTISING TIMING

Function: Gets the module's advertising (ATDSLE) timing parameters.

Command Format: ATSDSTLE?

Response Format: <Ad_Timeout>,<Unconnected_Ad_Interval>,<Connected_Ad_Interval>

Example(s):

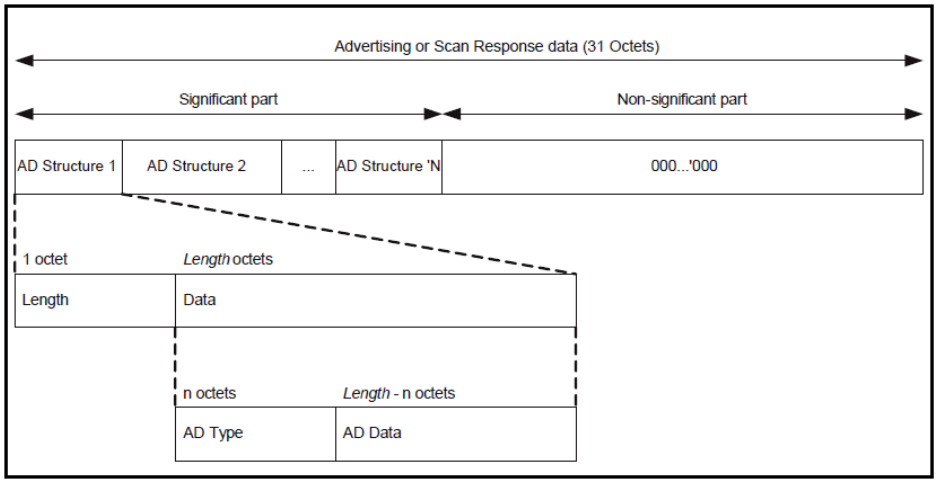
```
COMMAND:  ATSDSTLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,160,2048<cr_lf>
```

8.8.4 Advertising/Scan Response Data (ATSDSDLE)

SET ADVERTISING/SCAN RESPONSE DATA

Function: This command is used to configure the module's advertising and scan response data. The advertising data is transmitted in an advertising packet at the advertising interval set in ATSDIT, the scan response data will only be sent if requested by a device performing an active scan. For this reason, any data that may change frequently should be put in the advertising data, while static data (such as the name) should be put in the scan response data.

Advertising and scan response data is made up of one or more advertising data structures (AD structures), as seen in the figure below. Each one consists of a different type of data which is defined by the AD Type.



The full list of AD Types can be found here:
https://bluetooth.org/Technical/AssignedNumbers/generic_access_profile.htm

Descriptions of each AD Type can be found in the following document:
https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=282152

Data Checks:

All data is checked to make sure the ad structure lengths match up with the data length, and an ERROR,01 will be generated if the data is determined to be invalid, but it is up to the user to make sure the data meets the requirements of the BT5.0 specification.

The Flags AD structure (AD Type 0x01, 3 bytes) is required to be the first AD structure in the advertising data. If it is not detected it will automatically be inserted at the front of the data. This does not apply to scan response data.

Manufacturer Specific Data:

The examples using an Ad Type of Manufacturer Specific Data (0xFF) use a Company Identifier (CID) of 0x0085, which belongs to BlueRadios, Inc. This value can be used by our clients for Manufacturer Specific Data, as long as the next byte in the data is 0xFF (as shown in the examples), which states that the following bytes are BlueRadios Client Specific Data. However, we do recommend that our clients get their own CID. The Bluetooth SIG gives a unique Company Identifier Code to each Bluetooth SIG

member company that requests one:
https://www.bluetooth.org/Technical/AssignedNumbers/identifiers.htm

Auto Populating Ad Data Structures:

Certain ad data structures can now be set to automatically populate their data by setting the data to 0xFF. The data will not only auto populate when the command is sent, but also if the value of the data is changed by another command.

The following AD Types support auto populating data structures:

- **Flags** = 0201FF
- **Name** = 1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

  A maximum of 20 bytes can be allocated to the name, but if the name is shorter the data will automatically be trimmed to the length of the name. If only part of the name is desired in the ad data the length can be changed, for example 5 bytes of the name would be: 0609FFFFFFFFFF.
- **16-bit Service List** = 0302FFFF

  This ad data structure will automatically list enabled 16-bit services such as HRS (Heart Rate) but DIS (Device Information) and BAS (Battery) will never be listed.
- **128-bit Service List** = 1107FFFFFFFFFFFFFFFFFFFFFFFFFFFF

  This ad data structure will automatically list the BRSP service, unless a 16-bit service is enabled such as HRS, then the 16-bit services will be listed instead. DIS (Device Information) and BAS (Battery) will never be listed.
- **TX Power Level** = 020AFF

  This ad data structure will automatically populate the TX Power Level in dBm set by the ATSPL command.
- **Slave Connection Interval Range** = 0512FFFFFFFF

  This ad data structure will automatically populate the min and man connection intervals set in the ATSDCP command.
- **Battery Level (Service Data Ad Type)** = 04160F18FF

  This ad data structure will automatically populate the current battery level.
- **Appearance** = 0319FFFF

  This ad data structure will automatically populate the appearance set by the ATSAPP command.
- **Advertising Interval** = 031AFFFF

  This ad data structure will automatically populate the current advertising interval set by the ATSDSTLE command.

- **Manufacturer Specific Data** = 05FFFFFFFFFF

  This ad data structure will automatically populate with BlueRadios manufacturer specific data identifying the module type.

Command Format: ATSDSDLE,<Data_Type>,<Data>

Command Parameter(s):

- **Data_Type:**
  0 = Advertising Data
  1 = Scan Response Data
  2 = Extended Advertising Data

- **Data:** Data is comprised of one or more ad structures formatted as ASCII Hex Data. An ad structure consists of a Length byte, an Ad Type byte and (Length – 1) Data bytes. If Data is set to 0, the default value will be used.

  Data_Type = Advertising Data – 3-31 bytes (6-62 characters). First 3 bytes must be Flags (Ad Type 0x01), if not they will be automatically inserted.

  Default: 0201FF020AFF0512FFFFFFFF1107FFFFFFFFFFFFFFFFFFFFFFFFFFFF

  This default data will auto populate: Flags-TX Power-Slave Connection Interval Range-128b UUID Service List
  020106020A0805120800100011077960 22A0BEAFC0BDDE487962F1842BDA

  Data_Type = Scan Response Data – 3-31 bytes (6-62 characters). If the value is set to 00 the scan response will be disabled.

  Default: 05FFFFFFFFFF1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

  This default data will auto populate: BlueRadios Manufacturer Specific Data-Name
  05FF850000001109426C7565526164696F73303030303031

  Data_Type = Extended Ad Data – If using connectable Ad Type 3-238 bytes (6-476 characters), non-connectable 3-255 bytes (6-510 characters). First 3 bytes must be Flags (Ad Type 0x01), if not they will be automatically inserted.

  Default: 0201FF020AFF0512FFFFFFFF1107FFFFFFFFFFFFFFFFFFFFFFFFFFFF
  05FFFFFFFFFF1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

  This default data will auto populate: Flags-TX Power-Slave Connection Interval Range-128b UUID Service List-BlueRadios Manufacturer Specific Data-Name
  020106020A0805120800100011077960 22A0BEAFC0BDDE487962F1842BDA05FF850000001109426C7565526164696F73303030303031

Example(s):

1. The flags auto structure followed by an ad structure of 7 bytes, with an Ad Type of FF (Manufacturer Specific Data) and 6 bytes of data (BlueRadios Company Identifier (CID) of 0x0085 followed by data of 010203).

```
COMMAND:  ATSDSDLE,0,0201FF07FF8500FF010203<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

2. The same data as in example 1, but without specifying the flags. Since the flags are automatically inserted at the front of the ad data, the actual advertising data will be: 02010607FF8500FF010203.

```
COMMAND:  ATSDSDLE,0,07FF8500FF010203<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
COMMAND:  ATSDSDLE?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0201FF07FF8500FF010203<cr_lf>
```

3. This example fails because the CID was set to BlueRadios CID of 0x0085, but the next byte was not set to 0xFF to specify BlueRadios Client Specific Data.

```
COMMAND:  ATSDSDLE,0,0201FF07FF850000010203<cr>
RESPONSE: <cr_lf>
          ERROR,01<cr_lf>
```

4. This example fails because the length was set to 7, but only 6 bytes of data were passed.

```
COMMAND:  ATSDSDLE,0,0201FF07FF8500FF0102<cr>
RESPONSE: <cr_lf>
          ERROR,01<cr_lf>
```

5. Shows how multiple ad structures can be used. It has the Manufacturer Specific Data structure, followed by a TX Power Level structure and a Slave Connection Interval Range structure.

```
COMMAND:  ATSDSDLE,1,06FF8500FF0102020A04051208001000<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

6. Same as example 5 but using auto populating structures for TX Power and Slave Connection Interval Range.

```
COMMAND:  ATSDSDLE,1,06FF8500FF0102020AFF0512FFFFFFFF<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

7. Setting Data to 0 sets the Data back to the default.

```
COMMAND:  ATSDSDLE,1,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
COMMAND:  ATSDSDLE?,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          05FFFFFFFFFF1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF<cr_lf>
```

Note(s):

- *When Auto_Flash is enabled in ATSFC, the ad/scan data will be stored in flash each time this command is called. Advertising will be automatically cancelled and restarted to store the data in flash, so a new advertising packet will go out and the interval will start again from that point.*
- *If ad data needs to be changed frequently it is recommended that auto flashing be disabled by setting Auto_Flash to 0 in ATSFC to prevent writing to flash every time the ad data is changed.*
- *When not using auto populating data for the flags structure and trying to advertise using white list filtering with ATSDSLE White List Filter, neither the Limited Discoverable nor General Discoverable flag can be set.*

GET ADVERTISING DATA

Function: Gets the ATSDSDLE command settings.

Command Format: ATSDSDLE?,<Data_Type>

Command Parameter(s):

- **Data_Type:**
  0 = Advertising Data
  1 = Scan Response Data
  2 = Extended Advertising Data

Response Format: <Data>

Example(s):

1. Reading back the Advertising Data, after being written in Example 1 above. Notice how the flags have automatically been added.

```
COMMAND:  ATSDSDLE?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          02010607FF8500FF010203<cr_lf>
```

2. Reading back the Scan Response Data, after being written in Example 4 above.

```
COMMAND:  ATSDSDLE?,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          06FF8500FF0102020A04051208001000<cr_lf>
```

8.9 Discovery

- **Active vs. Passive Mode –** A master in the discovering state will receive advertising packets from slaves in the advertising state. In a passive scan the master will only print the data contained in these advertising packets. In an active scan, after receiving an advertising packet, the master will issue a scan request and ask the slave for additional data. The slave will then respond with a scan response.

- **Extended Advertising –** Whether the module receives extended advertising PHYs or not can be controlled through the Extended_Primary_Phys parameter.

8.9.1 Discovery (ATDILE)

DISCOVERY

Function: This command is used to discover nearby advertising LE devices. See the Discovery event section for information on the data returned in the Discovery events

Command Format: ATDILE,<Extended_Primary_Phys>

Command Parameter(s):

- **Extended_Primary_Phys:**
  Sets the PHYs to use when scanning for extended advertising packets on the primary advertising channels. All PHYs will automatically be used for scanning on the secondary advertising channels. The 2M PHY cannot be used for advertising on the primary advertising channels so the 2M PHY is not allowed. When scanning for extended advertising packets is enabled, non-extended advertising packets will still be received as well.

  0 = The module will not receive extended advertising packets.
  1 = The module will scan for extended advertising packets on the 1M PHY.
  4 = The module will scan for extended advertising packets on the LR PHY.
  5 = The module will scan for extended advertising packets on the 1M and LR PHYs. If this option is used the discovery timing interval must be greater than or equal to twice the window. If the settings in ATSDITLE do not meet this requirement the interval will automatically be increased but the setting will not be modified.

Example(s):

1. The ATDILE command is used to start an LE discovery and two devices are found, ECFE7E000001 and ECFE7E000002. ECFE7E000001 is advertising Connectable + Scannable, so two DISCOVERY events are sent for it, the first for the advertising data and the second for the scan response data. The ad data contains 4 ad data structures (Flags, TX Power, Slave Connection Interval Range, and BRSP Service), and the scan data contains 1 ad structure (Complete Local Name). ECFE7E000002 is advertising Connectable only, so only one DISCOVERY event is sent for advertising data:

```
COMMAND:  ATDILE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,2,ECFE7E000001:0,-45,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,6,ECFE7E000001:0,-45,1,
          1109426C7565526164696F73303030303031<cr_lf>
```

```
EVENT:    <cr_lf>
          DISCOVERY,3,ECFE7E000002:0,-45,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DONE,1,1<cr_lf>
```

Note(s):

- *An OK response is returned immediately following this command. A DONE event will appear after all devices have been found, or an inquiry timeout has occurred while searching for the number of devices specified.*
- *If multiple devices are found the scan response of a specific device is not guaranteed to show up immediately following its advertisement data.*

8.9.2 Discovery Configuration (ATSDILE)

SET DISCOVERY

Function: This command is used to configure the discovery (ATDILE) command settings.

Command Format: ATSDILE,<White_List_Filter>,<Limited_Filter>,<Active_Scan>,
<Max_Devices>,<Paired_Filter>,<Channel_Map>

Command Parameter(s):

- **White_List_Filter:** The whitelist can be configured using the ATSWL command.
  0 = Disabled
  1 = Discover White List devices only.
- **Limited_Filter:**
  0 = Disabled (Discover all discoverable devices)
  1 = Discover only limited discoverable devices. (Devices only discoverable for a limited time)
- **Active_Scan:**
  0 = Disabled
  1 = Request scan response data from discovered devices.
- **Max_Devices:** Maximum number of devices to discover [0-50], Default: 10
  0 = Discovery will run until the ATSDITLE Timeout is reached, even if the maximum number of 10 devices is found. Additionally, if a discovered device updates its advertising or scan response data, a new discovery event will be printed with the updated data.
  1-50 = Discovery will run until either Max_Devices are found or the ATSDITLE Timeout is reached. Updated advertising or scan response data will not be printed.
- **Paired_Filter:**
  0 = Disabled
  1 = Discover only paired devices. (Filtering is done at the application layer, as opposed to the link layer for White_List_Filter and Limited_Filter, so if unpaired devices are nearby the discovery may end prior to Max_Devices discovery events being sent.)

- **Channel_Map:**
  1 = 37
  2 = 38
  3 = 37,38
  4 = 39
  5 = 37,39
  6 = 38,39
  7 = 37,38,39

Example(s):

```
COMMAND:  ATSDILE,0,0,1,10,0,7<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used when the module is in the Discovering State.*

GET DISCOVERY LE

Function: Gets the discovery (ATSDILE) command settings.

Command Format: ATSDILE?

Response Format:<White_List_Filter>,<Limited_Filter>,<Active_Scan>,<Max_Devices>,
<Paired_Filter>,<Channel_Map>

Example(s):

```
COMMAND:  ATSDILE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,0,1,10,0,7<cr_lf>
```

8.9.3 Discovery Timing Configuration (ATSDITLE)

SET DISCOVERY TIMING

Function: Sets the module's discovery (ATDILE) timing parameters. The Interval and Window set the scanning duty cycle used during discovery. The Interval sets how frequently the module should scan and the Window sets how long it will scan at each Interval. If both are set to the same value, scanning will run continuously during the discovery.

Command Format: ATSDITLE,<Timeout>,<Interval>,<Window>

Command Parameter(s):

- **Timeout:** Integer value from 1 to 65535 [s]. A value of 0 can be used to disable the timeout.
  Default: 10
- **Interval:** Integer value from 4 to 65535 [.625ms].
  Default: 16 (10ms)
- **Window:** Integer value from 4 to 65535 [.625ms].
  Default: 16 (10ms)

Example(s):

```
COMMAND:  ATSDITLE,10240,16,16<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used when the module is in the Discovering State.*

GET DISCOVERY TIMING LE

Function: Gets the module's discovery (ATDILE) timing parameters.

Command Format: ATSDITLE?

Response Format: <Timeout>,<Interval>,<Window>

Example(s):

```
COMMAND:  ATSDITLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          10240,16,16<cr_lf>
```

8.9.4 Discovery Event Formatting (ATSDIF)

SET DISCOVERY FORMATTING

Function: Sets discovery event formatting parameters.

Command Format: ATSDIF,<Data_Structure_Flags>,<Name_Format>,<Hyphens>,<Discovery_Type_Format>,<Global_Format>

Command Parameter(s):

- **Data_Structure_Flags:** Data structures enabled in the mask will be printed in discovery events. If set to 0, no data structures will be printed.

Decimal Value	Hex Value	Response / Event
1	0x0001	Flags
2	0x0002	Services
4	0x0004	Local Name
8	0x0008	TX Power Level
16	0x0010	Simple Pairing Optional OOB Tags
32	0x0020	Device ID
64	0x0040	Security Manager OOB Flags
128	0x0080	Slave Connection Interval Range
256	0x0100	Service Solicitation
512	0x0200	Service Data
1024	0x0400	Target Address
2048	0x0800	Appearance
4096	0x1000	Advertising Interval
32768	0x8000	Manufacturer Specific Data
65535	0xFFFF	All Data Structures Enabled

- **Name_Format:**
  0 = Format as Data
  1 = Format as String

- **Hyphens:**
  0 = Disabled
  1 = Enabled, hyphens will be used to separate data structures

- **Discovery_Type_Format:**
  0 = Standard Format – The Discovery_Type will use the decimal values listed in the DISCOVERY and EXT_DISCOVERY event descriptions.

  1 = Hex Bit Field Format – **[Hex Formatted]** The Discovery_Type will be a hex character composed of the Bitfields shown below. For example, an extended connectable advertisement would be 0x11.

Bit	4	3	2	1	0
Value	Extended	Scan Response	Directed	Scannable	Connectable

2 = String Format – The Discovery_Type will be a string composed of a combination of the strings in the table below. For example, an extended connectable advertisement would be: EC

String	Value
C	Connectable
NC	Non-Connectable
S	Scannable
NS	Non-Scannable
SR	Scan Response
D	Directed
E	Extended

- **Global_Format:**
  0 = Separate events for non-extended and extended discovery events
  1 = All discovery events will use the extended discovery format with the DISCOVERY event tag

Example(s):

```
COMMAND:  ATSDIF,65535,1,1,0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command only changes how DISCOVERY and EXT_DISCOVERY events are formatted; it does not change the module's ad or scan response data.*

GET DISCOVERY FORMATTING

Function: Gets discovery event formatting parameters.

Command Format: ATSDIF?

Response Format:
<Data_Structure_Mask>,<Name_Format>,<Hyphens>,<Discovery_Type_Format>,<Global_Format>

Example(s):

```
COMMAND:  ATSDIF?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          65535,1,1,0,0<cr_lf>
```

8.10 Connection

- **Extended Advertising –** Whether the module can connect to a device using extended advertising PHYs or not can be controlled through the Extended_Primary_Phys parameter.

8.10.1 Connect (ATDMLE)

DIAL AS MASTER (CONNECT)

Function: This command initiates a connection to a slave device. If the command is an accepted an **"OK"** will be sent back immediately. Do not mistake this for a connection being complete. A completed connection will return a CONNECT event sometime after the command was sent. PIO_2 will go active when a BT5.0 connection is established.

Command Format: ATDMLE,<Addr>,<BRSP_Mode>,<Extended_Primary_Phys>

Command Parameter(s):

- **Addr:** Slave device address and address type. Set to "WL" to use the White List instead of a direct connection.

- **BRSP_Mode:**
  0 = Command Mode – The module will stay in command mode upon connecting.
  1 = BRSP Data Mode – The module will go into BRSP data mode upon connecting.
  2 = BRSP Remote Command Mode – The module will go into BRSP remote command mode upon connecting.

- **Extended_Primary_Phys:**
  Sets the PHYs to scan on the primary advertising channels when connecting to a device using extended advertising packets. All PHYs will automatically be used for scanning on the secondary advertising channels. The 2M PHY cannot be used for advertising on the primary advertising channels so the 2M PHY is not allowed. When scanning for extended advertising packets is enabled for the 1M PHY, a connection can be made to a device advertising using non-extended advertising packets on the 1M PHY as well.

  0 = The device to connect to is using non-extended advertising packets.
  1 = The device to connect to is using extended advertising packets on the 1M PHY.
  4 = The device to connect to is using extended advertising packets on the LR PHY.
  5 = The module will scan for extended advertising packets on both the 1M and LR PHYs. If this option is used the connect timing interval must be greater than or equal to twice the window. If the settings in ATSDMTLE do not meet this requirement the interval will automatically be increased but the setting will not be modified.

Example(s):

- The ATDMLE command is used to initiate an LE connection and once connected the CONNECT event is sent. The connected device is an LE device, that is not paired with the module and has an address of ECFE7E000001:0.

```
COMMAND:   ATDMLE,ECFE7E000001:0,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:     <cr_lf>
           MTU,0,247<cr_lf>
```

- The ATDMLE command is used to initiate an LE connection with BRSP_Mode set to Data Mode and once connected the CONNECT event is sent. Once Data Mode is ready, the BRSP,0,1 message is sent.

```
COMMAND:  ATDMLE,ECFE7E000001:0,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
EVENT:    <cr_lf>
          BRSP,0,1,0,0<cr_lf>
```

- The ATDMLE command is used to initiate an LE connection, but the device is not found and the request times out, triggering a DONE event.

```
COMMAND:  ATDMLE,ECFE7E000001:0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          DONE,1,2<cr_lf>
```

Note(s):

- *A single mode module can only connect in the LE master role to one device at a time. A dual mode module can connect to up to 4 LE slaves.*
- *If the remote Slave device is not present, a DONE event will occur after the connect timeout. By default, the connect timeout is set to 0 in ATSDMTLE so the connection attempt will never timeout, but it can be cancelled using ATDC.*
- *A discovery cannot be done while in the connecting state so ATDILE will return ERROR,03 when connecting. Use an ATDC to first cancel the connection attempt prior to executing an ATDILE.*
- *When connecting out to a device that has been RESOLVED, it is best to perform a discovery first prior to connecting. When the device is found during the discovery, the module will automatically detect a private address change allowing it to connect out to the correct address using ATDMLE. See the RESOLVED event for more information.*

8.10.2 Connect Last (ATDMLLE)

DIAL AS MASTER LAST (CONNECT LAST)

Function: Initiates a connection to the LE device address from the last LE CONNECT event. This address can be read using the ATLCALE?

Command Format: ATDMLLE,<BRSP_Mode>

Command Parameter(s):

- **BRSP_Mode:**
  0 = Command Mode – The module will stay in command mode upon connecting.
  1 = BRSP Data Mode – The module will go into BRSP data mode upon connecting.
  2 = BRSP Remote Command Mode – The module will go into BRSP remote command mode upon connecting.

Example(s):

```
COMMAND:  ATDMLLE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          BRSP,1<cr_lf>
```

Note(s):

- *To verify the last address that will be connected to use the ATLCALE? Command.*
- *The module can connect out to up to 4 devices at a time.*
- *If the remote Slave device is not present, a DONE event will occur after the connect timeout. By default, the connect timeout is set to 0 in ATSDMTLE so the connection attempt will never timeout, but it can be cancelled using ATDC.*
- *A discovery cannot be done while in the connecting state so ATDILE will return ERROR,03 when connecting. Use an ATDC to first cancel the connection attempt prior to executing an ATDILE.*
- *When connecting out to a device that has been RESOLVED, it is best to perform a discovery first prior to connecting. When the device is found during the discovery, the module will automatically detect a private address change allowing it to connect out to the correct address using ATDMLE. See the RESOLVED event for more information.*

8.10.3 Last Connect Address (ATLCALE?)

GET LAST CONNECT ADDRESS

Function: Gets the BT5.0 device address from the last LE CONNECT event.

Command Format: ATLCALE?

Response Format: <Addr>

Response Values:

- **Addr:** Last connected BT5.0 device address and address type.

Example(s):

```
COMMAND:  ATLCALE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          ECFE7E000001:0<cr_lf>
```

8.10.4 Connect Configuration (ATSDMLE)

SET CONNECT CONFIGURATION

Function: Sets the connection command parameters.

Command Format: ATSDMLE,<Channel_Map>,<BRSP_On_Connect>

Command Parameter(s):

- **Channel_Map:**
  1 = 37
  2 = 38
  3 = 37,38
  4 = 39
  5 = 37,39
  6 = 38,39
  7 = 37,38,39

- **BRSP_On_Connect:**
  Integer value from 1 to 65535 [s]. A value of 0 can be used to disable the timeout.
  0 = No BRSP On Connect – The module will stay in command mode upon connecting.
  1 = BRSP Data Mode – The module will go into BRSP data mode upon connecting.
  2 = BRSP Remote Command Mode – The module will go into BRSP remote command mode upon connecting.

Example(s):

```
COMMAND:  ATSDMLE,7,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used when the module is in the Connecting State.*

GET CONNECT CONFIGURATION LE

Function: Gets the connection initiation timing parameters.

Command Format: ATSDMLE?

Response Format: <Channel_Map>,<BRSP_On_Connect>

Example(s):

```
COMMAND:  ATSDMLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          7,1<cr_lf>
```

8.10.5 Connect Timing Configuration (ATSDMTLE)

SET CONNECT TIMING

Function: Sets the connection initiation timing parameters. The Interval and Window set the scanning duty cycle used during connection establishment. The Interval sets how frequently the module should scan and the Window sets how long it will scan at each Interval. If both are set to the same value, scanning will run continuously while in the Connecting State.

Command Format: ATSDMTLE,<Timeout>,<Interval>,<Window>

Command Parameter(s):

- **Timeout:** Integer value from 1 to 65535 [s]. A value of 0 can be used to disable the timeout.
  Default: 0 (No Timeout)
- **Interval:** Integer value from 4 to 65535 [.625ms].
  Default: 16 (10ms)
- **Window:** Integer value from 4 to 65535 [.625ms].
  Default: 16 (10ms)

Example(s):

```
COMMAND:  ATSDMTLE,0,16,16<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used when the module is in the Connecting State.*

GET CONNECT TIMING LE

Function: Gets the connection initiation timing parameters.

Command Format: ATSDMTLE?

Response Format: <Timeout>,<Interval>,<Window>

Example(s):

```
COMMAND:  ATSDMTLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,16,16<cr_lf>
```

8.10.6 Connection Parameters

The connection parameters control how often two devices in a connection will meet up to exchange data in a connection event. They control the balance between throughput and power savings.

- **Connection Interval –** The amount of time between two connection events. A shorter connection interval will result in higher throughput, lower latency, but will use more power, and a longer connection interval will result in lower throughput, higher latency, and less power consumption.
- **Slave Latency –** Allows a slave to skip a set number of connection events. This allows the slave to save power if it has no data to send. This will not affect the latency of data sent from slave to master but will result in increased latency from master to slave. The slave latency must not make the effective connection interval greater than 32s.
- **Supervision Timeout –** Amount of time allowed since the last successful connection event. If this timeout occurs the connection will be dropped.
- **Effective Connection Interval –** Amount of time between connection events, assuming the slave always skips [slave latency] connection events. It can be calculated with the following formula:

Effective Connection Interval **=** *(Connection Interval) * (1+ (Slave Latency))*

8.10.6.1 Default Connection Parameters (ATSDCP)

SET DEFAULT CONNECTION PARAMETERS

Function: Sets the module's default connection parameters.

Command Format: ATSDCP,<Min_Conn_Interval>,<Max_Conn_Interval>,<Slave_Latency>,
<Supervision_Timeout>,<Slave_Auto_Update>

Command Parameter(s):

- **Min_Conn_Interval:** Integer value from 6 to 3200 [1.25ms]. Can be set to 65535 (No specific interval preferred) on a slave role module.
  Default: 8 (10ms)

- **Max_Conn_Interval:** Integer value from 6 to 3200 [1.25ms]. Can be set to 65535 (No specific interval preferred) on a slave role module. Must be greater than or equal to Min_Conn_Interval.
  Default: 16 (20ms)

- **Slave_Latency:** Integer value from 0 to 499 [connection intervals].
  Default: 0

- **Supervision_Timeout:** Integer value from 10 to 3200 [10ms].
  Must be greater than 2 * (Max_Conn_Interval * (1+Slave_Latency)).
  Default: 400 (4s)

- **Slave_Auto_Update:**
  0 = If connected to as a slave, the module will accept any connection parameters.
  1-65534 = If connected to as a slave and the connection parameters do not match the module's default connection parameters, the module will automatically request a connection parameter update after waiting the 1-65534 ms. 5 seconds is recommended.
  65535 = If connected to as a slave and the connection parameters do not match the module's default connection parameters, the module will automatically request a connection parameter update after the pairing process is complete and the link is encrypted.

Example(s):

```
COMMAND:  ATSDCP,8,16,0,400,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *The default connection parameters will be placed in the advertising data and the GAP service data as the preferred connection parameters.*

- *Effective Connection Interval = Connection Interval * (1+Slave Latency)*

- *These settings will not take effect on an existing connection, use ATSCCP to update an existing connection.*

GET DEFAULT CONNECTION PARAMETERS

Function: Gets the module's default connection parameters.

Command Format: ATSDCP?

Response Format: <Min_Conn_Interval>,<Max_Conn_Interval>,<Slave_Latency>,
<Supervision_Timeout>,<Slave_Auto_Update>

Example(s):

```
COMMAND:  ATSDCP?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          8,16,0,400,0<cr_lf>
```

8.10.6.2 Current Connection Parameters (ATSCCP)

SET CURRENT CONNECTION PARAMETERS

Function: Attempts to update the current connection parameters for an existing connection.

If the module is in the master role of a connection, the connection parameter update will always be successful if the command is accepted and a CPU event will be generated when the parameters have been changed.

If the module is in the slave role, the master can accept or reject the request to change the parameters. An SCCPS event will be sent to let the user know if the update request was accepted or rejected by the master. If the request is accepted than a CPU event will be generated when the parameters have changed.

Command Format: ATSCCP,<Conn_Handle>,<Min_Conn_Interval>,<Max_Conn_Interval>,
<Slave_Latency>,<Supervision_Timeout>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to update.
- **Min_Conn_Interval:** Integer value from 6 to 3200 [1.25ms]. Can be set to 65535 (No specific interval preferred) on a slave role module.
- **Max_Conn_Interval:** Integer value from 6 to 3200 [1.25ms]. Can be set to 65535 (No specific interval preferred) on a slave role module. Must be greater than or equal to Min_Conn_Interval.
- **Slave_Latency:** Integer value from 0 to 499 [connection intervals].
- **Supervision_Timeout:** Integer value from 10 to 3200 [10ms].

  Must be greater than 2 * (Max_Conn_Interval * (1+Slave_Latency)).

Example(s):

1. The module is in the master role and a connection parameter update is requested on connection handle 0. The CPU event is sent when the connection parameters have been updated:

```
COMMAND:  ATSCCP,0,8,8,0,400<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CPU,0,8,0,400<cr_lf>
```

2. The module is in the slave role and a connection parameter update is requested on connection handle 0. The SCCPS event is sent, confirming that the update was accepted and then the CPU event is sent when the connection parameters have been updated:

```
COMMAND:  ATSCCP,0,8,8,0,400<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          SCCPS,0,0<cr_lf>
EVENT:    <cr_lf>
          CPU,0,8,0,400<cr_lf>
```

Note(s):

- *Effective Connection Interval = Connection Interval * (1+Slave Latency)*

GET CURRENT CONNECTION PARAMETERS

Function: Gets the current connection parameters for an existing connection.

Command Format: ATSCCP?,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to read.

Response Format: <Conn_Interval>,<Slave_Latency>,<Supervision_Timeout>

Example(s):

```
COMMAND:  ATSCCP?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          8,0,400<cr_lf>
```

8.10.7 Default MTU (ATSDMTU)

SET DEFAULT MTU

Function: Sets the module's default MTU (Maximum Transmission Unit).

Command Format: ATSDMTU,<MTU>

Command Parameter(s):

- **MTU:** Integer value from 23 to 247 bytes. Default: 247

Example(s):

```
COMMAND:  ATSDMTU,247<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This setting will not take effect on an existing connection.*
- *Lowering the MTU from the default value of 247 will lower the maximum throughput in BRSP data mode.*

GET DEFAULT MTU

Function: Gets the module's default MTU.

Command Format: ATSDMTU?

Response Format: <MTU>

Example(s):

```
COMMAND:  ATSDMTU?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          247<cr_lf>
```

8.10.8 Connection PHYS

8.10.8.1 Supported PHYS (ATSSPHY)

SET SUPPORTED PHYS

Function: Sets the PHYs the module will accept if a connection PHY update is requested by another device. The supported PHYs can be set separately for RX and TX.

Command Format: ATSSPHY,<RX_PHYS>,<TX_PHYS>

Command Parameter(s):

- **RX_PHYS:** Default: 7
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  3 = 1M, 2M
  4 = LR (125Kbps Coded Long Range)
  5 = 1M, LR
  6 = 2M, LR
  7 = 1M, 2M, LR

- **TX_PHYS:** Default: 7
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  3 = 1M, 2M
  4 = LR (125Kbps Coded Long Range)
  5 = 1M, LR
  6 = 2M, LR
  7 = 1M, 2M, LR

Example(s):

```
COMMAND:  ATSSPHY,7,7<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET SUPPORTED PHYS

Function: Gets the module's supported PHYs.

Command Format: ATSSPHY?

Response Format: <RX_PHYS>,<TX_PHYS>

- **RX_PHYS:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  3 = 1M, 2M
  4 = LR (125Kbps Coded Long Range)
  5 = 1M, LR
  6 = 2M, LR
  7 = 1M, 2M, LR

- **TX_PHYS:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  3 = 1M, 2M
  4 = LR (125Kbps Coded Long Range)
  5 = 1M, LR
  6 = 2M, LR
  7 = 1M, 2M, LR

Example(s):

```
COMMAND:  ATSSPHY?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          7,7<cr_lf>
```

8.10.8.2 Current PHYS (ATSCPHY)

SET CURRENT PHYS

Function: Requests to change the PHY for the current connection. A separate PHY can be requested for RX and TX. A PHYU event will happen after the request has been acknowledged, if a requested PHY was not supported by the remote device the PHY will not have changed in the PHYU event.

Command Format: ATSCPHY,<Conn_Handle>,<RX_PHY>,<TX_PHY>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to set.

- **RX_PHY:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  4 = LR (125Kbps Coded Long Range)

- **TX_PHY:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  4 = LR (125Kbps Coded Long Range)

Example(s):

1. A PHY update to 2M is requested on connection handle 0 and succeeds.

```
COMMAND:  ATSCPHY,0,2,2<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PHYU,0,2,2<cr_lf>
```

2. A PHY update to 2M is requested on connection handle 0 and fails, the PHY stays at 1M.

```
COMMAND:  ATSCPHY,0,2,2<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PHYU,0,1,1<cr_lf>
```

GET CURRENT PHY

Function: Gets the current PHYs being used for a connection.

Command Format: ATSCPHY?,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to read.

Response Format: <RX_PHY>,<TX_PHY>

Response Values:

- **RX_PHY:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  4 = LR (125Kbps Coded Long Range)

- **TX_PHY:**
  1 = 1M (1Mbps Standard)
  2 = 2M (2Mbps High Speed)
  4 = LR (125Kbps Coded Long Range)

Example(s):

```
COMMAND:  ATSCPHY?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          2,2<cr_lf>
```

8.10.9 Connection Status (ATCS?)

GET CONNECTION STATUS

Function: Gets the details of an existing connection. If no connection handle is specified, then the connection status for all open connections will be returned followed by a DONE event to signal the end of the connection list. A DONE event will not be sent if status is requested for a specific connection.

Command Format: ATCS?,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to read. If not specified, will print the connection status of all active connections.

Response Format: <Conn_Handle>,<Conn_Role>,<Security_Level>,<Addr>,<Conn_Interval>,<Slave_Latency>,<Supervision_Timeout>,<MTU>,<RX_PHY>,<TX_PHY>,<TX_Power>

Response Values:

- **Conn_Handle:** Connection handle
- **Conn_Role:**
  1 = LE Peripheral
  2 = LE Central
- **Security_Level:**
  1 = No Security
  2 = Encrypted, Not Authenticated
  3 = Encrypted, Authenticated
  4 = Encrypted, LESC Authenticated
- **Addr:** Address and address type of remote device.
- **Conn_Interval:** Integer value from 6 to 3200 [1.25ms].
- **Slave_Latency:** Integer value from 0 to 499 [connection intervals].
- **Supervision_Timeout:** Integer value from 10 to 3200 [10ms].
- **MTU:** 23-247
- **RX_PHY:**
  0 = Unknown (See note below)
  1 = 1 Mbps
  2 = 2 Mbps
  4 = 125 kbps Coded (Long Range)
- **TX_PHY:**
  0 = Unknown (See note below)
  1 = 1 Mbps
  2 = 2 Mbps
  4 = 125 kbps Coded (Long Range)
- **TX_Power:** Transmit power level in dBm.

Example(s):

1. Connection status for connection handle 0 is requested, it is an unpaired LE connection to ECFE7E000000.

```
COMMAND:  ATCS?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,2,1,ECFE7E000000:0,16,0,400,247,1,1,8<cr_lf>
```

2. Connection status for all connections is requested. One active connection is found, it is an unpaired LE connection to ECFE7E00000. Then the DONE event is triggered to signal the end of the connection list.

```
COMMAND:  ATCS?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,2,1,ECFE7E000000:0,16,0,400,247,1,1,8<cr_lf>
          <cr_lf>
          DONE,2,0<cr_lf>
```

3. Connection status for all connections is requested. Two active connections are found. Then the DONE event is triggered to signal the end of the connection list.

```
COMMAND:  ATCS?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,2,1,ECFE7E000000:0,16,0,400,247,1,1,8<cr_lf>
          <cr_lf>
          1,2,1,ECFE7E000001:0,16,0,400,247,1,1,8<cr_lf>
          <cr_lf>
          DONE,2,0<cr_lf>
```

Note(s):

- *RX_PHY and TX_PHY will be unknown after as a Central (ATDMLE) to a device advertising using extended advertising packets until a PHY Update event occurs. A PHY Update can be requested using the ATSCPHY command after the connection has been established. This issue will be resolved in a future release.*

8.10.10 Connection RSSI (ATRSSI?)

GET CONNECTION RSSI VALUE

Function: Used to obtain the Received Signal Strength Indication (RSSI) value of the last packet received when connected. The radio has a built-in received signal-strength indication, which calculates an 8-bit signed digital value that is the result of averaging the received power over eight symbol periods (128µs). The RSSI is a signed number on a logarithmic scale with 1-dB steps and a ±6 dBm accuracy.

If no connection handle is specified, then the RSSI for all open connections will be returned followed by a DONE event to signal the end of the connection list. A DONE event will not be sent if RSSI is requested for a specific connection.

Command Format: ATRSSI?,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to read RSSI from. If not specified, the RSSI for all active connections will be printed.

Response Format: <RSSI_Value>

Response Values:

- **RSSI_Value:** Absolute receiver RSSI value in dBm. If the RSSI cannot be read, 127. [-127 – +20 dBm, 127]

Example(s):

1. RSSI for connection handle 0 is requested.

```
COMMAND:  ATRSSI?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          RSSI,0,-34<cr_lf>
```

2. Connection status for all connections is requested and one active connection is found. Then the DONE event is triggered to signal the end of the RSSI list.

```
COMMAND:  ATRSSI?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          RSSI,0,-34<cr_lf>
          <cr_lf>
          DONE,2,0<cr_lf>
```

3. Connection status for all connections is requested and two active connections are found. Then the DONE event is triggered to signal the end of the connection list.

```
COMMAND:  ATRSSI?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          RSSI,0,-34<cr_lf>
          <cr_lf>
```

```
RSSI,1,-38<cr_lf>
<cr_lf>
DONE,2,0<cr_lf>
```

Note(s):

- *The module must be connected to read the RSSI.*

8.10.11 Continuous Connection RSSI (ATCRSSI)

CONTINUOUS CONNECTION RSSI

Function: Used to continuously obtain the Received Signal Strength Indication (RSSI) value of the last packet received when connected. The radio has a built-in received signal-strength indication, which calculates an 8-bit signed digital value that is the result of averaging the received power over eight symbol periods (128µs). The RSSI is a signed number on a logarithmic scale with 1-dB steps and a ±6 dBm accuracy.

Continuous RSSI can be cancelled using the ATDC command.

Command Format: ATCRSSI,<Conn_Handle>,<Event_Threshold>,<Skip_Count>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to read RSSI from. If not specified, the RSSI for all active connections will be continuously printed.
- **Event_Threshold:** Minimum change in dB before triggering a new RSSI event.
  Default: 0 dB
- **Skip_Count:** Number of samples with a change of Event_Threshold before triggering a new RSSI event.
  Default: 0

Example(s):

1. Continuous RSSI for connection handle 0 is requested and then cancelled with the ATDC command.

```
COMMAND:  ATCRSSI,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          RSSI,0,-34<cr_lf>
          <cr_lf>
          RSSI,0,-35<cr_lf>
          <cr_lf>
          ...
          <cr_lf>
COMMAND:  ATDC,3,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *The module must be connected to read the RSSI.*

8.10.12 Disconnect Command (ATDH)

DISCONNECT

Function: This command will terminate a specific connection or all active connections.

Command Format: ATDH,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to terminate. If not specified all active connections will be terminated.

Example(s):

1. ATDH is used to disconnect from connection handle 0, when the module has disconnected a DISCONNECT event is triggered.

```
COMMAND:  ATDH,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          DISCONNECT,0,0<cr_lf>
```

8.11 Pairing

8.11.1 Pair Command (ATPLE)

PAIR DEVICE

Function: This command is used to initiate LE pairing, which will enable encryption. It is also used to respond to a pairing request. The module must be in the connected state to use this command.

Command Format: ATPLE,<Addr_Conn_Handle>,<Accept_Request>

Command Parameter(s):

- **Addr_Conn_Handle:** Address and address type of device or connection handle if connected. If not specified, the lowest numbered active connection handle will be used.

- **Accept_Request:** (Only needed when responding to a PAIR_REQ event.)
  0 = Reject pairing request.
  1 = Accept pairing request.

Example(s):

1. LE pairing is initiated using the ATPLE command to connection handle 0. The remote device accepts the command and pairing is completed, triggering the PAIRED event:

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,ECFE7E000001:0,1<cr_lf>
```

Note(s):

- *Up to 32 devices can be paired.*

GET PAIRED DEVICES LE

Function: This command is used to read the paired LE devices.

Command Format: ATPLE?

Response Format: <Count>,<Addrs>

Response Value(s):

- **Count:** Number of paired devices
- **Addrs:** List of paired addresses, separated by '-' characters. 0 if Count is 0.

Example(s):

1. The module is not paired to any devices.

```
COMMAND:  ATPLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,0<cr_lf>
```

1. The module has been paired to two devices.

```
COMMAND:  ATPLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          2,ECFE7E000001:0-ECFE7E000002:0<cr_lf>
```

8.11.2 Pairing Configuration (ATSPLE)

PAIRING PARAMETERS

Function: This command is used to configure LE pairing.

Command Format: ATSPLE,<Request_Handling>,<Authentication>,<IO_Capabilities>,<Sync_White_List>,<Disconnect_Unpaired>,<LESC>

Command Parameter(s):

- **Request_Handling:**
  0 = Reject all pairing requests. (Disable Pairing)
  1 = Prompt for all pairing requests with PAIR_REQ event.
  2 = Automatically accept all pairing requests.

- **Authentication:**
  0 = Authentication not requested.
  1 = Request authentication (Man-in-the-Middle Protection). Requires that IO_Capabilities not be set to No Input or Output; else authentication cannot be performed.

  Enabling authentication does not guarantee that authentication will take place. The IO_Capabilities of the initiating and responding devices must be able to support the Passkey Entry method to authenticate. To support Passkey Entry at least one device needs a display and the other needs a numeric keyboard. All possible scenarios are shown in the IO Capability/Authentication Mapping table below with the cases where authentication will take place shown in green.

- **IO_Capabilities:**
  0 = Display Only
  1 = Display with Yes/No Buttons
  2 = Numeric Keyboard Only (No Display)
  3 = No Input or Output
  4 = Display and Numeric Keyboard

- **Sync_White_List:**
  0 = Disabled
  1 = Paired devices will automatically be added to the white list and removed if unpaired. To enable whitelist syncing, the module cannot be in the pairing, connecting, or advertising state. ATSWL, ATUWL and ATCWL will be disabled and return ERROR,3 if this option is enabled.

- **Disconnect_Unpaired:**
  0 = Disabled
  1 = Automatically disconnect unpaired devices that connect to the module.

- **LESC: LE Secure Connections**
  0 = Disable LE Secure Connections.
  1 = Enable LE Secure Connections. LE Secure Connections uses ECDH for key generation, so pairing is secure from passive eavesdroppers in all instances. Both devices that are pairing must support LESC for it to be used. When authentication is enabled and the IO_Capabilities support it, a passkey will need to be confirmed on both sides during the pairing process.

Example(s):

```
COMMAND:  ATSPLE,2,0,3<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Non-LESC IO Capability/Authentication Mapping:

Pairing Initiator → / Pairing Responder ↓	Display Only	Display with Yes/No Buttons	Numeric Keyboard Only	No Input or Output	Display and Numeric Keyboard
Display Only	Just Works: **No Auth**	Just Works: **No Auth**	Passkey Entry: responder displays, initiator inputs **Authenticated**	Just Works: **No Auth**	Passkey Entry: responder displays, initiator inputs **Authenticated**
Display with Yes/No Buttons	Just Works: **No Auth**	Just Works: **No Auth**	Passkey Entry: responder displays, initiator inputs **Authenticated**	Just Works: **No Auth**	Passkey Entry: responder displays, initiator inputs **Authenticated**
Numeric Keyboard Only	Passkey Entry: initiator displays, responder inputs **Authenticated**	Passkey Entry: initiator displays, responder inputs **Authenticated**	Passkey Entry: initiator and responder inputs **Authenticated**	Just Works: **No Auth**	Passkey Entry: initiator displays, responder inputs **Authenticated**
No Input or Output	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**
Display and Numeric Keyboard	Passkey Entry: initiator displays, responder inputs **Authenticated**	Passkey Entry: initiator displays, responder inputs **Authenticated**	Passkey Entry: responder displays, initiator inputs **Authenticated**	Just Works: **No Auth**	Passkey Entry: initiator displays, responder inputs **Authenticated**

LESC IO Capability/Authentication Mapping:

Pairing Initiator → / Pairing Responder ↓	Display Only	Display with Yes/No Buttons	Numeric Keyboard Only	No Input or Output	Display and Numeric Keyboard
Display Only	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**
Display with Yes/No Buttons	Just Works: **No Auth**	Numeric Comparison: both sides display and confirm **Authenticated**	Just Works: **No Auth**	Just Works: **No Auth**	Numeric Comparison: both sides display and confirm **Authenticated**
Numeric Keyboard Only	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**
No Input or Output	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: **No Auth**
Display and Numeric Keyboard	Just Works: **No Auth**	Numeric Comparison: both sides display and confirm	Just Works: **No Auth**	Just Works: **No Auth**	Numeric Comparison: both sides display and confirm

GET PAIRING LE

Function: Gets the ATSPLE settings.

Command Format: ATSPLE?

Response Format: <Request_Handling>,<Authentication>,<IO_Capabilities>,
<Sync_White_List>,<Disconnect_Unpaired>,<LESC>

Example(s):

```
COMMAND:  ATSPLE?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          2,0,3,1,0,1<cr_lf>
```

8.11.3 Unpair Device (ATUPLE)

UN PAIR DEVICE

Function: This command is used to delete the pairing information for an LE device.

Command Format: ATUPLE,<Addr>

Command Parameter(s):

- **Addr:** Device address and address type.

Example(s):

```
COMMAND:  ATUPLE,ECFE7E000001:0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

8.11.4 Clear Pair List (ATCPLE)

CLEAR PAIR LIST

Function: This command is used to delete the pairing information for all paired LE devices.

Command Format: ATCPLE

Example(s):

```
COMMAND:  ATCPLE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

8.11.5 Passkey Response (ATPKR)

PASSKEY RESPONSE

Function: Responds to a passkey request (PK_REQ) event. The passkey request event will be sent when a passkey input is needed for authentication during the pairing process.

Command Format: ATPKR,<Addr_Conn_Handle>,<Passkey>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Passkey:** 6 numeric characters

Example(s):

1. LE pairing is initiated using the ATPLE command to connection handle 0. Authentication is required by one side or the other and based on the module's IO_Capabilities (ATSPLE), PK_REQ is sent to the local module. The remote device will display a passkey. This event is then responded to with an ATPKR command with a passkey of 123456. Once pairing is completed the PAIRED event is sent:

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PK_REQ,ECFE7E000001:0<cr_lf>
COMMAND:  ATPKR,ECFE7E000001:0,123456<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,0,1<cr_lf>
```

8.11.6 Passkey Confirm (ATPKC)

PASSKEY CONFIRM

Function: Responds to a passkey display (PK_DIS) event that required confirmation.

Command Format: ATPKC,<Conn_Handle>,<Confirm>

Command Parameter(s):

- **Conn_Handle:** Connection handle.

- **Confirm:**
  0 = Passkey does not match.
  1 = Passkey match confirmed.

Example(s):

1. LE pairing is initiated using the ATPLE command to connection handle 0. Authentication is required by one side or the other and based on the module's IO_Capabilities (ATSPLE), PK_DIS is sent to the local module requiring confirmation. This passkey is then confirmed with the ATPKC command. Once pairing is completed the PAIRED event is sent:

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PK_DIS,0,ECFE7E000001:0,123456,1<cr_lf>
COMMAND:  ATPKC,0,1<cr>
EVENT:    <cr_lf>
          PAIRED,0,ECFE7E000001:0,1<cr_lf>
```

8.11.7 Fixed Passkey (ATSPK)

SET PASSKEY

Function: Sets a fixed passkey to be used for Low Energy pairing authentication. A fixed passkey can be used to support authentication on a device without a keypad or display, although it does not provide the same level of security that a randomly generated passkey would.

For a fixed passkey to be used, Authentication needs to be enabled in ATSPLE and the IO_Capabilities need to be set to something other than No Input or Output, or the Passkey Entry method of authentication will not be used and authentication will not occur (see the table in ATSPLE). In most cases the IO_Capabilities should be set to Display Only (0), which will cause the remote device to have to enter the fixed passkey. The IO_Capabilities can be set to Keyboard Only (2) if always pairing with a device with a fixed passkey set to Display Only (0).

Command Format: ATSPK,<Passkey>

Command Parameter(s):

- **Passkey:** 6 numeric characters. If set to 0, a fixed passkey will not be used. If a fixed passkey is set, in a case where a PK_REQ would have been sent, instead it will automatically be responded to using the fixed passkey. In a case where a PK_DIS would have been sent, the fixed passkey will be used internally instead.
  Default: 0 (Fixed passkey disabled)

Example(s):

1. Set a module to use a fixed passkey and set the IO Capabilities to Display Only, which will cause the remote device to have to enter the fixed passkey, but if the remote device's IO_Capabilities don't support a Keyboard than the link won't be authenticated.

```
COMMAND:  ATSPK,123456<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>

COMMAND:  ATSPLE,1,1,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

2. Set another module to automatically authenticate with the fixed passkey module in the previous example. The IO Capabilities are set to Numeric Keyboard Only, which will cause the module to automatically respond with the fixed passkey when requested during authentication.

```
COMMAND:  ATSPK,123456<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>

COMMAND:  ATSPLE,1,1,2<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET PASSKEY

Function: Gets the fixed passkey.

Command Format: ATSPK?

Response Format: <Passkey>

Example(s):

```
COMMAND:  ATSPK?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          123456<cr_lf>
```

8.12 White List

8.12.1 White List Device (ATSWL)

WHITE LIST

Function: This command is used to add a device to the White List. The White List allows an LE device to filter out only the devices it cares about and ignore all others. The White List works with the <White_List_Filter> parameters of ATSDSLE and ATSDILE, as well as with ATDMLE when a specific address is not used.

Command Format: ATSWL,<Addr>

Command Parameter(s):

- **Addr:** Device address and address type.

Example(s):

```
COMMAND:  ATSWL,ECFE7E000001:0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *Up to 8 devices can be added to the whitelist.*
- *This command cannot be used when the module is in the Advertising, Discovering or Connecting states or if Sync_White_List is enabled in ATSPLE.*

GET WHITE LIST

Function: This command is used to read the White List.

Command Format: ATSWL?

Response Format: <Count>,<Addrs>

Response Value(s):

- **Count:** Number of devices in the White List.
- **Addrs:** List of addresses in the White List, separated by '-' characters. 0 if Count is 0.

Example(s):

1. The module does not have any device in its White List.

```
COMMAND:  ATSWL?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,0<cr_lf>
```

2. The module has two devices in its White List.

```
COMMAND:  ATSWL?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          2,ECFE7E000001:0-ECFE7E000002:0<cr_lf>
```

8.12.2 Un White List Device (ATUWL)

UN WHITE LIST DEVICE

Function: This command is used to remove a device from the White List.

Command Format: ATUWL,<Addr>

Command Parameter(s):

- **Addr:** Device address and address type.

Example(s):

```
COMMAND:  ATUWL,ECFE7E000001:0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used when the module is in the Advertising, Discovering or Connecting states or if Sync_White_List is enabled in ATSPLE.*

8.12.3 Clear White List (ATCWL)

CLEAR WHITE LIST

Function: This command is used to remove all devices from the White List.

Command Format: ATCWL

Example(s):

```
COMMAND:   ATCWL<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

Note(s):

- *This command cannot be used when the module is in the Advertising, Discovering or Connecting states or if Sync_White_List is enabled in ATSPLE.*

8.13 GATT Client

The GATT client commands allow the module to use the Generic Attribute Profile (GATT) to discover and use the services on a remote device. GATT commands cannot be cancelled using the ATDC command.

8.13.1 Discover All Primary Services (ATGDPS)

GATT DISCOVER ALL PRIMARY SERVICES

Function: This command is used to discover all the primary services on a server.

Command Format: ATGDPS,<Conn_Handle>,<Value_Format>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Value_Format:** How the services will be formatted when returned by the GATT_DPS event.
  0 = Hex UUID
  5 = Service Acronym String – In this format if a 16-bit UUID of a BT5.0 defined service is discovered, the service acronym will be returned instead. All other UUIDs will still be returned as Hex UUIDs, except for the BRSP service.

Example(s):

1. ATGDPS is issued for connection handle 0 and 6 primary services are discovered: GAP, GATT, DIS (Device Info), BAS (Battery), DFU (Firmware Update) and BRSP. When all the primary services have been discovered a GATT_DONE event is triggered:

```
COMMAND:  ATGDPS,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,1,9,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,10,13,1801<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,14,24,180A<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,25,28,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,29,32,FE59<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,33,65535,DA2B84F1627948DEBDC0AFBEA0226079<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,0<cr_lf>
```

2. Same as example 1, but using the Service Acronym String format:

```
COMMAND:  ATGDPS,0,5<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,1,9,GAP<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,10,13,GATT<cr_lf>
```

```
EVENT:    <cr_lf>
          GATT_DPS,0,14,24,DIS<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,25,28,BAS<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,29,32,DFU<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,33,65535,BRSP<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,0<cr_lf>
```

8.13.2 Discover Primary Services by UUID (ATGDPSU)

GATT DISCOVER PRIMARY SERVICES BY UUID

Function: This command is used to discover a specific primary service on a server.

Command Format: ATGDPSU,<Conn_Handle>,<UUID>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **UUID:** UUID of the service to discover. [16-bit UUID = 4 chars, 128-bit UUID = 32 chars]

Example(s):

1. ATGDPSU is issued for connection handle 0 and the GAP Service. The service is found with a start handle of 1 and an end handle of 9.

```
COMMAND:  ATGDPSU,0,1800<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,1,9,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,0<cr_lf>
```

2. ATGDPSU is issued for connection handle 0 and the BRSP Service. The service is found with a start handle of 15 and an end handle of 65535, since it the last service in the device's GATT table.

```
COMMAND:  ATGDPSU,0,DA2B84F1627948DEBDC0AFBEA0226079<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,33,65535,DA2B84F1627948DEBDC0AFBEA0226079
          <cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,0<cr_lf>
```

8.13.3 Discover All Characteristics (ATGDC)

GATT DISCOVER ALL CHARACTERISTICS

Function: This command is used to discover all the characteristics on a server or all the characteristics within a specific attribute handle range.

Command Format: ATGDC,<Conn_Handle>,<Start_Att_Handle>,<End_Att_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Start_Att_Handle:** Attribute handle to start searching at, typically the Svc_Att_Handle of a specific service. Start_Att_Handle is included in the search. [1-65535, 1 if not specified]
- **End_Att_Handle:** Attribute handle to stop searching at, typically the Svc_End_Att_Handle of a specific service. End_Att_Handle is included in the search. Must be greater than or equal to Start_End_Handle. [1-65535, 65535 if not specified]

Example(s):

1. ATGDC is used to discover all characteristics between handles 1 and 9 (GAP Service), and 4 characteristics are found.

```
COMMAND:  ATGDC,0,1,9<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,3,2,2A00<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,5,2,2A01<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,7,2,2A04<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,9,2,2AA6<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,0<cr_lf>
```

Note(s):

- *If Start_Att_Handle is specified, End_Att_Handle must also be specified.*

8.13.4 Discover Characteristics by UUID (ATGDCU)

GATT DISCOVER CHARACTERISTICS BY UUID

Function: This command is used to discover specific characteristics on a server or specific characteristics within a specific attribute handle range.

Command Format: ATGDCU,<Conn_Handle>,<UUID>,<Start_Att_Handle>,<End_Att_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **UUID:** UUID of the characteristic to discover. [16-bit UUID = 4 chars, 128-bit UUID = 32 chars]
- **Start_Att_Handle:** Attribute handle to start searching at, typically the Svc_Att_Handle of a specific service. Start_Att_Handle is included in the search. [1-65535, 1 if not specified]
- **End_Att_Handle:** Attribute handle to stop searching at, typically the Svc_End_Att_Handle of a specific service. End_Att_Handle is included in the search. Must be greater than or equal to Start_End_Handle. [1-65535, 65535 if not specified]

Example(s):

1. ATGDCU is used to discover the Device Name Characteristic (UUID 2A00), and it is found at handle 3.

```
COMMAND:  ATGDCU,0,2A00,1,9<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,3,2,2A00<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,0<cr_lf>
```

Note(s):

- *If Start_Att_Handle is specified, End_Att_Handle must also be specified.*

8.13.5 Characteristic Descriptor Discovery (ATGDCD)

GATT DISCOVER CHARACTERISTIC DESCRIPTORS

Function: This command is used to discover all the characteristic descriptors within a characteristic definition. It will return all attributes within the specified handle range, so if the range is not correctly specified, attribute types other than descriptors will be returned.

If a range is not specified then all attributes will be returned, allowing the entire GATT table to be seen.

Command Format: ATGDCD,<Conn_Handle>,<Start_Att_Handle>,<End_Att_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Start_Att_Handle:** Attribute handle to start searching at, typically the Char_Value_Att_Handle+1 of a specific characteristic. Start_Att_Handle is included in the search. [1-65535, 1 if not specified]
- **End_Att_Handle:** Attribute handle to stop searching at, typically the ending handle of the characteristic. End_Att_Handle is included in the search. Must be greater than or equal to Start_End_Handle. [1-65535, 65535 if not specified]

Example(s):

2. ATGDCD is used to discover the characteristic descriptors of the BAS Battery Level characteristic on a remote device. First, ATGDPSU is used to find the BAS service, which has a start handle of 25 and an end handle of 28:

```
COMMAND:  ATGDPSU,0,180F<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,25,28,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,0<cr_lf>
```

Next, ATGDC is used to find the Battery Level characteristic:

```
COMMAND:  ATGDC,0,25,28<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,27,12,2A19<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,0<cr_lf>
```

Last, ATGDCD is used to find the Battery Level characteristic descriptors. The Start_Att_Handle is set to 28, which is the Battery Level characteristic value handle + 1 that was found by ATGDC. The End_Att_Handle is also set to 19, since this is the last handle in the BAS service (thus the last handle in the characteristic), which was found by ATGDPSU:

```
COMMAND:  ATGDCD,0,28,28<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

```
EVENT:    <cr_lf>
          GATT_DCD,0,28,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,3,0<cr_lf>
```

8.13.6 Characteristic Read (ATGR)

GATT READ

Function: This command is used to read specific characteristic values or descriptors from a server when the attribute handle of the characteristic value or descriptor is known.

This command can read values up to a total of MTU-1 bytes in length. The MTU (Maximum Transmission Unit) of a connection can be found using the ATCS? Command.

Command Format: ATGR,<Conn_Handle>,<Att_Handle>,<Value_Format>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Att_Handle:** Attribute handle of the characteristic value or descriptor to read.
- **Value_Format:** How the value will be formatted when returned by the GATT_VAL event.
  0 = Hex
  1 = 8-Bit Unsigned Decimal
  2 = 16-Bit Unsigned Decimal
  3 = 24-Bit Unsigned Decimal
  4 = 32-Bit Unsigned Decimal
  5 = String

Example(s):

1. Reading the remote BT5.0 Device Name, formatted in Hex by default.

```
COMMAND:  ATGR,0,3<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_VAL,0,3,0,0,16,426C7565526164696F7330303030303031<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,4,0<cr_lf>
```

8.13.7 Characteristic Read Multiple (ATGRM)

GATT READ MULTIPLE

Function: This command is used to read multiple characteristic values or descriptors from a server when the attribute handle of the characteristic values or descriptors is known. The data is returned as one blob of data since the module has no way to know the length of the multiple characteristic values being returned.

This command can read combined characteristic values up to a total of MTU-1 bytes in length. The MTU (Maximum Transmission Unit) of a connection can be found using the ATCS? Command.

Command Format: ATGRM,<Conn_Handle>,<Att_Handles>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Att_Handles:** List of attribute handles to read, separated by '-' characters.

Example(s):

1. Reading characteristics 44 and 46 from a remote device.

```
COMMAND:  ATGRM,0,44-46<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_VAL,0,0,3,0,19,01013F0202000001200000000015000600FD03<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,9,0<cr_lf>
```

Note(s):

- *The BR-LE5.0-S1 and PAN1780-AT only support multiple reads as a client. If you try to use ATGRM to read from another BR-LE5.0-S1 or PAN1780-AT, the command will fail.*

8.13.8 Characteristic Read Long (ATGRL)

GATT READ LONG

Function: This command is used to read specific characteristic values or descriptors from a server when the attribute handle of the characteristic value or descriptor is known.

This command allows reads greater than MTU-1 bytes in length. The MTU (Maximum Transmission Unit) of a connection can be found using the ATCS? Command.

Command Format: ATGRL,<Conn_Handle>,<Att_Handle>,<Value_Format>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Att_Handle:** Attribute handle of the characteristic value or descriptor to read.
- **Value_Format:** How the value will be formatted when returned by the GATT_VAL event.
  0 = Hex
  5 = String

Example(s):

1. ATGRL is used to read a value from handle 54. The value is 33 bytes long, so 2 GATT_LVAL's are returned followed by a GATT_DONE. The first GATT_VAL returns the first portion of the data from offset 0 and the second returns the final portion of the data from offset 22.

```
COMMAND:  ATGRL,0,54<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_LVAL,0,54,0,22,
          11223344556677889900112233445566778899001122<cr_lf>
EVENT:    <cr_lf>
          GATT_LVAL,0,54,22,11,33445566778899001122 33<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,6,0<cr_lf>
```

8.13.9 Characteristic Read by UUID (ATGRU)

GATT READ BY UUID

Function: This command is used to read specific characteristic values or descriptors from a server when the handle of the characteristic value or descriptor is not known. If Start_Att_Handle is specified, End_Att_Handle must also be specified.

This command can read values up to a total of MTU-1 bytes in length. The MTU (Maximum Transmission Unit) of a connection can be found using the ATCS? Command.

Command Format: ATGRU,<Conn_Handle>,<UUID>,<Value_Format>,
<Start_Att_Handle>,<End_Att_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **UUID:** UUID of the characteristic value or descriptor to read. [16-bit UUID = 4 chars, 128-bit UUID = 32 chars]
- **Value_Format:** How the value will be formatted when returned by the GATT_VAL event.
  0 = Hex
  1 = 8-Bit Unsigned Decimal
  2 = 16-Bit Unsigned Decimal
  3 = 24-Bit Unsigned Decimal
  4 = 32-Bit Unsigned Decimal
  5 = String
- **Start_Att_Handle:** Attribute handle to start searching at, typically the Svc_Att_Handle of a specific service. [1-65535]
- **End_Att_Handle:** Attribute handle to stop searching at, typically the Svc_End_Att_Handle of a specific service. End_Att_Handle is included in the search. Must be greater than or equal to Start_End_Handle. [1-65535]

Example(s):

1. Reading the remote BT5.0 Device Name, formatted in Hex by default.

```
COMMAND:  ATGRU,0,2A00<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_VAL,0,3,0,0,16,426C7565526164696F73303030303031<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,4,0<cr_lf>
```

2. Reading the remote BT5.0 Device Name, formatted as an ASCII string.

```
COMMAND:  ATGRU,0,2A00,5<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_VAL,0,3,0,5,16,BlueRadios000001<cr_lf>
```

```
EVENT:    <cr_lf>
          GATT_DONE,0,4,0<cr_lf>
```

8.13.10 Characteristic Write (ATGW)

GATT WRITE

Function: This command is used to write a specific characteristic value or descriptor from a server when the attribute handle of the characteristic value or descriptor is known. The server will return a response confirming the value was written, which will trigger a GATT_DONE event.

If more than the connection MTU-3 bytes are passed in the module will perform a prepared write and internally break the packet up into separate packets.

Command Format: ATGW,<Conn_Handle>,<Att_Handle>,<Value_Format>,<Value>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Att_Handle:** Attribute handle of the characteristic value or descriptor to write.
- **Value_Format:** Value format.
  0 = Hex
  1 = 8-Bit Unsigned Decimal
  2 = 16-Bit Unsigned Decimal
  3 = 24-Bit Unsigned Decimal
  4 = 32-Bit Unsigned Decimal
  5 = String
- **Value:** Value data.

Example(s):

1. Writing a 16-bit value of 1 using Value_Format 2.

```
COMMAND:  ATGW,0,99,2,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,5,0<cr_lf>
```

2. Writing a 16-bit value of 1 using Value_Format 0. This is equivalent to example 1.

```
COMMAND:  ATGW,0,99,0,0100<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,5,0<cr_lf>
```

3. Writing the string “HELLO” using Value_Format 5.

```
COMMAND:  ATGW,0,99,5,HELLO<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,5,0<cr_lf>
```

4. Writing the string “HELLO” using Value_Format 0. This is equivalent to example 3.

```
COMMAND:  ATGW,0,99,0,48454C4C4F<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,5,0<cr_lf>
```

5. Writing the string “THIS STRING IS LONGER THAN 20 BYTES” using Value_Format 5. Internally the write will be broken up into two packets, but only one GATT_DONE event will occur after both packets have been sent.

```
COMMAND:  ATGW,0,99,0,THIS STRING IS LONGER THAN 20 BYTES<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,5,0<cr_lf>
```

8.13.11 Characteristic Write No Response (ATGWN)

GATT WRITE NO RESPONSE

Function: This command is used to write a specific characteristic value or descriptor on a server when the attribute handle of the characteristic value or descriptor is known.

This command can write values up to a total of MTU-3 bytes in length. The MTU (Maximum Transmission Unit) of a connection can be found using the ATCS? Command.

Command Format: ATGWN,<Conn_Handle>,<Att_Handle>,<Value_Format>,<Value>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Att_Handle:** Attribute handle of the characteristic value or descriptor to write.
- **Value_Format:** Value format.
  0 = Hex
  1 = 8-Bit Unsigned Decimal
  2 = 16-Bit Unsigned Decimal
  3 = 24-Bit Unsigned Decimal
  4 = 32-Bit Unsigned Decimal
  5 = String
- **Value:** Value data.

Example(s):

1. Writing a 16-bit value of 1 using Value_Format 1.

```
COMMAND:  ATGWN,0,99,2,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

2. Writing a 16-bit value of 1 using Value_Format 0. This is equivalent to example 1.

```
COMMAND:  ATGWN,0,99,0,0100<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

3. Writing the string “HELLO” using Value_Format 5.

```
COMMAND:  ATGWN,0,99,5,HELLO<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

4. Writing the string “HELLO” using Value_Format 0. This is equivalent to example 3.

```
COMMAND:  ATGWN,0,99,0,48454C4C4F<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

8.13.12 Characteristic Write Prepared (ATGWP/ATGWPE)

GATT WRITE PREPARED

Function: This command can be used to manually perform a long write, allowing more bytes to be sent than with the ATGW command. It can also be used when multiple characteristic values must be written, in order, in a single operation. The attribute handle of the characteristic value(s) or descriptor(s) must be known. With each ATGWP the value written will be buffered on the remote device, but not actually written. Once the entire value has been sent using ATGWP, the ATGWPE command must be used to execute the write, at which point it will be written on the remote device. The server will return a response confirming the value was received after each ATGWP, which will trigger a GATT_DONE event, but the value will not actually be written on the remote device until an ATGWPE is executed.

This command can only write values up to ATT_MTU-5 bytes in length at a time. ATT_MTU is the Attribute Protocol Maximum Transmission Unit, which on Single Mode modules is 23, so the maximum write length per ATGWP is 18 bytes. Multiple ATGWP commands can be tied together to write a characteristic longer than 18 bytes by increasing the offset with each write. The maximum characteristic size set by the BT5.0 specification is 512 bytes, but the number of prepared bytes a device can receive will vary by device. For example, BlueRadios Single Mode modules can only receive a prepared write of up to 90 bytes (5 prepared writes of 18 bytes.)

Command Format: ATGWP,<Conn_Handle>,<Att_Handle>,<Value_Format>,<Value>,<Offset>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Att_Handle:** Attribute handle of the characteristic value or descriptor to write.
- **Value_Format:** Value format.
  0 = Hex
  1 = 8-Bit Unsigned Decimal
  2 = 16-Bit Unsigned Decimal
  3 = 24-Bit Unsigned Decimal
  4 = 32-Bit Unsigned Decimal
  5 = String
- **Value:** Value data.
- **Offset:** Value data offset.

Example(s):

1. Writing a 40-byte ASCII string “1234567890123456789012345678901234567890” using multiple prepared writes.

```
COMMAND:  ATGWP,0,99,5,123456789012345678,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,7,0<cr_lf>
COMMAND:  ATGWP,0,99,5,901234567890123456,18<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,7,0<cr_lf>
```

```
COMMAND:  ATGWP,0,99,5,7890,36<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,7,0<cr_lf>
COMMAND:  ATGWPE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,8,0<cr_lf>
```

GATT WRITE PREPARED EXECUTE

Function: This command is used to finalize a prepared write.

Command Format: ATGWPE,<Conn_Handle>,<Execute>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Execute:**
  0 = Cancel Prepared Write
  1 = 8-Bit Unsigned Decimal

Example(s): See ATGWP example above.

8.14 GATT Service Configuration

8.14.1 Device Information Service (ATSDIS)

SET DIS CONFIGURATION

Function: Configures the Device Information Service (DIS) on modules running Peripheral Observer firmware. DIS exposes manufacturer and/or vendor information about a device. All characteristics are optional though, so individual characteristics can be disabled if not needed. All devices should implement the Device Information Service, but if all the characteristics are disabled the service will be removed.

[Reset Required] Any changes to this service will require a reset to take effect.

DIS Characteristics:

- **Manufacturer Name String:** Represents the name of the manufacturer of the device.
- **Model Number String:** Represents the model number that is assigned by the vendor.
- **Serial Number String:** The serial number for a particular instance of the device.
- **Hardware Revision String:** The hardware revision for the hardware within the device.
- **Firmware Revision String:** The firmware revision for the firmware within the device.
- **Software Revision String:** The software revision for the software withing the device.
- **System ID:** A structure containing an Organizationally Unique Identifier (OUI) followed by a manufacturer-defined identifier and is unique for each individual instance of the product.
- **IEEE 11073-20601 Regulatory Certification Data List:** A structure representing regulatory and certification information for the product defined in IEEE 11073-20601.
- **PnP ID:** A structure containing a set of values that shall be used to create a device ID that is unique for this device. Included in the characteristic are a Vendor ID source field, a Vendor ID field, a Product ID field, and a Product Version field. These values are used to identify all devices of a given type/model/version using numbers.

The Device Information Service specification can be found here:
https://www.bluetooth.org/en-us/specification/adopted-specifications

Command Format: ATSDIS,<Characteristic_ID>,<Enable>,<Value>

Command Parameter(s):

- **Characteristic_ID:**
  - 0 = Manufacturer Name String
    Default: Enabled, “BlueRadios,Inc.”
  - 1 = Model Number String
    Default: Enabled, “BR-LE5.0-S1”
  - 2 = Serial Number String
    Default: Enabled, last six digits of IEEE address ex: “000001”
  - 3 = Hardware Revision String
    Default: Disabled, “Hardware Revision”
  - 4 = Firmware Revision String
    Default: Enabled, “5.0.1.0-S1”
  - 5 = Software Revision String
    Default: Disabled, “Software Revision”
  - 6 = System ID
    Default: Disabled, “0000000000000000”
  - 7 = IEEE 11073-20601 Regulatory Certification Data List

Default: Disabled, “FE006578706572696D656E74616C”
8 = PnP ID
Default: Enabled, “01850000000001”

- **Enable:**
  0 = Characteristic Disabled
  1 = Characteristic Enabled

- **Value:**
  The value characteristic is optional to allow enabling/disabling without changing the value.

  Characteristic_ID 0-5: 1-20-byte ASCII string
  Characteristic_ID 6: 8-byte ASCII Hex string (16 characters)
  Characteristic_ID 7: 1-20-byte ASCII Hex string (2-40 characters)
  Characteristic_ID 8: 7-byte ASCII Hex string (14 characters)

Example(s):

1. Sets the Manufacturer Name String characteristic.

```
COMMAND:  ATSDIS,0,1,NewManufacturerName<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

2. Sets the PnP ID characteristic.

```
COMMAND:  ATSDIS,8,1,01850000000001<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

3. Disables the PnP ID characteristic.

```
COMMAND:  ATSDIS,8,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *A reset is required for any changes to this command to take effect.*

GET DIS CONFIGURATION

Function: Gets the DIS service configuration.

Command Format: ATSDIS?,<Characteristic_ID>

Command Parameter(s):

- **Characteristic_ID:**
  - 0 = Manufacturer Name String
  - 1 = Model Number String
  - 2 = Serial Number String
  - 3 = Hardware Revision String
  - 4 = Firmware Revision String
  - 5 = Software Revision String
  - 6 = System ID
  - 7 = IEEE 11073-20601 Regulatory Certification Data List
  - 8 = PnP ID

Response Format: <Enable>,<Value>

Example(s):

```
COMMAND:  ATSDIS?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,BlueRadios,Inc.<cr_lf>
```

8.14.2 Battery Service (ATSBAS)

SET BAS CONFIGURATION

Function: Configures the Battery Service (BAS).

The Battery Service specification can be found here:
https://www.bluetooth.com/specifications/gatt

Command Format: ATSBAS,<Service_Enable>,<Mode>,<Update_Interval>,<Critical_Level>

Command Parameter(s):

- **Service_Enable: [Reset required for change to take effect]**
  0 = BAS Service Disabled
  1 = BAS Service Enabled

- **Mode:**
  0 = Manual Mode – Level is set manually by the user with ATSBASL. Use if not powering the module directly from a 3.0V coin cell and battery level is being monitored by an external controller.

  1 = Automatic Mode – Level is automatically calculated internally. The BAS level characteristic will be updated when read through GATT and updated at the specified update interval when connected and notifications are enabled. Use if a 3.0V coin cell is connected directly to VDD.

  If BAS is enabled in Automatic mode with Critical_Level set to a non-zero value, the BATT event will fire to alert the application that the battery has dropped below the critical level. It will continue to fire each time the battery percentage drops when below this level. To be notified anytime the battery level drops, Critical_Level can be set to 100.

- **Update_Interval:** 1-687194 s. Only applicable if Mode =1, 0 if Mode = 0.

- **Critical_Level:**
  0-100%. Only applicable if Mode =1. When the battery level is less than or equal to the Critical_Level the BATT event will be sent to notify the application of a low battery.

Example(s):

1. Enables the BAS service in manual mode.

```
COMMAND:  ATSBAS,1,0,0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

2. Enables the BAS service in automatic mode, set to update the level every hour (3600s).

```
COMMAND:  ATSBAS,1,1,3600<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

3. Enables the BAS service in automatic mode, set to update the level every hour (3600s) with a critical level of 10%.

```
COMMAND:  ATSBAS,1,1,3600,10<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *A reset is required for a change to the Service_Enable parameter to take effect.*

.

GET BAS CONFIGURATION

Function: Gets the BAS service parameters.

Command Format: ATSBAS?

Response Format: <Service_Enable>,<Mode>,<Update_Interval>,<Critical_Level>

Example(s):

```
COMMAND:  ATSBAS?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,0,0,0<cr_lf>
```

8.14.2.1 Battery Service Level (ATSBASL)

SET BAS LEVEL

Function: Sets the Battery Service battery level.

Command Format: ATSBASL,<Level>

Command Parameter(s):
- **Level:** 0-100%

Example(s):
1. Sets the battery level to 100%.

```
COMMAND:  ATSBASL,100<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):
- *The battery level is not stored in flash and will default to 100 after a reset.*

GET BAS LEVEL

Function: Gets the Battery Service battery level.

Command Format: ATSBASL?

Response Format: <Level>

Example(s):

```
COMMAND:  ATSBASL?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          100<cr_lf>
```

8.14.3 DFU Service (ATSDFU)

SET DFU CONFIGURATION

Function: Configures the Over the Air Firmware Update Service (DFU).

Command Format: ATSDFU,<Service_Enable>,<Active>

Command Parameter(s):

- **Service_Enable: [Reset required for change to take effect]**
  0 = DFU Service Disabled
  1 = DFU Service Enabled

- **Active:** Since Service_Enable requires a reset to take effect, the Active parameter allows OTA updates to be disabled while keeping the service in place
  0 = DFU Service Inactive – The DFU service is enabled, but a remote device will not be allowed to start a firmware update.
  1 = DFU Service Active – The DFU service is enabled and a paired device can start a firmware update.

Example(s):

1. Enables the DFU service in active mode.

```
COMMAND:  ATSDFU,1,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *A reset is required for a change to the Service_Enable parameter to take effect.*
  .

GET DFU CONFIGURATION

Function: Gets the DFU service parameters.

Command Format: ATSDFU?

Response Format: <Service_Enable>,<Active>

Example(s):

```
COMMAND:  ATSDFU?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,1<cr_lf>
```

8.14.4 BRSP Service (ATSBRSP)

The *n*Blue™ modules use a proprietary GATT profile developed by BlueRadios to stream data; it is not an official BT5.0 profile. BlueRadios serial port implementation simplifies the user experience, allowing users to stream data similar to the way the official BT Serial Port Profile (SPP) works on BR/EDR devices. It allows the *n*Blue™ modules to behave very similar to the BlueRadios ATMP modules.

BRSP throughput can now reach speeds of up to 50kBps between two BR-LE5.0-S1A and PAN1780-AT modules when the MTU is set to 247 bytes. When connecting a BR-LE5.0-S1A or PAN1780-AT to an older BR-LE4.0 module the MTU will be 23 and the maximum BRSP throughput will be 1.3kBps.

SET BRSP CONFIGURATION

Function: Sets the BRSP service parameters.

Command Format: ATSBRSP,<Service_Enable>,<Features_Flags>,<Security_Mode>,<TX_Mode>,<RX_Mode>

Command Parameter(s):

- **Service_Enable: [Reset required for change to take effect]**
  0 = BRSP Service Disabled
  1 = BRSP Service Enabled

- **Features_Flags:** Specifies what BRSP features are enabled.
  1 = Data Mode Supported
  2 = Remote Command Mode Supported

- **Security_Mode:**
  0 = No Security Required (Security Mode 1, Level 1)
  1 = Unauthenticated Pairing Required (Security Mode 1, Level 2)
  2 = Authenticated Pairing Required (Security Mode 1, Level 3)
  3 = LESC Authenticated Pairing Required (Security Mode 1, Level 4)

- **TX_Mode:**
  0 = Serial Port Mode – In BRSP mode, all data received on the UART will be sent over BRSP.
  1 = Command Mode – In BRSP mode, all data received on the UART will continue to be parsed locally as commands. Data can be sent over BRSP using the ATBRSPW command.

- **RX_Mode:**
  0 = Serial Port Mode – In BRSP mode, all data received over the air through BRSP will be sent directly through the UART.
  1 = Event Mode – In BRSP mode, all data received over the air through BRSP will be sent in BRSPD events.

Example(s):

1. Enables the BRSP service with support for Data Mode and Remote Command Mode.

```
COMMAND:  ATSBRSP,1,3,0,0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *To set Security_Mode to 2 or 3 the IO_Capabilities of ATSPLE must not be set to 3 (No Input No Output.)*
- *A reset is required for a change to the Service_Enable parameter to take effect.*

GET BRSP CONFIGURATION

Function: Gets the BRSP service parameters.

Command Format: ATSBRSP?

Response Format: <Service_Enable>,<Features_Mask>,<Security_Mode>,<TX_Mode>,<RX_Mode>

Example(s):

```
COMMAND:  ATSBRSP?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,3,0,0,0<cr_lf>
```

8.14.4.1 Serial Port Mode Configuration (ATSSP)

SET SERIAL PORT MODE CONFIGURATION

Function: Sets the escape character and no data timeout parameters

Command Format: ATSSP,<Escape_Char>,<No_Data_Timeout>

Command Parameter(s):

- **Escape_Char:** Integer value from 0 to 255. The escape character is used to put the radio into Command Mode if the module is in BRSP Data Mode or Remote Command Mode and ATSBRSP TX_Mode is set to Serial Port Mode. For example, if the escape character is changed to '-', then ---<cr> would put the radio into command mode instead of +++<cr>. The escape character is entered as the decimal value of a non-extended ASCII character. For example, '+' is entered as 43. **If set to 0, the module will not look for escape characters and all received data will be sent over the air.**
  Default: 43
- **No_Data_Timeout:** Integer value from 0 to 60000 [ms] (0= No Timeout). If in Data Mode and no data is sent or received for the set timeout the module will automatically disconnect.
  Default: 0

Example(s):

```
COMMAND:  ATSSP,43,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET SERIAL PROFILE

Function: Gets the escape character and no data timeout parameters.

Command Format: ATSSP?

Response Format: <Escape_Char>,<No_Data_Timeout>

Example(s):

```
COMMAND:  ATSSP?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          43,0<cr_lf>
```

8.14.4.2 Serial Port Mode Escape Sequence (+++)

COMMAND MODE

Function: This command is used to switch from Data Mode or Remote Command Mode back to Command Mode. In Command Mode all UART data is interpreted locally as AT commands.

Command Format: <Escape_Char><Escape_Char><Escape_Char>

Command Parameter(s):

- **Escape_Char:** The escape character set in the ATSSP command.

Example(s):

1. A connection is made to ECFE7E000001 using ATDMLE with a BRSP Mode set to Data Mode. After receiving the BRSP,0,1 message, the message “HELLO” is sent to the remote device. +++ is then used to switch to Command Mode. Once in Command Mode an ATDH is sent to disconnect from the remote device.

```
COMMAND:  ATDMLE,ECFE7E000001:0,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
EVENT:    <cr_lf>
          BRSP,0,1,0,0<cr_lf>
TXDATA:   HELLO
COMMAND:  +++<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
COMMAND:  ATDH,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCONNECT,0,0<cr_lf>
```

Note(s):

- *The three escape characters will not be passed through to the remote module.*
- *The escape character can be changed using the ATSSP command.*

8.14.4.3 Data Mode (ATMD)

DATA MODE

Function: Puts the module into Data Mode. In Data Mode all UART data is sent to the remote device using the BRSP service for LE connections. Any data sent from the remote device will be sent out of the module's UART.

Command Format: ATMD,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle to enter data mode on. If only one connection is active the connection handle does not have to be specified.

Example(s):

1. A connection is made to ECFE7E000001 using ATDMLE. ATMD is then used to put the module into Data Mode. Once in Data Mode, the BRSP,0,1 event is fired, after this the message "HELLO" is sent to the remote device.

```
COMMAND:   ATDMLE,ECFE7E000001:0,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
COMMAND:   ATMD<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           BRSP,0,1,0,0<cr_lf>
TXDATA:    HELLO
```

Note(s):

- *The module must be connected to use this command.*

8.14.4.4 Remote Command Mode (ATMRC)

REMOTE COMMAND MODE

Function: Attempts to put the module into Remote Command Mode. In Remote Command Mode all UART data is sent to the remote device using the BRSP service for LE connections. On the remote device the data is then interpreted as an AT command, executed on the remote device, and then responded to over the air.

Command Format: ATMRC,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle to enter remote command mode on. If only one connection is active the connection handle does not have to be specified.

Example(s):

- A connection is made to ECFE7E000001 using ATDMLE. ATMRC is then used to put the module into Remote Command Mode. Once in Remote Command Mode, the BRSP,0,2 event is fired, after this an ATA is sent and the address from the remote device is returned.

```
COMMAND:   ATDMLE,ECFE7E000001:0,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
COMMAND:   ATMRC<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           BRSP,0,2,0,0<cr_lf>
COMMAND:   ATA<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
           <cr_lf>
           ECFE7E000001:0<cr_lf>
```

Note(s):

- *The module must be connected to use this command.*
- ***Remote command mode is enabled by default, allowing AT commands to be sent to the device over the air. If you do not want remote command mode enabled, it can be disabled using ATSBRSP.***

8.14.4.5 BRSP Write (ATBRSPW)

BRSP WRITE

Function: Used to send BRSP data when BRSP TX_Mode is set to Command Mode.

Command Format: ATBRSPW,<Conn_Handle>,<Data>,<Carriage_Return>

Command Parameter(s):

- **Conn_Handle:** Connection handle to send the data to.
- **Data:** ASCII data, up to 244 characters.
- **Carriage_Return:** Used to append a carriage return to the end of the data string – allowing commands to be sent with ATBRSPW when in remote command mode.
  0 = No carriage return appended.
  1 = Carriage return appended to data.

Example(s):

1. A connection is made to ECFE7E000001 using ATDMLE. ATMD is then used to put the module into Data Mode. Once in Data Mode, the BRSP,0,1 event is fired, after this the message "HELLO" is sent to the remote device.

```
COMMAND:   ATDMLE,ECFE7E000001:0,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
COMMAND:   ATMD<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           BRSP,0,1,1,1<cr_lf>
TXDATA:    ATBRSPW,0,HELLO
```

Note(s):

- *The module must be connected and in BRSP data mode to use this command.*

8.15 RF Testing

8.15.1 LE Direct Test Mode (ATDTM)

LE DIRECT TEST MODE

Function: This command is used to put the module into the LE Direct Test Mode (DTM) described in Volume 6 Part F of the BT5.0 Specification. It is used for testing the module over the air using a BT tester such as the R&S CBT. The module will use the UART TX and RX lines for DTM and will immediately reset into DTM upon receiving the ATDTM command so no response will be sent to the command. To get the module back into AT command mode it will need to be hardware reset via the RESET pin.

Command Format: ATDTM

Example(s):

```
COMMAND:  ATDTM<cr>
RESPONSE: None
```

Note(s):

- *The UART baud rate will be 19200 in Direct Test Mode.*
- *The output power will be 8dBm in Direct Test Mode.*
- *DTM mode cannot be used over USB. If the ATDTM command is sent over USB the module will go into DTM mode, but then the USB port will not be available.*

8.15.2 Test Mode (ATTEST)

TEST MODE

Function: This command is used to put the module into Test Mode so that ATTXT and ATRXT can be used. Wait until the TEST MODE string is received before sending any of the test commands.

Command Format: ATTEST

Example(s):

```
COMMAND:  ATTEST<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          TEST MODE<cr_lf>
```

Note(s):

- *The module must be reset to exit test mode. This can be done using either the ATRST command or using the hardware reset pin.*
- *Test mode cannot be used over USB. If the ATDTM command is sent over USB the module will go into DTM mode, but then the USB port will not be available.*

8.15.3 Transmitter Test (ATTXT)

TRANSMITTER TEST

Function: This command is used to start an RF transmitter test. This can be used to satisfy in part radio regulation requirements as specific in standards such as ARIB STD-T66.

Command Format: ATTXT,<Mode>,<Channel_Hop_Rate>,<TX_Power>,<Data_Rate>

Command Parameter(s):

- **Mode:**
  0 = Continuous Unmodulated Transmit – Continuously transmits using an unmodulated carrier on the specified channel.

  1 = Continuous Modulated Transmit – Continuously transmits using a modulated carrier on the specified channel.

  2 = Channel Hopping Unmodulated Transmit –Transmits an unmodulated carrier, transmitting a packet on each channel for the specified hop rate, continuously cycling linearly through all channels.

  3 = Channel Hopping Modulated Transmit –Transmits a modulated carrier, transmitting a packet on each channel for the specified hop rate, continuously cycling linearly through all channels.

- **Channel_Hop_Rate:**
  Mode = 0,1 – Channel
  RF channel [0-39]. 17 (2440 MHz) if not specified.
  *Frequency = 2402+(Channel*2) MHz*

  Mode = 2,3 – Hop Rate
  Hop rate in microseconds. 625 if not specified.

- **TX_Power:** -40,-20,-16,-12,-8,-4,0,2,3,4,5,6,7,8 dBm. 8 if not specified.

- **Date_Rate:**
  0 = 1 Mbit/s
  1 = 2 Mbit/s

Example(s):

```
COMMAND:  ATTXT,0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *The module must be in test mode (ATTEST) before executing an ATTXT. The test can be canceled with an ATDC command.*

- *The power level can be changed during the test using ATSCPL (set Conn_Handle to 0).*

8.15.4 Receiver Test (ATRXT)

RECEIVER TEST

Function: This command is used to start an RF receiver test. The module can receive from a module in the transmitter test mode 0 on the same channel. The module will sample the RSSI at a specified rate for a specified duration and report back the values, with an average being reported at the end of the test. This can be used to satisfy in part radio regulation requirements as specific in standards such as ARIB STD-T66.

Command Format: ATRXT,<Mode>,<Channel_Hop_Rate>,<RSSI_Sampling_Rate>,
<RSSI_Sample_Count>,<Print_RSSI_Samples>,<Data_Rate>

Command Parameter(s):

- **Mode:**
  0 = Continuous Receive – Continuously receives on the specified channel.

  1= Continuous Receive with RSSI – Continuously receives on the specified channel. RSSI values will be reported at the specified RSSI_Sampling_Rate and the average will be reported at the end of the test.

  2 = Channel Hopping Receive – Receives on a different channel at the specified hop rate, continuously cycling linearly through all channels.

- **Channel_Hop_Rate:**
  Mode = 0 – Channel
  RF channel [0-39]. 17 (2440 MHz) if not specified.
  *Frequency = 2402+(Channel*2) MHz*

  Mode = 2 – Hop Rate
  Hop rate in microseconds. 625 if not specified.

- **RSSI_Sampling_Rate:** How often the RSSI will be sampled. [ms] 100 if not specified.

- **RSSI_Sample_Count:** How many RSSI samples to take. A value of 0 will run until cancelled. 10 if not specified.

- **Print_RSSI_Samples:**
  0 = Do not print any RSSI samples, only print the average RSSI at the end of the test.
  1 = Print all samples as they are taken in addition to the RSSI at the end of the test.

- **Data_Rate:**
  0 = 1 Mbit/s
  1 = 2 Mbit/s

Example(s):

1. A receiver test is started on channel 0, it will take a sample every second, for three seconds and print each sample. After three samples are taken the average is printed and a DONE event is triggered.

```
COMMAND:  ATRXT,1,0,1000,3,1,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
```

```
-42<cr_lf>
<cr_lf>
-43<cr_lf>
<cr_lf>
-44<cr_lf>
<cr_lf>
AVG,-43.00<cr_lf>
<cr_lf>
```

Note(s):

- *The module must be in test mode (ATTEST) before executing an ATRXT. The test can be canceled with an ATDC command.*

8.15.5 RF Observation (ATRFO)

RF OBSERVATION

Function: This command is used to enable RF observation. When enabled, the TX_PIO will toggle when the module is transmitting and the RX_PIO will go toggle when the module is receiving.

Command Format: ATRFO,<Enable>,<TX_PIO>,<RX_PIO>,<Active_High>

Command Parameter(s):

- **Enable:**
  0 = RF Observation Disabled
  1 = RF Observation Enabled

- **TX_PIO:** PIO to toggle when the radio is transmitting. 19 if not specified.

- **RX_PIO:** PIO to toggle when the radio is receiving. 20 if not specified.

- **Active_High:**
  0 = PIOs will toggle from high to low when transmitting/receiving.
  1 = PIOs will toggle from low to high when transmitting/receiving.

Example(s):

```
COMMAND:  ATRFO,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used when the module is Discovering, Connecting, or Connected.*

8.16 Bootloader

8.16.1 Run Bootloader (ATBOOT)

RUN BOOTLOADER

Function: This commands the module into the bootloader, after the specified delay.

Command Format: ATBOOT,<Delay>

Command Parameter(s):

- **Delay:** Integer value from 0 to 65535 [ms]. Default: 0 [ms]

Example(s):

```
COMMAND:  ATBOOT<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          NBOOT,0,3,BR-LE5.0-S1<cr_lf>
```

8.16.2 Bootloader Configuration (ATSBOOT)

SET BOOTLOADER

Function: Configures the module's bootloader transport settings. The UART transport mode cannot be disabled.

Command Format: ATSBOOT,<USB_Enable>,<UART_Baud_Rate>,<UART_Flow_Control>

Command Parameter(s):

- **USB_Enable**
  0 = Disabled
  1 = Enabled

- **UART_Baud_Rate:** 3-10

Baud rate (bps)	Value
9600	3
19200	4
38400	5
57600	6
115200	7
230400	8
460800	9
921600	10

- **UART_Flow_Control:**
  0 = Flow Control Off
  1 = Flow Control On

Example(s):

```
COMMAND:  ATSBOOT,1,7,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET BOOTLOADER

Function: Gets the bootloader configuration.

Command Format: ATSBOOT?

Response Format: <USB_Enable>,<UART_Baud_Rate>,<UART_Flow_Control>

Example(s):

```
COMMAND:  ATSBOOT?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,7,1<cr_lf>
```

8.17 Custom Profile

The custom profile allows the user to add custom services to the GATT server profile. Custom profile requires a host controller connected via the UART or USB interfaces to responded to Custom Profile Read and Custom Profile Write events from a client. The custom profile is stored in RAM, and thus will need to be setup again if the module is reset or power cycled. The last value set via the Custom Profile Update Value command is stored in the GATT profile, so when the host responds to a CP_READ from a client with the only the Custom Profile Read Response command, the stored value will be sent to the client.

Indication and notifications are handled automatically by the *n*Blue™ module. When the attribute value is updated from the host using the Custom Profile Update Value command, notifications and indications will automatically be sent to any client that has enabled them. For an example on how to enable notifications, see the Enabling Notifications on an LE Device Using GATT Commands example.

The module has a limited amount of RAM for the GATT profile database. A maximum of 15 services can be added. Custom Profile is limited to a maximum of 63 characteristics and 63 descriptors. However, this is dependent on the settings chosen for the custom service. For instance, the larger the characteristic/descriptor maximum attribute length, the lower the number of services/characteristics/descriptors that can be added. Unfortunately, Nordic does not make the RAM utilization calculation publicly available, so the only way to determine the maximum number of attributes that can be added is by adding them using the Add Custom Profile Service command and see if it is accepted or returns an ERROR,11 (Memory Allocation Error).

8.17.1 Custom Profile Settings (ATSCP)

SET CUSTOM PROFILE SETTINGS

Function: Configures the custom profile settings.

Command Format: ATSCP,<No_Response_Timeout>

Command Parameter(s):

- **No_Response_Timeout:** Integer value from 1-65535 [ms]. The timeout after no response received from the host to CP_READ and CP_WRITE events.
  Default: 3000

Example(s):

```
COMMAND:  ATSCP,3000<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET CUSTOM PROFILE SETTINGS

Function: Gets the custom profile settings.

Command Format: ATSCP?

Response Format:<No_Response_Timeout>

Example(s):

```
COMMAND:  ATSCP?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          3000<cr_lf>
```

8.17.2 Custom Profile Service (ATSCPS)

ADD CUSTOM PROFILE SERVICE

Function: Adds a custom profile service.

Service Structure

UUID (uint8[16])
128-bit base UUID, do not include hyphens. Example: A3040000E8DF4E40A8EC5EB89C80E29D

Characteristics Count (uint8)
The number of characteristics included.

Characteristic Structures

UUID (uint8[16])

Settings (uint8 bit field)

Reserved	Reserved	Reserved	Indicate	Notify	Write	Write No Response	Read
0	0	0	1	1	1	1	1

Read
Reading the value permitted.

Write No Response
Writing the value with Write Command permitted.

Write
Writing the value with Write Request permitted.

Notify
Notification of the value permitted.

Indicate
Indications of the value permitted.

Read Permission (uint8)

Security Access	Value	Description
SEC_NO_ACCESS	0x00	Not possible to access
SEC_OPEN	0x01	Access open
SEC_JUST_WORKS	0x02	Access possible with 'Just Works' security at least
SEC_MITM	0x03	Access possible with 'MITM' security at least

Write Permission (uint8)

Security Access	Value	Description
SEC_NO_ACCESS	0x00	Not possible to access
SEC_OPEN	0x01	Access open
SEC_JUST_WORKS	0x02	Access possible with 'Just Works' security at least
SEC_MITM	0x03	Access possible with 'MITM' security at least

CCCD Permissions

Write Permission (uint8)

Security Access	Value	Description
SEC_NO_ACCESS	0x00	Not possible to access
SEC_OPEN	0x01	Access open
SEC_JUST_WORKS	0x02	Access possible with 'Just Works' security at least
SEC_MITM	0x03	Access possible with 'MITM' security at least

Maximum Attribute Value Length (uint16 - little-endian format)
Maximum value of 512 (0x0002)

Characteristic User Description Length (uint8)
Maximum value of 20 (0x14)

Characteristic User Description (uint8 [20])
Formatted as ASCII Hex Data. Unused bytes can be omitted.

Example: “Test Char 01” => 54657374204368617203031

Descriptor Count (uint8)
The number of descriptors included.

Descriptor Structures

UUID (uint16)
16-bit UUID. Example: 2907

Read Permission (uint8)

Security Access	Value	Description
SEC_NO_ACCESS	0x00	Not possible to access
SEC_OPEN	0x01	Access open
SEC_JUST_WORKS	0x02	Access possible with 'Just Works' security at least
SEC_MITM	0x03	Access possible with 'MITM' security at least

Write Permission (uint8)

Security Access	Value	Description
SEC_NO_ACCESS	0x00	Not possible to access
SEC_OPEN	0x01	Access open
SEC_JUST_WORKS	0x02	Access possible with 'Just Works' security at least
SEC_MITM	0x03	Access possible with 'MITM' security at least

Maximum Attribute Value Length (uint16 - little-endian format)
Maximum value of 512 (0x0002)

Command Format: ATSCPS,<Config_Data>

Command Parameter(s):

- **Config_Data:** Composed of the configuration structure formatted as ASCII Hex Data. The length of this parameter cannot exceed 1048 or 524 bytes.

Example(s):

```
COMMAND:  ATSCPS,B3040000E8DF4E40A8EC5EB89C80E29D02B3040001E8DF4E40A8EC5EB8
          9C80E29D1F0103011400064368617220310 22AA7010104002AA800010400B304
          0002E8DF4E40A8EC5EB89C80E29D1501030114000643686172203201 2AA70101
          0400<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Config_Data		
Service Settings	**Hex Value**	**Value**
Service UUID	B3040000E8DF4E40A8EC5EB89C80E29D	
Characteristics Count	02	2
Characteristic 1 Settings	**Hex Value**	**Value**
Characteristic UUID	B3040001E8DF4E40A8EC5EB89C80E29D	
Characteristic Settings	1F	Read, Write No Response, Write, Notify, Indicate
Characteristic Read Permission	01	SEC_OPEN
Characteristic Write Permission	03	SEC_MITM
CCCD Write Permission	01	SEC_OPEN
Characteristic Max Attribute Value Length	1400	20
Characteristic User Description Length	06	6
Characteristic User Description	436861722031	Char 1
Descriptor Count	02	2
Descriptor 1 UUID	2AA7	
Descriptor 1 Read Permission	01	SEC_OPEN
Descriptor 1 Write Permission	01	SEC_OPEN
Descriptor 1 Max Attribute Value Length	0400	4

Descriptor 2 UUID	2AA8	
Descriptor 2 Read Permission	00	SEC_NO_ACCESS
Descriptor 2 Write Permission	01	SEC_OPEN
Descriptor 2 Max Attribute Value Length	0400	4
Characteristic 2 Settings	**Hex Value**	**Value**
Characteristic UUID	B3040002E8DF4E40A8EC5EB89C80E29D	
Characteristic Settings	15	Read, Write, Indicate
Characteristic Read Permission	01	SEC_OPEN
Characteristic Write Permission	03	SEC_MITM
CCCD Write Permission	01	SEC_OPEN
Characteristic Max Attribute Value Length	1400	20
Characteristic User Description Length	06	6
Characteristic User Description	436861722032	Char 2
Descriptor Count	01	1
Descriptor 1 UUID	2AA7	
Descriptor 1 Read Permission	01	SEC_OPEN
Descriptor 1 Write Permission	01	SEC_OPEN
Descriptor 1 Max Attribute Value Length	0400	4

8.17.3 Custom Profile Update Value (ATCPUV)

SET CUSTOM PROFILE UPDATE VALUE

Function: This command is used to update the characteristic values and descriptors in the custom profile service.

Read Longs are handled internally by the module, so send all data in one ATCPUV command.

Command Format: ATCPUV,<Svc_Index>,<Char_Index>,<Desc_Index>,<Value_Format>,<Value>

Command Parameter(s):

- **Svc_Index:** The service index as an Integer value from 0 - 14.
- **Char_Index:** The characteristic index as an Integer value from 0 - 62.
- **Desc_Index:** The descriptor index as an Integer value from 0 - 62, or 63. When set to a value of 63, this index will be ignored, thus setting the specified characteristic index value.
- **Value_Format:** Value format. See ATGW for more examples of formatting.
  0 = Hex
  1 = 8-Bit Unsigned Decimal
  2 = 16-Bit Unsigned Decimal
  3 = 24-Bit Unsigned Decimal
  4 = 32-Bit Unsigned Decimal
  5 = String
- **Value:** Value data.

Example(s):

```
EVENT:     CP_READ,0,0,1,63<cr>
COMMAND:   ATCPUV,0,1,63,0,B706B05BEF2EFB0BBE64F32E7BFD8ACA92A673BE<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
COMMAND:   ATCPRR,0,0,1,63,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

8.17.4 Custom Profile Read Response (ATCPRR)

SET CUSTOM PROFILE READ RESPONSE

Function: This command is used to respond to CP_READ events. If this does not follow a Custom Profile Update Value (ATCPUV) command, the value stored in the stack will be sent. If the stored value has not been previously set; the value sent will be blank.

Command Format: ATCPRR,<Conn_Handle ><Svc_Index>,<Char_Index>,<Desc_Index>,<Result>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Svc_Index:** The service index as an Integer value from 0 - 14.
- **Char_Index:** The characteristic index as an Integer value from 0 - 62.
- **Desc_Index:** The descriptor index as an Integer value from 0 - 62, or 63. When set to a value of 63, this index will be ignored, thus setting the specified characteristic index value.
- **Result:** Value format.
  0 = Accepted
  1 = Rejected

Example(s):

```
EVENT:     CP_READ,0,0,4,1<cr>
COMMAND:   ATCPUV,0,4,1,5,HELLO<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
COMMAND:   ATCPRR,0,0,4,1,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

8.17.5 Custom Profile Write Response (ATCPWR)

SET CUSTOM PROFILE WRITE RESPONSE

Function: This command is used to respond to CP_WRITE events.

Command Format: ATCPWR,<Conn_Handle ><Svc_Index>,<Char_Index>,<Desc_Index>,<Result>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Svc_Index:** The service index as an Integer value from 0-14.
- **Char_Index:** The characteristic index as an Integer value from 0-62.
- **Desc_Index:** The descriptor index as an Integer value from 0-62, or 63. When set to a value of 63, this index will be ignored, thus setting the specified characteristic index value.
- **Result:** Value format.
  0 = Accepted
  1 = Rejected

Example(s):

```
EVENT:    CP_WRITE,4,0,0,63,18,FEE7CBF3D8288CDB58F61A5193589BEAB236<cr>
COMMAND:  ATCPWR,4,0,0,63,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

9 Extended Examples

The examples in this section will demonstrate using various commands together to accomplish common scenarios and use cases, as opposed to the command specific examples shown in the command definitions. New examples will be continually added to this section.

All examples start from a factory reset module.

9.1 Enabling Notifications on an LE Device Using GATT Commands

1. Connect to the LE device using ATDMLE with BRSP_Mode set to 0, to keep the module in command mode upon connecting. In this example we will connect to another S1 module.

```
COMMAND:  ATDMLE,ECFE7E000001:0,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E000001:0<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
```

2. Discover the service you are looking, in this example we will look for the battery service (BAS, UUID 0x180F). The service is found at attribute handles 25 – 28.

```
COMMAND:  ATGDPSU,0,180F<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,25,28,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,0<cr_lf>
```

3. Discover the characteristic you are looking for within the service, in this example we will look for the battery level (UUID = 0x2A19) within the battery service handle range of 25-28. The battery level characteristic is found at handle 27, with Read/Notify properties.

```
COMMAND:  ATGDCU,0,2A19,25,28<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,27,12,2A19<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,0<cr_lf>
```

4. Discover the client characteristic configuration descriptor (UUID = 0x2902), which allows the client to enable notifications from the characteristic. ATGDCD is used to discover all characteristic descriptors for the battery level characteristic, the Start_Att_Handle is set to 28, which is the battery level characteristic value handle + 1 that was found by ATGDC. The End_Att_Handle is also set to 28, since this is the last handle in the BAS service (thus the last handle in the characteristic), which was found by ATGDPSU.

```
COMMAND:  ATGDCD,0,28,28<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

```
EVENT:    <cr_lf>
          GATT_DCD,0,28,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,3,0<cr_lf>
```

5. Enable notifications by writing the value 0x0001 to the characteristic configuration descriptor.

```
COMMAND:  ATGW,0,28,2,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,5,0<cr_lf>
```

6. Now that notifications have been enabled, they will be received in GATT_VAL events. If connected to an S2 module, we can manually set the battery level to trigger a notification. Here we set the battery level to 99% and receive a notification on the master of the updated value (GATT_VAL data is formatted in hex).

(On Slave Module)

```
COMMAND:  ATSBASL,99<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

(On Master Module)

```
EVENT:    <cr_lf>
          GATT_VAL,0,27,1,0,1,63<cr_lf>
```

7. Optionally, if we pair with the remote device, it will retain that our device enabled notifications on the battery level client characteristic configuration descriptor the next time we connect and we won't need to enable notifications again.

9.2 Discovering and Connecting with Extended Advertising

1. On the advertising module ATSDSLE to configure the module to advertise using extended connectable advertisements. For this example, we will also use long range mode for the primary PHY and 2M for the secondary PHY. We will use the default extended advertising data, so we will not modify ATSDSDLE,2.

(On Peripheral Module)

```
COMMAND:  ATSDSLE,0,4,7,4,2,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

2. On the discovering module use ATDILE,4 to scan for extended advertising packets on the LR PHY. An EXT_DISCOVERY event will occur when an extended ad packet is received from the advertising device. The 4 and 2 highlighted below indicate the device is advertising with a primary PHY of LR and a secondary PHY of 2M.

(On Central Module)

```
COMMAND:  ATDILE,4<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          EXT_DISCOVERY,7,ECFE7E1CBAE1:0,-41,4,2,0,3362,6,020106-020A08-
          051208001000-1107796022A0BEAFC0BDDE487962F1842BDA-05FF85000005-
          1109BlueRadios1CBAE1<cr_lf>
EVENT:    <cr_lf>
          DONE,1,1<cr_lf>
```

3. On the discovering module use ATDMLE to connect with the primary PHY set to LR.

(On Central Module)

```
COMMAND:  ATDMLE,ECFE7E1CBAE1:0,0,4<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,2,0,ECFE7E1CBAE1:0<cr_lf>
EVENT:    <cr_lf>
          MTU,0,247<cr_lf>
```

4. The modules will now be connected on the 2M PHY, indicated by the highlighted 2's below in the ATCS? response from the peripheral module. When connecting using extended advertising the connection will take place on the secondary PHY.

(On Peripheral Module)

```
COMMAND:  ATCS?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          4,1,1,ECFE7E1CBAE0:0,16,0,400,247,2,2,8<cr_lf>
EVENT:    <cr_lf>
          DONE,2,0<cr_lf>
```

5. If we do an ATCS? on the central module the RX and TX PHYs will show 0 for Unknown. This is because internally the secondary PHY that the connection was made on is not available. This should be fixed in a future release.

(On Central Module)

```
COMMAND:  ATCS?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          4,2,1,ECFE7E1CBAE1:0,16,0,400,247,0,0,8<cr_lf>
EVENT:    <cr_lf>
          DONE,2,0<cr_lf>
```

6. Once connected the PHY can be changed using the ATSCPHY command. On the central module we will request to update the PHY to LR for both RX and TX. Once the PHY has been changed a PHYU event will occur on each module.

(On Central Module)

```
COMMAND:  ATSCPHY,0,4,4<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PHYU,0,4,4<cr_lf>
```

(On Peripheral Module)

```
EVENT:    <cr_lf>
          PHYU,0,4,4<cr_lf>
```

iBeacon Example

AT.s modules can be easily configured to advertise as iBeacons.

9.2.1 Advertising Data Format

iBeacon advertising packets consist of manufacturer specific data defined by Apple with the following fields:

- **Header:** Fixed bytes identifying the ad data structure as Manufacturer Specific Data for an Apple iBeacon [0x1AFF4C000215]
- **Proximity UUID:** A UUID used to identify a set of beacons in a certain location or region. Can be created using a UUID generator, such as http://www.uuidgenerator.net/. [16 bytes, big endian]
- **Major ID:** A value used to identify a group of related beacons. [2 bytes, big endian]
- **Minor ID:** A value used to identify a specific beacon within its Major group. [2 bytes, big endian]
- **Measured Power:** A value containing the average RSSI measured at 1m from the beacon. Set to default of -59 (0x5C) if unknown. [1 byte]

9.2.2 Relevant Commands

ATSDSDLE should be used as follows to set the fields of the beacon ad data:

ATSDSDLE,0,<Header><Proximity_UUID><Major_ID><Minor_ID><Measured_Power>

Beacons should operate in the Broadcaster role, so **ATSDSLE** should be used as follows to configure the device to advertise neither connectable nor scannable by setting the Ad_Type to 3:

ATSDSLE,0,3,7

(WARNING: After executing this command the module will not be connectable.)

If the beacon needs to be connectable or scannable, ATSDSDLE,1 needs to be used to change the default scan response data. By default, AT.s modules contain a BlueRadios manufacturer specific data structure in the scan response data, but this needs to be changed or it will interfere with iOS detecting the device as an iBeacon. This can be changed to the device name or transmit power level for example:

ATSDSDLE,1,080969426561636F6E (name = iBeacon)

or

ATSDSDLE,1,010A00 (transmit power = 0dB)

9.2.3 House Example

In this example we will create a set of beacons for a house. First generate a unique Proximity UUID to group all the beacons in the house: F5DA8F6E-CDEA-4B93-84F6-992103233A65

Then assign Major ID's based on each level: 0=Basement, 1=Main Floor, 2=Second Floor.

Next assign Minor ID's to each room in the house. Basement: 0=Media Room, 1=Utility Room. Main Floor: 0=Garage,1=Kitchen,2=Family Room. Second Floor: 0 = Master Bedroom, 1=Guest Bedroom, 2=Office.

A beacon for the Family Room would have a Major ID = 1 and Minor ID = 2.

To configure an AT.s module to act as a beacon for the Family Room the following commands would be sent:

```
COMMAND:  ATSDSDLE,0,1AFF4C000215F5DA8F6ECDEA4B9384F6992103233A6500010002C5<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>

COMMAND:  ATSDSLE,0,3,7<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

The ATSDSDLE Data can be broken down into:

- **Header:** 1AFF4C000215
- **Proximity UUID:** F5DA8F6ECDEA4B9384F6992103233A65
- **Major ID:** 0001
- **Minor ID:** 0002
- **Measured Power:** C5

10 Command Set Summary Table

10.1 Command Status Responses

Response	Description
OK	
OK	Command Accepted
Error	
ERROR	Command Not Accepted
No Response	
	Invalid Command

10.2 Events

Event	Description
General	
RESET	Reset
NBOOT	Bootloader
DONE	Background Task Complete
Discovery	
DISCOVERY	Non-Extended Discovery
EXT_DISCOVERY	Extended Discovery
Connection	
CONNECT	Connect
DISCONNECT	Disconnect
SCCPS	Set Current Connection Parameter Status
CPU	Connection Parameter Update
MTU	Maximum Transmission Unit Update
PHYU	PHY Update
RSSI	RSSI
Pairing	
PAIR_REQ	Pairing Request
PAIRED	Pairing Successful
PAIR_FAIL	Pairing Failed
PK_REQ	Passkey Request
PK_DIS	Passkey Display
RESOLVED	Private Resolvable Address Resolved
GATT Client	
GATT_DONE	GATT Done
GATT_DPS	Discovered Primary Service
GATT_DC	Discovered Characteristic
GATT_DCD	Discovered Characteristic Descriptor
GATT_VAL	Characteristic/Descriptor Value
GATT_LVAL	Long Characteristic/Descriptor Value

Gatt Server	
BATT	Battery Event
BRSP	BRSP Status
BRSPD	BRSP Data
CP_READ	Custom Profile Read
CP_WRITE	Custom Profile Write

10.3 Commands

Command	Description	Factory Default	Stored?
General			
AT	Attention	-	-
ATHELP	Help	-	-
ATSRM/ATSRM?	Set/Get Response Mode	0,0	X
Reset			
ATRST	Reset	-	-
ATFRST	Factory Reset	-	-
Config Control			
ATSCL/ATSCL?	Set/Get Configuration Lock	0	X
ATSFC/ATSFC?	Set/Get Flash Storage Configuration	1,1	X
ATFC	Flash Config (Manual)	-	-
ATSCFG/ATSCFG?	Set/Get Configuration	-	X
ATCFG?	Configuration Dump	-	-
Sleep			
ATZ	Enable Sleep Mode	-	-
ATSZ/ATSZ?	Set/Get Sleep Configuration	0,0,0,0	X
Module Information			
ATMT?	Get Module Type	-	-
ATST?	Get Stack Type	-	-
ATV?	Get Version	-	-
ATA?	Get Address	-	-
ATSN/ATSN?	Set/Get Name	BlueRadios######,0	X
ATSAPP/ATSAPP?	Set/Get Appearance	0	X
ATSAT/ATSAT?	Set/Get Address Type	0,0,900	X
Module State			
ATSLE?	Get State LE		
ATSDBLE/ATSDBLE?	Set/Get Default Behavior LE	1,0	X
ATDC	Cancel	-	-
Hardware Config			
ATSDPL/ATSDPL?	Set/Get Default Power Level	8,8	X
ATSCPL/ATSCPL?	Set/Get Current Power Level	-	-
ATSUART/ATSUART?	Set/Get UART Settings	7,0,0,1	X
ATSUSB/ATSUSB?	Set/Get USB Settings	1,0	X
ATSNFC/ATSNFC?	Set/Get NFC Settings	0	X
ATSPIO/ATSPIO?	Set/Get PIO Configuration	0/1/2,0,0	X
ATPIOL/ATPIOL?	Set/Get PIO Level	-	-

ATSLED/ATSLED?	Set/Get LED	Led 0 = 100,65535,0 Led 1 = 10,2000,0 Led 2 = 100,65535,0 Led 3 = 100,20,0	X
ATADC?	Get ADC		
ATBL?	Get Battery Level		
ATT?	Get Temperature		
ATCT	Calibrate Temperature Sensor		X
Advertising			
ATDSLE	Dial as Slave LE	-	-
ATSDSLE/ATSDSLE?	Set/Get ATDSLE Parameters	0,0,7,1,1,0	X
ATSDSTLE/ATSDSTLE?	Set/Get ATDSLE Timing Parameters	0,160,2048	X
ATSDSDLE/ATSDSDLE?	Set/Get ATDSLE Advertising Data	Advertising Data = 0201FF020AFF0512FFFFFFFF1107FFFFFFFFFFFFFFFFFFFFFFFFFFFF Scan Response Data = 05FFFFFFFFF1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF Ext Advertising Data = 0201FF020AFF0512FFFFFFFF1107FFFFFFFFFFFFFFFFFFFFFFFFFF05FFFFFFFFF1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	X
Discovery			
ATDILE	LE Discovery	-	-
ATSDILE/ATSDILE?	Set/Get ATDILE Parameters	0,0,1,10,0,7	X
ATSDITLE/ATSDITLE?	Set/Get ATDILE Timing Parameters	10,16,16	X
ATSDIF/ATSDIF?	Set/Get Discovery Formatting	65535,1,1,0,0	X
Connection			
ATDMLE	Dial as Master LE	-	-
ATDMLLE	Dial as Master Last LE	-	-
ATLCALE?	Get Last Connected Address LE	FFFFFFFFFFFF	X
ATSDMLE/ATSDMLE?	Set/Get ATDMLE Config Parameters	7,1	X
ATSDMTLE/ATSDMTLE?	Set/Get ATDMLE Timing Parameters	0,16,16	X
ATSDCP/ATSDCP?	Set/Get Default Connection Params	8,16,0,400,0	X
ATSCCP/ATSCCP?	Set/Get Current Connection Params	-	-
ATSDMTU	Set/Get Default MTU	247	X
ATSSPHY/ATSSPHY?	Set/Get Supported PHYS	7,7	X
ATSCPHY/ATSCPHY?	Set/Get Current PHY	-	-
ATCS?	Get Connection Status	-	-
ATRSSI?	Get RSSI Value	-	-
ATCRSSI	Get Continuous RSSI Value		
ATDH	Disconnect	-	-

Pairing			
ATPLE/ATPLE?	Pair Device/Get Paired Device List LE	0,0	X
ATSPLE/ATSPLE?	Set/Get Pairing Parameters LE	2,0,3,1,0,1	X
ATUPLE	Un Pair Device LE	-	X
ATCPLE	Clear Pair List LE	-	X
ATPKR	Passkey Response	-	-
ATPKC	Passkey Confirm	-	-
ATSPK/ATSPK?	Set/Get Fixed Passkey	0	X
White List			
ATSWL/ATSWL?	Set/Get White List	0,0	X
ATUWL	Un White List Device	-	X
ATCWL	Clear White List	-	X
GATT Client			
ATGDPS/ATGDPSU	GATT Discover Primary Services	-	-
ATGDC/ATGDCU	GATT Discover Characteristics	-	-
ATGDCD	GATT Discover Char Descriptors	-	-
ATGR/ATGRU	GATT Read	-	-
ATGRM	GATT Read Multiple	-	-
ATGRL	GATT Read Long	-	-
ATGW/ATGWN	GATT Write	-	-
ATGWP/ATGWPE	GATT Write Prepared	-	-
GATT Service Configuration			
ATSDIS/ATSDIS?	Set/Get DIS Configuration	See ATSDIS	X
ATSBAS/ATSBAS?	Set/Get BAS Configuration	1,0,0,0	X
ATSBASL/ATSBASL?	Set/Get BAS Level	100	-
ATSDFU/ATSDFU?	Set/Get DFU Configuration	1,1	
ATSBRSP/ATSBRSP?	Set/Get BRSP Configuration	1,3,0,0,0	X
BRSP Service Commands			
ATSSP/ATSSP?	Set/Get Serial Configuration	43,0	X
+++	Escape to Command Mode	-	-
ATMD	Data Mode	-	-
ATMRC	Remote Command Mode	-	-
ATBRSPW	BRSP Write	-	-
RF Testing			
ATDTM	LE Direct Test Mode	-	-
ATTEST	Test Mode	-	-
ATTXT	Transmitter Test	-	-
ATRXT	Receiver Test	-	-
ATRFO	RF Observation	-	-
Bootloader Commands			
ATBOOT	Run Bootloader	-	-
ATSBOOT/ATSBOOT?	Set/Get Bootloader Configuration	1,7,1	X

Custom Profile			
ATSCP/ATSCP?	Set Custom Profile Settings/Get Custom Profile Settings	3000	-
ATSCPS	Add Custom Profile Service	-	-
ATCPUV	Custom Profile Update Value	-	-
ATCPRR	Custom Profile Read Response	-	-
ATCPWR	Custom Profile Write Response	-	-

Appendix A: Acronyms/Abbreviations

AD – Advertisement
API – Application Protocol Interface
AT – Attention
ASCII – American Standard Code for Information Interchange
BLE – Bluetooth Low Energy
BOB – Breakout Board
BR – BlueRadios
BR/EDR – Basic Rate/Enhanced Data Rate (classic Bluetooth)
BRSP – BlueRadios Serial Port
BT – Bluetooth
CB – Classic BR/EDR Bluetooth
COD – Class of Device
COM/COMM – Communications
CR – Carriage Return
CTS – Clear to Send
DC – Direct Current
EDR – Enhanced Data Rate
GAP – Generic Access Profile
GATT – Generic Attribute Profile
GFSK – Gaussian Frequency-Shift Keying
GND – Ground
HCI – Host Controller Interface
IP – Internet Protocol
ISM – Industrial, Scientific, and Medical
LED – Light Emitting Diode
LESC – Low Energy Secure Connections
LF – Line Feed
MCU – Microcontroller Unit
MISO – Master In Slave Out
MOSI – Master Out Slave In
ms – Milliseconds
NC – No Connect
PC – Personal Computer
PCB – Printed Circuit Board
PHY – Physical Layer
PIN – Personal Identification Number
RF – Radio Frequency
PIO – Programmable Input/Output
RSSI – Received Signal Strength Indication
RST – Reset
RTS – Ready To Send
RX – Receive
SBB – Serial Breakout Board
SPI – Serial Protocol Interface
SPP – Serial Port Profile
TDMA – Time Division Multiple Access
TX – Transmit
UART – Universal Asynchronous Receiver/Transmitter
µs – Microseconds
USB – Universal Serial Bus
UUID – Universal Unique Identifier
V – Volts
VDD – DC Power