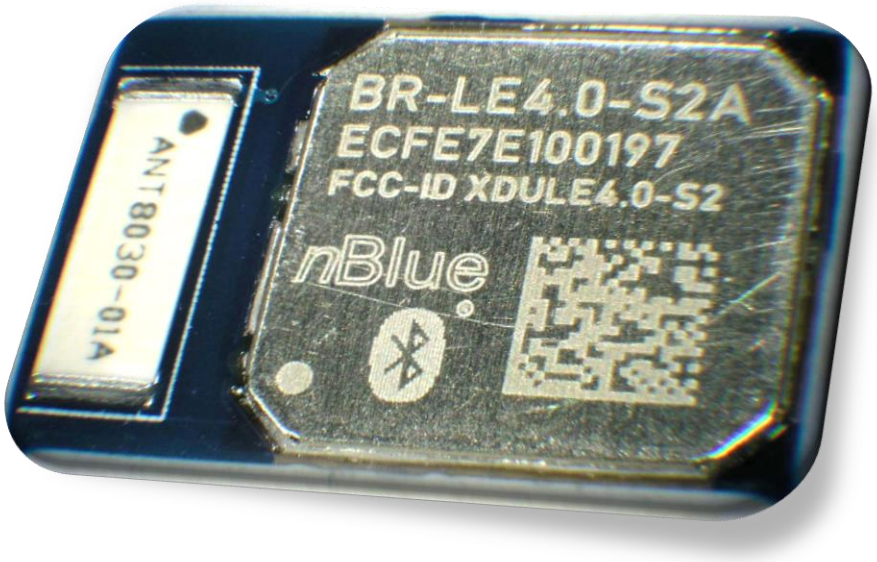
*n*Blue™ *Bluetooth*® 4.0
AT.s Command Set
v3.5



BR-LE4.0-S2A Single Mode Low Energy Module
(Actual Size Not Shown)

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

Table of Contents

Revision History

Rev #	Date	Description
2.0	11/17/2011	**Initial Document** ▪ AT commands have been separated from the User's Guide into this Command Set document, which now includes commands for dual mode modules as well as single mode modules. **New Features:** ▪ Classic Bluetooth (BR/EDR) commands for Dual Mode modules. ▪ LE pairing and whitelist are now supported. ▪ Updated BRSP service with remote command mode support. Over the air firmware updates supported on single mode modules with external flash. ▪ Device discoveries now come back in real time instead of after completion and up to 20 devices can be discovered. ▪ Full control over advertising/discovery/connection parameters and timing. ▪ All connection intervals are now supported and can be updated in connection. ▪ Improved Event system (previously Alerts). **New Commands:** ▪ ATSLP to put module into sleep mode as an alternative to PIO_3. ▪ ATST to return stack type. ▪ ATCS? to get connection status details. ▪ ATSCH to support multiple connections. ▪ ATFC/ATSFC for manual control of configuration flashing. ▪ ATSRM to configure response mode. ▪ ATSDIF for discovery data formatting. ▪ ATSSP to control serial profile data mode settings. ▪ ATMRC to support remote command mode through BRSP. ▪ ATPKR to support Passkey Entry for pairing. ▪ ATCFG? for configuration dump. ▪ ATSDBLE to configure the module's default behavior. ▪ ATSBRSP to allow configuration of the BRSP service. ▪ ATDSDLE to support connectable direct advertising mode. ▪ ATSDSLE/ATSDSTLE to allow advertising configuration. ▪ ATSDILE/ATSDITLE to allow discovery configuration. ▪ ATSDMTLE to allow connection initiation timing configuration. ▪ ATSCCP to allow connection parameter updates while connected. ▪ ATPLE/ATSPLE/ATUPLE/ATCPLE/ATSPK to support LE pairing. ▪ ATSWL/ATUWL/ATCWL to support LE whitelist. ▪ Classic Bluetooth (CB) commands to support BR/EDR in Dual Mode modules. **Changes:** ▪ All LE commands with a matching CB command now have an LE appended to the end, such as ATDM, which is now ATDMLE. ▪ Many commands now require a <Conn_Handle> parameter, which was added to support multiple connections. ▪ ATSN no longer requires a reset to take effect. ▪ ATSS command was removed. ▪ ATSCL now uses its own password instead of the PIN. ▪ ATSDM replaced by ATSDBLE, <Default_Unconnected_Comm_Mode> now part of ATSRM command. ▪ 230400 and 460800 Baud disabled for Single Mode modules due to instability. ▪ <Store> parameter removed from ATSPIO. ▪ ATRFT command split into ATTXT and ATRXT. ▪ ATSESC removed, escape character now part of ATSSP. ▪ ATSTP removed, discovery timeout now part of ATSDITLE, connect timeout part of ATSDMTLE, advertising timeout part of ATSDSTLE, no data timeout part of ATSSP. ▪ ATS changed to ATSLE and response split out into individual states. ▪ ATDI parameters have been moved to ATSDILE. ▪ ATDM is now ATDMLE and can accept optional <BRSP_Mode> and <Address_Type> parameters. ▪ ATDL has been changed to ATDMLLE. ▪ ATSCP is now ATSDCP. ▪ ATSCOD no longer applies to LE, only to CB. ▪ ATSAA has been removed as its functionality is provided by the whitelist. **Single Mode Bug Fixes:** ▪ Sleep mode now works on master role devices.

		▪ BRSP data mode is now much more stable.
2.1	2/22/12	**New Features:** ▪ GATT client commands and events for single mode devices. ▪ Configurable sleep mode for single mode devices. ▪ Data bridging for dual mode devices. **New Commands:** ▪ ATSZ command replaces ATSLP. ▪ ATSZ command to configure sleep mode. ▪ ATGDPS, ATGDPSU for GATT Primary Service Discovery. ▪ ATGDC, ATGDCU for GATT Characteristic Discovery. ▪ ATGDCD for GATT Characteristic Descriptor Discovery. ▪ ATGR, ATGRU for GATT Read. ▪ ATGW, ATGWN for GATT Write. ▪ ATB to enable data bridging between two different connections. ▪ ATUCR to handle CB numeric comparison authentication. **Changes:** ▪ The BRSP service now uses 128-bit UUIDs, so it will not be backwards compatible with previous versions. ▪ Added BRSP characteristic handles to Info characteristic to speed up BRSP initialization. ▪ BRSP can no longer be enabled/disabled if paired or connected. ▪ ATDC can now cancel specific commands, in addition to cancelling all. ▪ Added ATRRN, ATRRS, ATCS, ATTXT, ATRXT to DONE Event, added General Command Type and renumbered CB and LE commands. ▪ ATCS will now print DONE when complete. ▪ Pairing events now just print the address instead of a Conn_Handle, since CB pairing can take place without being connected. ▪ Pairing commands can now accept an address in place of a Conn_Handle. ▪ SCCP event changed to SCCPS for ATSCCP status, and status values updated so success = 0. ▪ CPU event added for actual connection parameter update. ▪ Bridge parameter added to ATSDB/ATSDBLE commands. ▪ 0 value added to ATSDILE Max_Devices, allowing the module to print advertising data updates. ▪ Removed discover only general discoverable devices option from ATSDI/ATSDILE as this option was invalid. ▪ GATT_DONE, GATT_DPS, GATT_DC, GATT_DCD and GATT_VAL events added to support GATT commands. ▪ ATSDIF, ATSDILE, ATSDITLE and ATSDMTLE will now respond ERROR,02 on slave single mode modules. ▪ If PIO2/5 Duty_Cycle in ATSLED is set to 0, PIO2/5 can be manually controlled using ATSPIO. ▪ ATSDILE Max_Devices had to be reduced to a range of 1-10 on single modules. **Single Mode Bug Fixes:** ▪ ATSPL command is now functional. ▪ ATTXT and ATRXT will now print DONE when complete. ▪ Advertising data now updates dynamically with ATSN, ATSPL, ATSDCP and ATSBRSP commands. ▪ ATDILE will no longer crash if a device is constantly updating its advertising data. ▪ ATRXT Test_Duration == 0 and Print_Samples are functional again.
2.2	3/2/2012	**Changes:** ▪ PIO_4 and PIO_6 are no longer software debounced for fastest response time when controlled by digital IO. Occasional bounces may be detected when using the buttons on the development boards. **Single Mode Bug Fixes:** ▪ Fixed a bug where random data could come back with the first response to a remote command.
3.0	4/9/12	**New Features:** ▪ Master and slave roles combined into one firmware version for single modes. ▪ Support for Battery Service (BAS) on single modes. ▪ Support for custom advertising and scan response data on single modes. ▪ Appearance can be set on single modes. ▪ Commands to read battery level, ADCs, and temperature on single modes. ▪ RF observation outputs on single modes. ▪ Auto slave connection parameter updates on single modes. **New Commands:** ▪ ATSBAS/ATSBAS? to configure the Battery Service.

		▪ ATSDSDLE/ATSDSDLE? to configure advertising and scan response data. ▪ ATSAPP/ATSAPP? to set the appearance. ▪ ATADC? to read ADCs. ▪ ATBL? to read the battery level. ▪ ATT? to read the temperature. ▪ ATCT to calibrate the internal temperature sensor. ▪ ATRFO to enable RF observation outputs. **Single Mode Changes:** ▪ See the Important Notes section for details on changes made to accommodate the combined master/slave architecture. ▪ A device connected in the slave role will now dynamically widen its Rx window when a previous connection event was missed. This improves connection stability by accounting for additional clock drift that may have occurred since the last successful connection event. ▪ The min default connection parameter is now defaulted to 8, and the max to 16. ▪ Scannable advertising is now allowed in connection. ▪ Discovery is now allowed in connection. ▪ Increased length of name in ATSN to 20. ▪ ATSCCP will now return ERROR,03 if the requested connection parameters are equal to the current connection parameters. ▪ Slave_Auto_Update optional parameter added to ATSDCP. ▪ "-S" and "-M" removed from RESET Event and ATMT, ATV responses. ▪ "-S2" added to ATV response. ▪ PIO_4 now only does an ATDMLE when in the idle state. ▪ Value_Format option added to ATGRU command. ▪ BlueRadios Manufacturer Specific data added to default scan response data in front of the name. **Single Mode Bug Fixes:** ▪ ATSDBLE? now responds with the bridge parameter on single modes. ▪ ATSCL <Password> can no longer be set to an empty string. ▪ Fixed a bug where CPU event wasn't printing on the remote side. ▪ When an ATSCCP is done from a slave, the SCCP event now comes back when the update request is accepted instead of with the CPU event. ▪ A factory reset will now reset the last connected address to FFFFFFFFFFFF. ▪ Fixed a bug where malformed ad data could crash a discovery.
3.1	6/1/12	**New Features:** ▪ First official dual mode release. **Changes:** ▪ PIO2/5 can now be set to inputs if <Duty_Cycle> is set to 0 in ATSLED. ▪ PIO15/16 can now be controlled through ATSPIO, if <Flow_Control> is disabled in ATSUART. ▪ ATSCH and ATSCH? have been removed. Conn_Handle parameters have been added to ATMD and ATMRC to replace the ATSCH functionality. ▪ ATDH can now be called without a Conn_Handle to disconnect all active connections. ▪ Minimal response mode in ATSRM will now send events with requested data. **Single Mode Changes:** ▪ When advertising is enabled by calling ATDS, the first advertisement event will now occur within a few milliseconds, rather than waiting for 10ms. **Single Mode Bug Fixes:** ▪ Initiating pairing using ATPLE should no longer cause a DISCONNECT to occur when using connection intervals less than 12.5ms. ▪ ATDC while in the idle state will no longer trigger the default behavior. ▪ Fixed a bug where ATDMLE,WL wouldn't connect unless an ATDC was done first.
3.2	12/14/12	**New Features:** ▪ RESOLVED event added to alert the user that a public address has been resolved from a private resolvable address during LE pairing. 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. ▪ ATSSN can now take an optional UUID parameter allowing the UUID of the CB service to be set. **Dual Mode Changes:** ▪ ATSPL command arguments have been changed to <LE_TX_Power>,<CB_TX_Power>, so LE and CB output power can be set independently. Output power can now be set up to 10 dBm for LE and 12 dBm for CB and both now default to 10dBm.

		▪ Due to instability, 921600 has been disabled, 460800 will now be the maximum allowed baud rate. ▪ ATSDBLE factory default changed to 1,0,0 so modules default to LE discoverable. ▪ BlueRadios Manufacturer Specific data added to default scan response data in front of the name. **Dual Mode Bug Fixes:** ▪ Fixed a bug causing some modules not to function properly at 3V. ▪ Fixed a bug causing data loss and dropped connections when streaming data. ▪ Fixed a bug causing lower than expected throughput over LE connections. ▪ Fixed a bug causing some modules to sometimes get stuck in a FACTORY_RESET loop on power up. ▪ Fixed a bug causing low LE output power, LE range is now much improved. ▪ Fixed a bug causing modules to lock up if CTS was set high with flow control enabled. ▪ Fixed ATSRM? and ATSPK? return values. ▪ Fixed a bug where PIO_3 wouldn't always toggle the module in and out of sleep. ▪ Fixed a bug causing ATSCOD not to take effect.
3.3	2/8/13	**New Features:** ▪ The D2 can now act as a true Classic to Low Energy bridge. Previously a D2 could only bridge to another BRSP device, but it can now be used to bridge to any LE device. For example an Android phone with no LE support can connect to the D2, put it in remote command mode, connect the D2 out to an LE device and then use the GATT commands to communicate with the LE device. ▪ ATSMRC to enable/disable remote command mode over CB. ▪ ATTXTE transmitter test with extended options for CB. **Firmware Version Number:** ▪ BlueRadios is switching to a simplified firmware version numbering system from this release forward. Instead of this release being 4.0.9.3.3.0 it will instead be 3.3.0.0. We are removing the first 3 digits which indicated the internal libraries AT.s was built on and adding an extra digit to allow for tracking bug fix releases (if necessary). Release versions will still always have a final digit of 0. The ATV documentation has been updated to reflect these changes. **Dual Mode Changes:** ▪ Events now come back over the air when the module is in remote command mode. ▪ A D2 module can be put into remote command mode over a CB SPP connection by sending "ATMRC<cr>" immediately after connecting, prior to sending any other data. This allows non-BlueRadios devices to be able to put a D2 into remote command mode. ▪ 921600 has been re-enabled, but when streaming bi-directional data at the maximum rate over a CB connection at 921600 baud, data may be lost. ▪ A Requested Mode Not Supported (6) SPP_Status has been added to the SPP status event. ▪ Input pulse of 250µs or greater will trigger PIO 3,4,6 now, previously was 2ms. ▪ Built on Bluetopia+LE 4.0.1.4. **Dual Mode Bug Fixes:** ▪ Fixed a bug causing LE connections to drop when a CB device was connected as well. ▪ Fixed a bug with ATSBRSP security mode not being set properly. ▪ Fixed a bug where ATSBRSP couldn't be modified when advertising. ▪ Fixed a bug where the PK_DIS event would be sent when a fixed passkey was set. ▪ Fixed a bug where setting a name longer than 20 bytes would cause LE to stop working. ▪ Fixed a bug where a GATT_VAL with data longer than 20 bytes would lock up the module **Documentation Changes:** ▪ Added an extended examples section. ▪ Added IO Capability/Authentication mapping tables for ATSP, ATSPLE and better ATSPK documentation. ▪ Added LE Protocol stack info back into introduction section. ▪ Added abbreviations appendix back to end of doc. ▪ Added more description to LE Discovery and Connection Parameters sections. ▪ Updated ATV documentation with new versioning system. ▪ Various minor corrections.
3.3b	8/9/13	**Documentation Update Only** **Documentation Changes:** ▪ The factory default setting of any stored parameters will now be identified by a blue background. ▪ Added new Lowering Power Consumption section that details sleep mode and provides ways to lower the module's power consumption ▪ Moved sleep mode details from PIO_3 notes to Lowering Power Consumption ▪ Added default IO state section to Hardware Notes. ▪ Added important note about configuration not being stored until disconnecting to Important Notes ,

		Command Usage and ATMRC sections. ▪ Added Related Applications section. ▪ Removed Command Usage requirement that numerical data in responses will print with leading zeroes to keep the response length consistent as D2 does not behave this way and future S2/S3 builds will not either. ▪ The factory default setting of any stored parameters will be identified by a blue background. ▪ Cleaned up ATSPL documentation. ▪ Added BRSP maximum theoretical throughput of 1.3kB/s note to important notes and ATSBRSP sections. ▪ Added a note about connecting to RESOLVED addresses in important notes and ATDMLE. ▪ Added a secure connection example to the extended examples section. ▪ Various minor corrections.
3.3c	1/30/2014	**Documentation Update Only** **Documentation Changes:** ▪ Added iBeacon example
3.4	3/10/2014	**New Features:** ▪ Added Apple MFi support for D2 modules. Command documentation only available to Apple MFi licensees, contact BlueRadios for more information. ▪ Added an optional parameter to ATSN to allow changing the name over GATT. # characters can now be used in the name to automatically insert all or part of the device address into the name. ▪ Added an optional parameter Override_State to ATSLED to allow direct control of led pulsing separate from module state. ▪ Added optional parameters Sync_White_List and Disconnect_Unpaired to ATSPLE to allow white list syncing and auto disconnecting unpaired devices. ▪ Added Critical_Level parameter to ATSBAS to alert application of low battery. ▪ Added BATT event to handle critical level alerts and client notification registration when in manual mode. **New Commands:** ▪ ATHELP to get the address of the BlueRadios forum. **Changes:** ▪ ATSPLE default Mode changed from 1 (Prompt for all pairing requests with PAIR_REQ event) to 2 (Automatically accept all pairing requests) to match ATSP. ▪ ATSDIF default Name_Format changed to string format. ▪ Added Target Address, Appearance and Ad Interval ad types to ATSDIF Data_Structure_Mask. ▪ ATSDSTLE <Timeout> is now in units of seconds instead of ms and the maximum value is now 180. **Dual Mode Changes:** ▪ Added support for ATSBAS in manual mode. ▪ Increased whitelist size to 30. ▪ Built on Bluetopia+LE 4.0.1.7. ▪ D2 will no longer issue a slave security request when connected to by a paired device, it will wait for the master to initiate security similar to how the S2 behaves. This fixes improves compatibility with the S2 and iOS devices. **Dual Mode Bug Fixes:** ▪ Fixed a bug with PIO2/5 (led) behavior in sleep mode. Increased accuracy of PIO2/5 ATSLED timing. ▪ Fixed some remote command mode command response issues ▪ Fixed issues with ATSDB/ATSDBLE behavior. ▪ ERROR,12 now returned if a GATT command is issued before previous GATT command completed. ▪ Value_Format parameter now works with ATGRU. ▪ Fixed a bug where displaying paired devices with ATP? or ATPLE? could fail when paired to both Classic and LE devices. ▪ Fixed a bug causing all connections to be returned by ATCS? when a specific handle was specified. ▪ Fixed a bug causing ATRN to not return a DONE event when a name was returned. **Documentation Changes:** ▪ Added the following to important notes: Due to connection timing constraints, changes to single mode module's (S2/S3) configuration cannot be stored in flash while the module is connected. Changes will still take effect immediately, but they will not be permanently stored until the module has disconnected. If configuration changes are made or a new device is paired while connected and then power is lost before disconnecting, these changes will not have been saved. This does not apply to dual mode modules (D2), only single modes (S2/S3).

		▪ Added a Related Applications section to important notes. ▪ Added unbonded paired states to PAIRED event. ▪ Updated ATSPL command documentation to account for S3 maximum output power of 0dB vs 4dB on S2. Also updated receive sensitivity numbers.
3.5.0.0	4/24/2015	**New Features:** ▪ Peripheral Observer (PO) single mode firmware is now available. Using this firmware the module will not support the central role (master), so it will not be able to connect out to other devices. However, it can still perform a discovery, advertise and be connected to. Disabling this functionality frees up the memory to allow support for more services, starting with the Device Information Service, Heart Rate Service and Proximity Profile Services (Link Loss Service, Immediate Alert Service and TX Power Service). ▪ Device Information Service support in PO firmware. ▪ Heart Rate Service support in PO firmware. ▪ Proximity Profile Services (Link Loss Service, Immediate Alert Service and TX Power Service) support in PO firmware. ▪ Support for long characteristic write and reads (for characteristics longer than 20 bytes.) ▪ Support for prepared writes. ▪ Support for random addresses. ▪ Support for low duty cycle directed advertisements. ▪ Auto populating ad data structures. ▪ Paired_Filter for ATSDILE to enable only discovering paired devices. ▪ Service Acronym String format for ATGDPS. ▪ ATSPIO can configure PIO_8 to control the bypass line of the TI TPS62730 DC-DC Converter on single mode modules. **New Commands:** ▪ ATSDIS/ATSDIS? to support the Device Information Service. ▪ ATSHRS/ATSHRS? and ATHRM to support the Heart Rate Service. ▪ ATSPXP/ATSPXP? to support Proximity Profile Services. ▪ ATGRL to support long characteristic reads. ▪ ATGWP/ATGWPE to support long/prepared characteristic writes. ▪ ATSAT/ATSAT? to support random addresses. **New Events:** ▪ HRS to support Heart Rate Service. ▪ PXP to support Link Loss and Immediate Alert Services. ▪ GATT_LVAL to support long reads. **Changes:** ▪ ATSDSDLE default settings now use auto populating ad data structures. ▪ Duty_Cycle parameter added to ATDSDLE to support low duty cycle directed advertisements. ▪ ATCWL can now be used while connected. ▪ For commands that store in flash with optional parameters – if an optional parameter isn't specified then the current value will be used instead of the default value. ▪ connHandle is now optional on ATRSSI? – 0 will be used if not specified. **Single Mode Changes:** ▪ Built on TI BLE-Stack 1.4.0. ▪ All single mode applicable changes since 3.1. **Single Mode Bug Fixes:** ▪ Fixed a bug causing the Ad Data and ATSCL Password to not be stored when changed over the air. ▪ Various minor bug fixes.

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** *Bluetooth* Version 2.0 radios, but many changes have been made to improve the user experience. Users already familiar with the ATMP command set will want to read the new command set documentation carefully, paying attention to all of 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 *Bluetooth* 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 *Bluetooth* RF connection.

1.2 Background

Bluetooth 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. *Bluetooth* Version 4.0 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 *Bluetooth* (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 *Bluetooth* and are likely to become a standard feature in almost all new *Bluetooth* enabled cellular phones and computers (i.e., gateway devices).

The ***Blue*****Radios *n*Blue™** modules are *Bluetooth* Version 4.0 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.

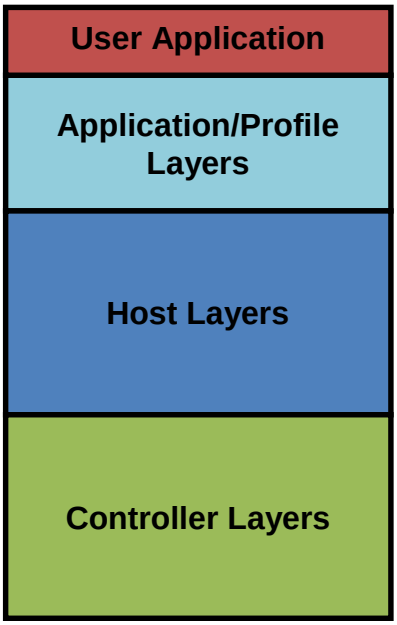
The ***n*Blue™** modules use a proprietary GATT profile developed by BlueRadios to stream data; it is not an official *Bluetooth* profile. BlueRadios serial port implementation simplifies the user experience, allowing users to stream data similar to the way the official *Bluetooth* Serial Port Profile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

1.3 LE Protocol Stack

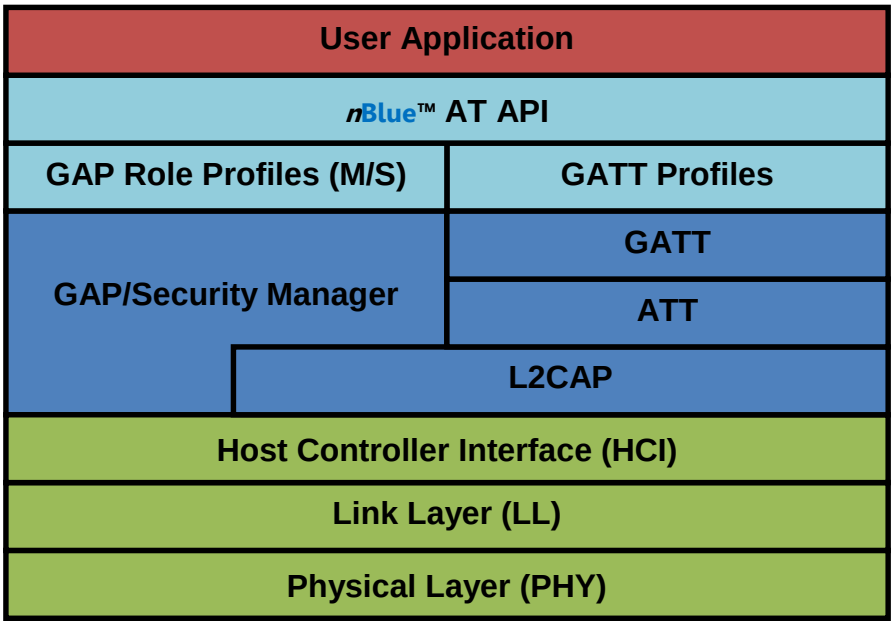
The *Bluetooth* low energy protocol stack is built similarly to the BR/EDR stack, with a controller layer at the bottom, followed by a host layer, and finally the profile and application layers. The GAP and GATT layers and above are the only layers that should concern the user. The GAP layer controls 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. The User Application can use the AT API through the UART (AT.s), using an external microcontroller, or by writing a custom application to run directly on the module (AT.e).

Simplified Stack



Detailed Stack



2 Important Notes – Please Read Prior To Continuing

2.1 Important Notes

- To provide the best firmware architecture, design, and future profile support there will not be 100% backwards compatibility between releases.
- 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. At the time of this document publishing the latest S2/S3 firmware version is 3.5.0.0 and the latest D2 firmware version is 3.4.1.0..
- At this time, Single Mode devices still only support one connection at a time, so <Conn_Handle> can always be set to 0, but support for multiple connections will be added in a future release.
- Modules running firmware older than version 2.1.0.0 are not compatible with modules with newer firmware when trying to send data using BRSP as the BRSP service has been re-architected.
- The single mode modules will add leading zeroes to fixed sized response fields to keep response lengths consistent, the dual mode will not. The single mode may be changed in the future to remove the leading zeroes in order to decrease response lengths.
- Remote command mode is enabled by default for both LE and CB, 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 for LE and ATSMRC for CB.
- Due to connection timing constraints, changes to single mode module's (S2/S3) configuration cannot be stored in flash while the module is connected. Changes will still take effect immediately, but they will not be permanently stored until the module has disconnected. If configuration changes are made or a new device is paired while connected and then power is lost before disconnecting, these changes will not have been saved. This does not apply to dual mode modules (D2), only single modes (S2/S3).
- BRSP will currently provide a maximum theoretical throughput of 1.3kB/s at the minimum LE connection interval of 7.5ms.
- 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.

2.2 Important Notes on Single Mode Combined Master/Slave Architecture

- Modules default to the slave role with their default behavior set to advertising, with an Ad_Type of connectable indirect advertisement (connectable + scannable).
- Modules will stay in the slave role until an ATDMLE/ATDMLLE command is executed. At this point they will automatically switch to the master role and initiate a connection. There is no event to indicate a role switch, it will happen transparently in the background.
- Modules will stay in the master role until an ATDSLE/ATDSDLE command is executed with a connectable Ad_Type. Since the default behavior is set to advertising, then by default once a master disconnects it will

switch to the slave role and start advertising again. So if the module will always be a master it is recommended to disable advertising as the default behavior, to prevent unnecessary role switching.

- To switch roles, the module must put itself into the idle state. So if the module is advertising/discovering and an ATDMLE is executed, these actions will automatically be cancelled before the connection attempt is initiated. Similarly to how a DONE,1,0 isn't sent when an advertising device is connected to, a DONE,0,0 won't be printed when an advertising device is issued an ATDMLE command and transitions to the initiating state. A DONE,1,1 will print if an ATDILE command is automatically cancelled.

2.3 Known Issues in This Version

2.3.1 Single Mode

- High data latency and/or data loss can occur when using connection intervals less than 12.5ms when the module's baud rate is set to 9600.
- Standard firmware does not support the following commands: ATSDIS, ATSDIS?, ATSHRS, ATSHRS? and ATHRM.
- Peripheral Observer (PO) firmware does not support the following commands: ATDMLE, ATDMLLE, ATSDMTLE and ATSDMTLE?

2.3.2 Dual Mode

- ATRXT does not support reading the RSSI values or printing them out. It also does not support timing out and must be cancelled with an ATDC.
- Legacy Android applications cannot connect to the module if the ATSP IO_Capabilities are set to No Input or Output. To guarantee that the module can connect to any Android application, set the IO_Capabilities in ATSP to Display Only or Display With Yes/No Buttons.
- If an ATTXT command is executed then cancelled and then executed again, the output power will be approximately 1dBm lower the second time and every time after. We recommend doing an ATRST between each ATTXT to guarantee accurate output power measurements.
- When streaming bi-directional data at the maximum rate over a CB connection at 921600 baud, data may be lost.
- The module needs to be reset after using ATSPL in order for the new power levels to take effect.

2.3.3 Single Mode / Dual Mode

- When connecting out as a master using ATDMLE, the connection can occasionally be dropped immediately after a CONNECT event with a DISCONNECT,05 (connection failed to be established) event. If the modules were paired a PAIR_FAIL with a reason of 7 (unknown) will be sent prior to the DISCONNECT on the D2. If this problem is encountered the best solution is to immediately retry connecting.

2.4 D2 Apple MFi Support

Apple requires Bluetooth devices connecting to iOS devices over classic Bluetooth (not LE) using Bluetooth profiles that are not supported by iOS (such as Serial Port Profile) to be authenticated as licensed MFi (Made for iPhone/iPod/iPad) accessories.

Support for MFi authentication has been added to the D2, but documentation can only be provided to verified Apple MFi licensees, contact BlueRadios for more information.

For more information on the MFi program, see: https://developer.apple.com/programs/mfi/

2.5 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™ *Bluetooth* 4.0 modules. Updates can be performed through the module's UART interface on all modules and Over the Air (OTA) through a BLE connection on Single Mode modules (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™ *Bluetooth* 4.0 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™ *Bluetooth* 4.0 modules over BRSP in both data and remote command modes. It is available to download for free from the Apple App Store.

2.6 Related Documents

- *n*Blue Module User's Guide
- *n*Blue BR-EVAL-4.0-X2A 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.

3.1 Electrical Specifications Summary

- The modules operate at a supply voltage (VDD) of 2.0-3.6V for single modes and 2.4-3.6V for dual modes.
- VDD ripple should not exceed 100mV.
- Maximum voltage level on any pin should not exceed 3.6V. **The I/O is not 5V tolerant.**
- Applying VDD to a PIO set to an output may permanently damage the module.
- All inputs are pulled low by an internal 20 kΩ resistor except for RX.

3.2 Power-up and Reset

There are no strict requirements for power up timing. The module is ready to receive commands once the boot string is sent out of the UART, approximately 165ms after power on the single mode and 880ms on the dual mode. To reset the module, the RESET line must be pulsed low for at least 1µS.

3.3 Default IO States

All IO on the module except for PIO_2/5,TX and RTS default to inputs. All inputs are pulled low internally except for RX, since it is normally driven by another UART. If the module is not connected to another UART, RX should be pulled high to keep it from floating.

3.4 PIO Functions

3.4.1 PIO_2/5 Status Indicators Outputs

PIO_2 and PIO_5 are defaulted to outputs used to indicate the current status of the module. Both are capable of driving up to 20mA, so they can be connected to LEDs. 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. Both can be configured using the ATSLED/ATSPIO commands.

3.4.2 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 at least 20ns on the S2 and 250µs on the D2. 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 when the module is sleeping and high when it is awake.

3.4.3 PIO_4 - Multipurpose Input

PIO_4 has multiple purposes. First, it can be used to factory reset the module by setting it to VDD during power up or reset, and holding it at VDD until "<cr_lf>FACTORY RESET<cr_lf>" is printed from the UART.

Secondly, it can be used as a hardware shortcut to reconnect/advertise/cancel and disconnect when pulsed high for at least 5µs on the S2/S3 and 250µs on the D2.

- On all modules, pulsing PIO_4 while connected will perform an ATDH to disconnect.

- On single mode modules running Peripheral Observer firmware, pulsing PIO_4 while not advertising will perform an ATDSLE to start advertising. If already advertising, pulsing PIO_4 will perform and ATDC to cancel advertising.
- On dual mode modules and single mode modules running standard firmware, pulsing PIO_4 while not connecting will perform an ATDMLLE to connect out to the last connected device. If already connecting, pulsing PIO_4 will perform and ATDC to cancel connecting.
- On singe mode modules running standard firmware the behavior can be changed from ATDMLLE to ATDSLE using the ATSPIO command.
- The module will respond to the PIO_4 pulse with the same response as if the command was entered through the UART.

Single mode module's running PeripIf PIO_4's Direction in ATSPIO is set to 2, pulsing PIO_4 while not connected will perform and ATDSLE to start advertising instead of ATDMLLE to connect out.

3.4.4 PIO_6 – BRSP Comm Mode Toggle Input

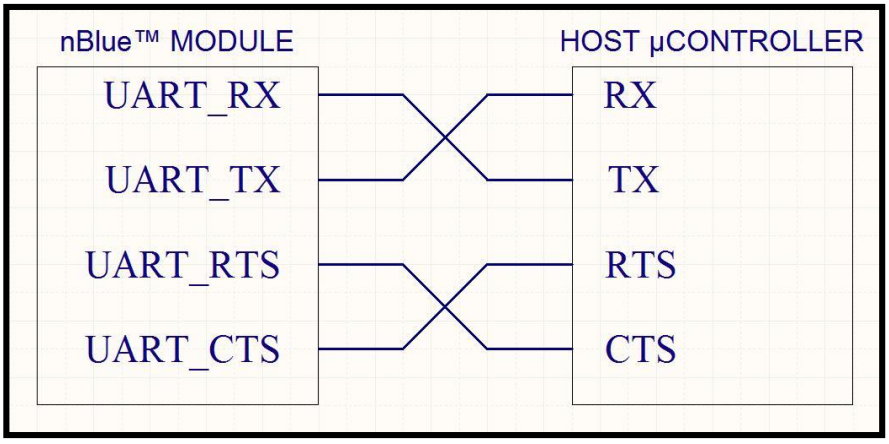
PIO_6 can be used as a hardware shortcut for toggling between command mode and data mode when connected by pulsing it high for at least 5µs on the S2 and 250µs on the D2. The module will respond with the same response as if the command was entered through the UART.

3.4.5 PIO_14 - Firmware Upgrade Mode Input

PIO_14 can be used to manually put the module into firmware upgrade mode by setting it to VDD during power up or reset and holding it at VDD until the "nBoot" message is sent from the UART.

3.5 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.



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). In order to receive commands and data via the UART, the UART needs to be active and in order 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.

4.1.1 Enabling/Disabling Sleep Mode

Sleep mode can be enabled by executing the ATZ command or by pulsing PIO_3 high 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 while the module is asleep. ATSZ can be used to configure single mode modules (S2/S3) 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. This feature is not available on dual mode modules (D2).

PIO_3 needs to be pulsed high for at least 20ns on the S2 and 250µs on the D2 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.

4.1.1 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.

4.1.2 Single Mode Sleep Power States

The following information applies to single mode modules only, and is meant to provide further details about the specific CC2540/2541 power modes being used when sleep is enabled or disabled.

The BR-LE4.0-S2 (S2) utilizes the Texas Instruments CC2540F256 SoC. For detailed specifications see the CC2540 datasheet: **http://focus.ti.com/lit/ds/symlink/cc2540.pdf**

The BR-LE4.0-S3 (S3) utilizes the Texas Instruments CC2541F256 SoC. For detailed specifications see the CC2541 datasheet: **http://focus.ti.com/lit/ds/symlink/cc2541.pdf**

Sleep Mode Enabled and Idle (Deep Sleep)
When sleep mode is enabled and the module is in the idle state, it will enter a deep sleep state known as Power Mode 3. In this state all internal clocks are disabled and the module will consume the least amount of current possible: ~0.4 µA (S2/S3).

Sleep Mode Enabled and Not Idle
When sleep mode is enabled and the module is not idle (advertising, discovering, connecting, connected) it cannot enter Power Mode 3 as it needs to keep a timer active to handle RF events. In this case it will use Power Mode 2 which consumes ~1.1 µA. Power Mode 2 will only be entered in between RF events, so the module will constantly be alternating between Power Mode 2, the active state and the RX/TX states to handle RF events.

Sleep Mode Disabled
When sleep mode is disabled the module will be in the active state with the UART enabled which consumes ~8.1mA.

4.2 IO Considerations

Make sure no current is being leaked through any IO. All IO on the module except for PIO_2/5,TX and RTS default to inputs. All inputs are pulled low internally except for RX, since it is normally driven by another UART. If the module is not connected to another UART, RX should be pulled high to keep it from floating.

If LEDs are connected to PIO_2 and PIO_5 (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. This allows the module to enter the deep sleep state when sleep mode is enabled. 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 ATSPL to lower the output power / receive sensitivity to the lowest possible values acceptable for the intended application.

4.5 Higher Scan/Advertising Interval

Use the ATSDST (CB) / ATSDSTLE (LE) commands to set the scan/advertising interval to the highest possible values acceptable for the intended application. Be aware that increasing the scan/advertising interval will lower the rate of scanning/advertising which will increase the amount of time it may take to connect.

4.6 Higher Connection Interval

For LE connections, 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

For LE connections, 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

For LE connections, 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 (.4uA S2/S3, 90uA D2)

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 during reset.

Make sure no current is being leaked through any IO. All IO on the module except for PIO_2/5,TX and RTS default to inputs. All inputs are pulled low internally except for RX, since it is normally driven by another UART.

4.10.1 Through Local UART

1. Put the module in the idle state using the ATDC command.
2. Enable sleep mode using PIO_3 or ATZ.
3. An S2/S3 module will now be drawing ~.4uA, a D2 module ~90uA.

4.10.2 Through Remote Command Mode

WARNING: After executing these instructions the module will not be connectable until advertising is re-enabled through the local UART.

If the module is not connected to another UART, RX should be pulled high to keep it from floating.

1. Connect to the module using ATDMLE.
2. Use ATMRC to enter remote command mode and wait for the BRSP,0,2 event.
3. Use ATSDBLE,0,0 to make the module default to the LE idle state to keep it from advertising after disconnecting.
4. (D2 Only) Use ATSDB,0,0 to make the module default to the CB idle state to keep it from being CB discoverable after disconnecting.
5. Enable sleep mode using PIO_3 or ATZ.
6. Disconnect using ATDH.
7. An S2/S3 module will now be drawing ~.4uA, a D2 module ~90uA.

4.11 Calculating Application Specific Current Consumption

For information on calculating application specific current consumption data, see:
http://www.ti.com/general/docs/litabsmultiplefilelist.tsp?literatureNumber=swra347a

5 Command Usage Guidelines

5.1 Command Usage

- All commands are entered in the following format: “COMMAND”<cr>. The command parser does not accept a line feed <lf> after the carriage return <cr>. If using HyperTerminal the following option should be disabled, or commands will not be submitted correctly: Send line ends with line feeds.
- 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.
- Due to connection timing constraints, changes to single mode module’s (S2/S3) configuration cannot be stored in flash while the module is connected. Changes will still take effect immediately, but they will not be permanently stored until the module has disconnected. If configuration changes are made or a new device is paired while connected and then power is lost before disconnecting, these changes will not have been saved. This does not apply to dual mode modules (D2), only single modes (S2/S3).
- 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. For commands that store in flash with optional parameters – if an optional parameter isn’t specified then the current value will be used instead of the default value.
- Commands/events are color coded to identify which module they are applicable to:

SD	**SINGLE/DUAL MODE COMMAND/EVENT**
SM	**SINGLE MODE ONLY COMMAND/EVENT**
SM PO	**SINGLE MODE PERIPHERAL OBSERVER COMMAND/EVENT**
DM	**DUAL MODE ONLY COMMAND/EVENT**

5.2 Common Parameter/Response Value Descriptions

- <cr> = 0x0D (carriage return)
- <cr_lf> = 0x0D0A (carriage return, linefeed)
- <Addr> = Device Address, 12 hex characters.
- <Conn_Handle> = Connection Handle, 0-9.
- <Att_Handle> = Attribute Handle, specific to the GATT commands and events, 1-65535.

5.3 Common Terms/Abbreviations

- CB = Classic Bluetooth (BR/EDR)
- LE = Low Energy

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)

SD	**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)

SD	**ERROR STATUS RESPONSE** **Function:** All unsuccessful commands will respond immediately with an ERROR response. **Response Format:** ERROR,<Error Code> **Response Values:** ▪ **Error Code:** 01 = Invalid Parameters 02 = Invalid Role 03 = Invalid State 04 = Invalid Password 05 = Invalid Connection Handle 06 = Configuration Locked 07 = List Error 08 = Hardware Error 09 = No Address Stored 10 = Bluetooth Error 11 = Memory Allocation Error 12 = GATT Request Pending **Example(s):** RESPONSE: **<cr_lf>** **ERROR,01<cr_lf>**

7 Events

7.1 General Events

7.1.1 Reset (BR-LE4.0-XX)

SD **RESET EVENT**

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

Event Format: <Module_Type>

Event Values:

- **Module_Type:**

BR-LE4.0-S2	S2 Single Mode LE Module
BR-LE4.0-S3	S3 Single Mode Power-Optimized LE Module
BR-LE4.0-D2	D2 Dual Mode BR/EDR and LE Module

Example(s):

1. Following a reset on an S2 module, the Module_Type is sent:

```
EVENT: <cr_lf>
       BR-LE4.0-S2<cr_lf>
```

Note(s):

- *This event will occur ~165ms after reset on the S2/S3 and ~880ms after reset on the D2.*

7.1.2 Done (DONE)

SD **DONE EVENT**

Function: The done event will be sent when a command that runs in the background times out or is cancelled.

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

Event Values:

- **Command_Type:**
  0 = Classic *Bluetooth* (CB)
  1 = Low Energy (LE)
  2 = General

- **Completed_Command:**
  Command_Type = 0 (CB):
  0 = ATDS
  1 = ATDI
  2 = ATDM
  3 = ATRRN
  4 = ATRRS

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

Command_Type = 2 (General):
0 = ATCS?
1 = ATTXT
2 = ATRXT

Example(s):

1. Advertising is started using ATDSLE, then cancelled using ATDC, which causes the DONE event to print, signaling that the module is no longer advertising:

```
COMMAND:  ATDSLE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
COMMAND:  ATDC<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DONE,1,0<cr_lf>
```

Note(s):

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

7.1.3 Connect (CONNECT)

SD

CONNECT EVENT

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

Event Format: CONNECT,<Conn_Handle>,<Conn_Type>,<Pair_Status>,<Addr>

Event Values:

- **Conn_Handle:** Connection handle
- **Conn_Type:**
  0 = Classic *Bluetooth*
  1 = Low Energy
  2 = Classic *Bluetooth* with iOS Device (Requires MFi)
- **Pair_Status:**
  0 = Not Paired
  1 = Paired, Unauthenticated
  2 = Paired, Authenticated
- **Addr:** The address 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:

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

7.1.4 Disconnect (DISCONNECT)

SD **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:

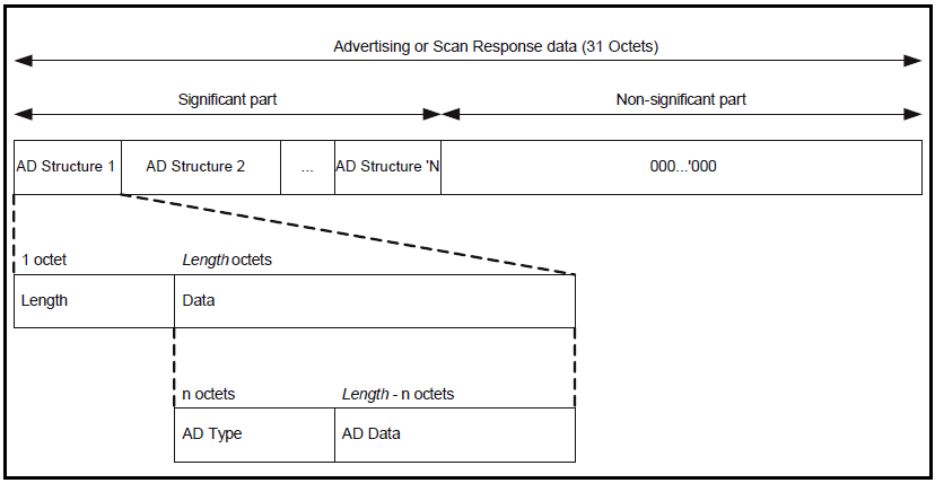
```
COMMAND:   ATDH,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           DISCONNECT,0,0<cr_lf>
```

7.1.5 Discovery (DISCOVERY)

SD **DISCOVERY EVENT**

Function: The discovery event will occur when a device is discovered after the ATDI or ATDILE command has been issued.

Advertising and scan response data (Extended Inquiry Response (EIR) data for CB devices) 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. LE advertising and scan response data can be up to 31 bytes long, CB EIR data can be up to 240 bytes long.



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

Descriptions of each AD/EIR Type can be found in the following document:
https://bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=260933

Event Format: DISCOVERY,<Discovery_Type>,<Addr>,<COD/Addr_Type>,<RSSI>,
<Data_Structure_Count>,<Data_Structures>

Event Values:

- **Discovery_Type:**
  0 = Classic Bluetooth Inquiry Response
  1 = Classic Bluetooth Extended Inquiry Response

  2 = Low Energy Connectable Indirect Advertisement (Connectable + Scannable)
  3 = Low Energy Connectable Direct Advertisement (Connectable Only)
  4 = Low Energy Scannable Indirect Advertisement (Scannable Only, Not Connectable)
  5 = Low Energy Non-connectable Indirect Advertisement (Not Connectable or Scannable)
  6 = Low Energy Scan Response

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

- **COD/Addr_Type:** Value is based on the Discovery Type.

  Discovery_Type = 0-1 (Classic Bluetooth): Class of Device

  Discovery_Type = 2-6 (Low Energy): Address Type
  0 = Public Address
  1 = Static Random Address
  2 = Non-resolvable Private Random Address
  3 = Resolvable Private Random Address

- **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 ATDI command is used to start a CB discovery. A single device is found, with an address of ECFE7E000000, a COD of 000000, an RSSI of -34 and 0 data structures:

```
COMMAND:  ATDI<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,0,ECFE7E000000,000000,-034,0,0<cr_lf>
          <cr_lf>
          DONE,1,1<cr_lf>
```

2. 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<c _lf>
EVENT:    <cr_lf>
          DISCOVERY,2,ECFE7E000001,0,-045,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,6,ECFE7E000001,0,-045,1,
          1109426C7565526164696F73303030303031<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,3,ECFE7E000002,0,-045,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DONE,1,1<cr_lf>
```

7.1.6 Pairing Request (PAIR_REQ)

SD	**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,<Addr>,<Pairing_Type>

Event Values:

- **Addr:** The address of the device requesting pairing.

- **Pairing_Type:**
  0 = Classic *Bluetooth*
  1 = Low Energy

Example(s):

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

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

7.1.7 Paired (PAIRED)

SD **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,<Addr>,<Pairing_Status>

Event Values:

- **Addr:** The address of the device paired device.

- **Pairing_Status:**
  1 = Paired, Unauthenticated
  2 = Paired, Authenticated

  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.

  0 = Unbonded Paired, Unauthenticated
  3 = Unbonded Paired, Authenticated

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,ECFE7E000001,1<cr_lf>
```

7.1.8 Pairing Failed (PAIR_FAIL)

SD

PAIRING FAILED EVENT

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

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

Event Values:

- **Addr:** The address 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 doesn't support pairing, triggering the PAIR_FAIL event:

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

7.1.9 Passkey Request (PK_REQ)

SD

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,<Addr>

Event Values:

- **Addr:** The address 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,ECFE7E000001<cr_lf>
    COMMAND:    ATPKR,ECFE7E000001,123456<cr>
    RESPONSE:   <cr_lf>
                OK<cr_lf>
    EVENT:      <cr_lf>
                PAIRED,ECFE7E000001,1<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.*

7.1.10 Passkey Display (PK_DIS)

SD **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,<Addr>,<Passkey>

Event Values:

- **Addr:** The address of the remote device.
- **Passkey:** The passkey to be displayed. 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_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,ECFE7E000001,123456<cr_lf>
    EVENT:      <cr_lf>
                PAIRED,ECFE7E000001,1<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.*

7.2 Low Energy Events

7.2.1 Connection Parameter Update Status (SCCPS)

SD **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.*

7.2.2 Connection Parameter Update (CPU)

SD **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.*

7.2.3 GATT Done (GATT_DONE)

SD

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
  1 = ATGDIS
  2 = ATGDC/ATGDCU
  3 = ATGDCD/ATGDCDU
  4 = ATGR/ATGRU
  5 = ATGW
  6 = ATGRL
  7 = ATGWP
  8 = ATGWPE
- **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 doesn't 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,0,4,01<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,0,5,00<cr_lf>
```

7.2.4 GATT Discovered Primary Service (GATT_DPS)

SD

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 4 primary services are discovered: GAP, GATT, BAS (Battery) and BRSP. When all of the primary services have been discovered a GATT_DONE event it triggered:

```
COMMAND:  ATGDPS,0<cr>
```

```
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00001,00011,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00012,00014,1801<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00016,00019,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00015,65535,DA2B84F1627948DEBDC0AFBEA0226079<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<cr_lf>
```

7.2.5 GATT Discovered Characteristic (GATT_DC)

SD **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_Mask>,<Char_UUID>

Event Values:

- **Conn_Handle:** Connection handle.
- **Char_Value_Att_Handle:** Characteristic value attribute handle. [1-65535]
- **Char_Properties_Mask:** 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,0,00003,02,2A00<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,00<cr_lf>
```

7.2.6 GATT Discovered Characteristic Descriptor (GATT_DCD)

SD

GATT DISCOVER CHARACTERISTIC DESCRIPTOR EVENT

Function: This event will be sent for each characteristic descriptor discovered by an ATGDCD or ATGDCDU 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 all characteristic descriptors on a remote device. 4 Client Characteristic Configuration (UUID 2902) descriptors are found at handles 15,19, 24 and 29. A GATT_DONE is triggered once all descriptors have been found:

```
COMMAND:  ATGDCD,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DCD,0,00015,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DCD,0,00019,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DCD,0,00029,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DCD,0,00024,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,3,00<cr_lf>
```

7.2.7 GATT Characteristic/Descriptor Value (GATT_VAL)

SD

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_Length>,<Value>

Event Values:

- **Conn_Handle:** Connection handle.
- **Value_Att_Handle:** Value attribute handle. [1-65535]

- **Value_Event_Type:**
  0 = Read Response
  1 = Notification
  2 = Indication

- **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,0,00003,0,16,426C7565526164696F73303030303031<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,4,00<cr_lf>
```

7.2.8 GATT Long Characteristic/Descriptor Value (GATT_LVAL)

SD **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:** Patial 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 secod 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,0,00054,000,22,
                11223344556677889900112233445566778899001122<cr_lf>
EVENT:          <cr_lf>
                GATT_LVAL,0,00054,022,11,33445566778899001122 33<cr_lf>
EVENT:          <cr_lf>
                GATT_DONE,0,6,00<cr_lf>
```

7.2.9 Address Resolved (RESOLVED)

SD **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<c _lf>
EVENT:    <cr_lf>
          DISCOVERY,2,7CED99C0E2B3,3,-045,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,6,7CED99C0E2B3,3,-045,1,
          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,7CED99C0E2B3,0,3<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,1,0,7CED99C0E2B3<cr_lf>
```

```
COMMAND:   ATPLE,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           RESOLVED,7CED99C0E2B3,ECFE7E000001<cr_lf>
EVENT:     <cr_lf>
           PAIRED,ECFE7E000001,1<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<c _lf>
EVENT:     <cr_lf>
           DISCOVERY,2,ECFE7E000001,0,-045,4,020106-020A04-051208000800-
           1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:     <cr_lf>
           DISCOVERY,6,ECFE7E000001,0,-045,1,
           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,0,1,1,ECFE7E000001<cr_lf>
```

Note(s):

- *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.*

7.2.10 BRSP Status (BRSP)

SD

BRSP STATUS EVENT

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

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

Event Values:

- **Conn_Handle:** Connection handle

- **BRSP_Status:**
  - 1 = Data Mode
  - 2 = Remote Command Mode
  - 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

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,ECFE7E000001,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<cr_lf>
EVENT:    <cr_lf>
          BRSP,0,1<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.*

7.2.11 Battery Event (BATT)

SD

BATTERY EVENT

Function: This event will be fired based on the configuration of the ATSBAS (Battery Profile) command.

If BAS is enabled in Manual mode the BATT event will fire when a client device has registered/deregistered to receive notifications from the Battery Level characteristic of the BAS service. When a client registers for notifications the battery level should be updated using ATSBAS if it changes, which will cause the service to notify the client of the change in level.

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.

Event Format: BATT,<Notify_Level>

Event Values:

- **Notify_Level:**

  ATSBAS Mode = 0 (Manual)
  Notify:
  0 = Client has deregistered for notifications
  1 = Client has registered for notifications

ATSBAS Mode = 1 (Automatic)
Level: 0-99 %

Example(s):

1. A client registers for notifications and then disconnects (which deregisters.)

```
EVENT:    <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<cr_lf>
EVENT:    <cr_lf>
          BATT,1<cr_lf>
EVENT:    <cr_lf>
          BATT,0<cr_lf>
EVENT:    <cr_lf>
          DISCONNECT,0,0<cr_lf>
```

7.2.12 Heart Rate Service Event (HRS)

SM PO

HEART RATE SERVICE EVENT

Function: This event will fire when a client device has registered/deregistered to receive notifications from the Heart Rate Measurement characteristic of the HRS service. When a client registers for notifications, the heart rate value should be updated using ATHRV approximately 1 time per second which will cause the service to notify the client of the new value.

Event Format: HRS,<Notify>

Event Values:

- **Notify:**
  0 = Client has deregistered for notifications
  1 = Client has registered for notifications

Example(s):

1. A client registers for notifications and then disconnects (which deregisters.)

```
EVENT:    <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<cr_lf>
EVENT:    <cr_lf>
          HRS,1<cr_lf>
EVENT:    <cr_lf>
          HRS,0<cr_lf>
EVENT:    <cr_lf>
          DISCONNECT,0,0<cr_lf>
```

7.2.13 Proximity Profile Service Event (PXP)

SM PO

PROXIMITY PROFILE SERVICE EVENT

Function: This event will fire when the alert level had changed for either the Link Loss Service or the

Immediate Alert Service.

Event Format: PXP,<Service>,<Alert_Level>

Event Values:

- **Service:**
  0 = Link Loss Service generated the event.
  1 = Immediate Alert Service generated the event.

- **Alert_Level:**
  0 = None - no alerting shall be done.
  1 = Mild - the device shall alert.
  2 = High - the device shall alert in the strongest possible way.

  The specific action that occurs for the Mild and High alert levels is implementation specific. Alerts could included flashing lights, making noise, vibrating or any other method to alert that could alert the user. Alerts continue until one of the following occurs: an implementation specific timeout, a user interaction to disablet the alert, a new alert level is written or the link is disconnected. If a new alert level is written or the link is disconnected a new PXP event will be generated.

Example(s):

1. A client connects and enables a Mild alert level in the Immediate Alert Service then disconnects. A new PXP event is sent disabling the alert when the client disconnects.

```
EVENT:    <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<cr_lf>
EVENT:    <cr_lf>
          PXP,1,1<cr_lf>
EVENT:    <cr_lf>
          PXP,1,0<cr_lf>
EVENT:    <cr_lf>
          DISCONNECT,0,0<cr_lf>
```

2. A client connects and enables a High alert level in the Link Loss Service. When the client goes out of range and the connection drops a PXP event is sent to enable the alert. When the connection is reestablished the alert is disabled.

```
EVENT:    <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<cr_lf>
EVENT:    <cr_lf>
          PXP,0,2<cr_lf>
EVENT:    <cr_lf>
          DISCONNECT,0,2<cr_lf>
EVENT:    <cr_lf>
          PXP,0,0<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<cr_lf>
```

7.3 Classic Bluetooth Events

7.3.1 Remote Name (RN)

DM	REMOTE NAME EVENT

Function: The remote name event will be sent to return a remote device name after issuing an ATRRN command.

Event Format: RN,<Device_Name>

Event Values:

- **Device_Name:** The *Bluetooth* Device Name of the remote device.

Example(s):

1. ATRRN is used to request the device name of ECFE7E000001, when the process is complete an RN event is triggered, followed by a DONE event:

```
COMMAND:  ATRRN,ECFE7E000001<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          RN,BlueRadiosECFE7E000001<cr_lf>
EVENT:    <cr_lf>
          DONE,0,3<cr_lf>
```

7.3.2 Remote Service (RS)

DM	REMOTE SERVICE EVENT

Function: The remote service name event will be sent to return a remote service after issuing an ATRRS command.

Event Format: RS,<Service_UUID_Profile_Type>,<RFCOMM_Channel>,<Service_Name>

Event Values:

- **Service_UUID_Profile_Type:** UUID or Type of service. If ATRRS was issued with Service_UUID set to a UUID, the Service_UUID will print. If ATRRS was issued with Service_UUID set to 0, then the Service_Type will print.

  Service Types:

0 = Unknown	14 = Basic Printing Reflection UI
1 = Serial Port	15 = Basic Printing Status
2 = Headset Audio Gateway	16 = Basic Imaging Responder
3 = Headset	17 = Basic Imaging Reference Objects
4 = Dial Up Networking	18 = Basic Imaging Archive
5 = Fax	19 = Video Distribution Sink
6 = LAN Access	20 = Video Distribution Source
7 = Object Push	21 = Phonebook Access Server
8 = File Transfer	22 = Phonebook Access Client

9 = Synchronization	23 = Message Access Server
10 = Hands Free Audio Gateway	24 = Message Access Notification Server
11 = Hands Free	25 = iOS Device
12 = SIM Access	26 = iOS Accessory
13 = Basic Printing Reference Objects	

- **RFCOMM_Channel:** RFCOMM channel of service. [1-31]
- **Service_Name:** The name of the service.

Example(s):

1. ATRRS is used to search for SPP services (UUID 1101) on ECFE7E000001. 2 services are found, one at RFComm Channel 1 and the other at RFComm Channel 2. When the process is complete a DONE event is triggered:

```
COMMAND:  ATRRS,ECFE7E000001,1101<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          RS,1101,01,COM1<cr_lf>
EVENT:    <cr_lf>
          RS,1101,02,COM2<cr_lf>
EVENT:    <cr_lf>
          DONE,0,4<cr_lf>
```

2. ATRRS is used to search for SPP services (UUID 1101) on ECFE7E000001, but no services are found:

```
COMMAND:  ATRRS,ECFE7E000001,1101<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          RS,0,0,0<cr_lf>
EVENT:    <cr_lf>
          DONE,0,4<cr_lf>
```

7.3.3 User Confirmation Request (UCR)

DM **USER CONFIRMATION REQUEST**

Function: The user confirmation request event will be sent when a numeric value needs to be displayed and confirmed for authentication during pairing, when Simple Secure Pairing is used.

Event Format: UC_REQ,<Addr>,<Numeric_Value>,<Response_Needed>

Event Values:

- **Addr:** The address of the remote device.
- **Numeric_Value:** The numeric value to be displayed. 6 numeric characters.
- **Response_Needed:**

0 = No response needed, Numeric_Value just needs to be displayed.
1 = Response needed, Numeric_Value needs to be displayed and confirmed using the ATUCR command.

Example(s):

1. CB pairing is initiated using the ATP command to connection handle 0. Authentication is required by one side or the other and based on the module's IO_Capabilities (ATSP), UC_REQ is sent to the local module. The Numeric_Value is confirmed using the ATUCR command. Once both sides confirm the Numeric_Value, pairing is completed and the PAIRED event is sent:

```
COMMAND:  ATP,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          UC_REQ,ECFE7E000001,123456,1<cr_lf>
COMMAND:  ATUCR,ECFE7E000001,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,ECFE7E000001,1<cr_lf>
```

7.3.4 PIN Request (PIN_REQ)

DM

PIN REQUEST

Function: The PIN request event will be sent when a PIN is needed for authentication during pairing, when the legacy pairing procedure is used.

Response Format: PIN_REQ,<Addr>

Response Values:

- **Addr:** The address of the remote device.

Example(s):

1. CB pairing is initiated using the ATP command to connection handle 0. Authentication is required by one side or the other and since the remote device is a legacy pairing device (< Bluetooth v2.1), PIN_REQ is sent to the local module. The remote device will either display a PIN or have it available in its documentation. In this case the PIN is "default" and is entered using the ATPINR command. Once pairing is completed the PAIRED event is sent :

```
COMMAND:  ATP,00A0962EDB41<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PIN_REQ,00A0962EDB41<cr_lf>
COMMAND:  ATPINR,00A0962EDB41,default<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,ECFE7E000001,1<cr_lf>
```

Note(s):

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

7.3.5 SPP Status (SPP)

DM **SPP STATUS EVENT**

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

Event Format: SPP,<Conn_Handle>,<SPP_Status>

Event Values:

- **Conn_Handle:** Connection handle
- **SPP_Status:**
  1 = Data Mode
  2 = Remote Command Mode
  6 = Requested Mode Not Supported

Example(s):

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

```
COMMAND:  ATDM,ECFE7E000001<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,0,0,ECFE7E000001<cr_lf>
EVENT:    <cr_lf>
          SPP,0,1<cr_lf>
```

Note(s):

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

7.3.6 IOS MFi Authentication Status (IOSSTAT)

DM **IOS MFI AUTHENTICATION STATUS EVENT**

Function: The MFi Authentication Status event will be sent when the MFi authentication process completes. Apple requires Bluetooth devices connecting to iOS devices over classic Bluetooth (not LE) using Bluetooth profiles that are not supported by iOS (such as Serial Port Profile) to be authenticated as licensed MFi (Made for iPhone/iPod/iPad) accessories.

Support for MFi authentication has been added to the D2, but documentation can only be provided to verified Apple MFi licensees, contact BlueRadios for more information.

For more information on the MFi program, see: https://developer.apple.com/programs/mfi/

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

Event Values:

- **Conn_Handle:** Connection handle
- **Status:**
  0 = Success
  1 = Failure

Example(s):

1. A connection is made to iOS device AAAAAAAAAAAA over classic Bluetooth, but once SPP enters data mode authentications fails since the D2 is not configured for MFi support:

```
COMMAND:  ATDM,AAAAAAAAAAAA<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,1,2,1,AAAAAAAAAAAA<cr_lf>
EVENT:    <cr_lf>
          SPP,1,1<cr_lf>
EVENT:    <cr_lf>
          IOSSTAT,1,1<cr_lf>
EVENT:    <cr_lf>
          DISCONNECT,1,1<cr_lf>
```

8 General Commands

8.1 AT

SD	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>
```

8.2 Help

SD	HELP

Function: Provides the address of the BlueRadios forum.

Example(s):

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

8.3 Reset Commands

8.3.1 Reset (ATRST)

SD	RESET

Function: Resets the module.

Command Format: ATRST

Example(s):

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

```
COMMAND:  ATRST<cr>
RESPONSE: <cr_lf>
          BR-LE4.0-S2<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.*

8.3.2 Factory Reset (ATFRST)

SD

FACTORY RESET

Function: Resets the module back to its factory defaults.

Command Format: ATFRST

Example(s):

1. An ATFRST is sent and once the module has reset, the RESET event is triggered, preceded by the message “FACTORY_RESET”.

```
COMMAND:  ATFRST<cr>
RESPONSE: <cr_lf>
          FACTORY_RESET<cr_lf>
          <cr_lf>
          BR-LE4.0-S2<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 during reset can also be used to factory reset a module.*

8.4 Sleep Commands

8.4.1 Sleep (ATZ)

SD

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.

Command Format: ATZ

Example(s):

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

Note(s):

- ***When sleep mode 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.***
- *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)

SD **SET SLEEP**

Function: Configures sleep mode.

Command Format: ATSZ,<Sleep_On_Reset>,<Wake_On_Rx>,<PIO_Sleep_Status_Mask>

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:** (Single Mode Only, must be 0 on Dual Modes)
  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_Sleep_Status_Mask:**
  0 = Disabled.
  1 = PIO5's default behavior will be overridden to show sleep mode status. PIO5 will be low when sleep mode is enabled, and high when sleep mode is disabled and the module is awake and able to receive commands.
  2 = PIO2's default behavior will be overridden to show when the module is in its low power state. PIO2 will be high when the MCU is idle and in the low power state, and low when it is active. This status is intended for debugging/observation only.

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 PIO5 sleep status to know when the module is awake. If neither of these lines can be monitored, wait at least 5ms before sending data.*

GET SLEEP

Function: Gets the sleep configuration.

Command Format: ATSZ?

Response Format: <Sleep_On_Reset>,<Wake_On_Rx>,<PIO_Sleep_Status_Mask>

Example(s):

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

8.5 Module Information Commands

8.5.1 Module Type (ATMT?)

SD **GET MODULE TYPE**

Function: Gets the module type.

Command Format: ATMT?

Response Format: <Module_Type>

Response Values:

- **Module_Type:**

BR-LE4.0-S2	S2 Single Mode LE Module
BR-LE4.0-S3	S3 Single Mode Power-Optimized LE Module
BR-LE4.0-D2	D2 Dual Mode BR/EDR and LE Module

Example(s):

```
COMMAND:  ATMT?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          BR-LE4.0-S2<cr_lf>
```

8.5.2 Stack Type (ATST?)

SD **GET STACK TYPE**

Function: Gets the stack type.

Command Format: ATST?

Response Format: BlueRadios nBlue (Single Mode) / Stonestreet One nBlue (Dual Mode)

Example(s):

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

8.5.3 Firmware Version (ATV?)

SD

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>
          3.5.0.0-S2<cr_lf>
```

8.5.4 *Bluetooth* Device Address (ATA?)

SD

GET ADDRESS

Function: Gets the module's *Bluetooth* Device Address.

Command Format: ATA?

Response Format: <Addr>

Response Values:

- **Addr:** Local *Bluetooth* device address

Example(s):

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

8.5.5 *Bluetooth* Device Name (ATSN)

SD

SET NAME

Function: Sets the module's *Bluetooth* 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:** *Bluetooth* Device Name – SM: 1-20 characters, DM: 1-32 characters
  Default: BlueRadios######

  Part or all of the *Bluetooth* 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 over GATT.
  1 = Writeable - The name can be changed by any remote device over GATT.
  2 = Authenticated Writeable - Name can only be changed by authenticated devices over GATT.

Example(s):

(*Bluetooth* Device Address = ECFE7E123456)

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

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

GET NAME

Function: Gets the module's *Bluetooth* Device Name.

Command Format: ATSN?

Response Format: <Name>

Example(s):

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

8.6 Module Status Commands

8.6.1 Connection Status (ATCS?)

SD **GET CONNECTION STATUS**

Function: Gets the details of an existing 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_Type>,<Pairing_Status>,<Addr>

Response Values:

- **Conn_Handle:** Connection handle
- **Conn_Type:**
  0 = Classic *Bluetooth* Connection
  1 = Low Energy Connection
  2 = Classic *Bluetooth* Connection with iOS Device (Requires MFi)
- **Pairing_Status:**
  0 = Not Paired
  1 = Paired, Unauthenticated (Encrypted)
  2 = Paired, Authenticated (Encrypted)
- **Addr:** Address of remote device

Example(s):

1. One active connection is found, it is an unpaired LE connection to ECFE7E00000. Then DONE event is triggered to signal the end of the connection list.

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

8.6.2 RSSI (ATRSSI?)

SD	GET 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. The RSSI has different meanings for LE and CB connections, which are detailed below.

Command Format: ATRSSI?,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to read RSSI from. If not specified 0 will be used.

Response Format: <RSSI_Value>

Response Values:

- **RSSI_Value:**

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

  CB: Difference between the measured absolute RSSI and the limits of the Golden Receive Power Range. A positive RSSI indicates how many dB the RSSI is above the upper limit, and a negative value indicates how many dB the RSSI is below the lower limit. A value of 0 is within the Golden Receive Power Range. The lower limit of the Golden Receive Power Range corresponds to a receive power between -56 dBm and 6 dB above the actual sensitivity of the receiver. The upper limit is 20 dB above the lower threshold level to an accuracy of +/-6 dB.
  [-128 – 127 dB]

Example(s):

```
COMMAND:  ATRSSI?,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          -034<cr_lf>
```

Note(s):

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

8.7 Configuration Control Commands

8.7.1 Configuration Lock (ATSCL)

SD	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,04<cr_lf>
COMMAND:  ATSCL,0,good_password<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *All configuration commands will reply ERROR,06 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>
```

8.7.2 Flash Configuration (ATFC)

SD	**FLASH CONFIG**

Function: Stores the current configuration in flash. This command is intended to be used after all settings have been configured. If Auto_Flash 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 of 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.*

8.7.3 Configure Auto Configuration Flashing (ATSFC)

SD	**SET FLASH CONFIGURATION**

Function: Sets the configuration flashing settings.

Command Format: ATSFC,<Auto_Flash>

Command Parameter(s):

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

Example(s):

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

GET FLASH CONFIGURATION

Function: Gets the configuration flashing settings.

Command Format: ATSFC?

Response Format: <Auto_Flash>

Example(s):

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

8.8 Response Configuration Commands

8.8.1 Response Mode Configuration (ATSRM)

SD

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>

Command Parameter(s):

- **Response_Mode:**
  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. The following events will be sent in this mode: DISCOVERY, GATT_DPS, GATT_DC, GATT_DCD, GATT_VAL, RN, RS.

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

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>

Example(s):

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

8.8.2 Discovery Event Formatting (ATSDIF)

SD

SET DISCOVERY FORMATTING

Function: Sets discovery event formatting parameters.

Command Format: ATSDIF,<Data_Structure_Mask>,<Name_Format>,<Hyphens>

Command Parameter(s):

- **Data_Structure_Mask:** Data structures enabled in the mask will be printed in discovery events. If set to 0, no data structures will be printed.
  - 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
  - 2048 (0x0400) = Target Address
  - 1024 (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

Example(s):

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

Note(s):

- *This command only changes how 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>

Example(s):

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

8.9 Hardware Configuration / Control Commands

8.9.1 Set Power Level (ATSPL)

The ATSPL command takes different arguments for single and dual mode modules, so the documentation is split into two sections below. This is due to dual modes having separate output power levels for CB and LE and a fixed receive sensitivity.

SM

SET POWER LEVEL (Single Mode)

Function: Sets the transmit power and receive sensitivity of a single mode module. The S2 module has a maximum output power of 4dBm and the S3 module has a maximum output power of 0dBm.

Command Format: ATSPL,<TX_Power>,<RX_Sensitivity>

Command Parameter(s):

- **TX_Power:**

  S2
  0 = -23 dBm
  1 = -6 dBm
  2 = 0 dBm
  3 = 4 dBm

  S3
  0 = -23 dBm
  1 = -6 dBm
  2 = 0 dBm

- **RX_Sensitivity:**
  0 = Standard Gain Mode (-89 dBm S2 / -90 dBm S3)
  1 = High Gain Mode (-95 dBm S2 / -96 dBm S3)

Example(s):

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

Note(s):

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

GET POWER LEVEL (Single Mode)

Function: Gets the transmit power and receive sensitivity of a single mode module.

Command Format: ATSPL?

	Response Format: <TX_Power>,<RX_Sensitivity> **Example(s):** COMMAND: **ATSPL?<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **<cr_lf>** **2,1<cr_lf>**
DM	**SET POWER LEVELS (Dual Mode)** **Function:** Sets the LE and CB transmit power of a dual mode module. **Command Format:** ATSPL,<LE_TX_Power>,<CB_TX_Power> **Command Parameter(s):** ▪ **LE_TX_Power:** Integer value from -23 to 10 [dBm]. Default: 10 ▪ **CB_TX_Power:** Integer value from -23 to 12 [dBm]. Default: 10 **Example(s):** COMMAND: **ATSPL,10,10<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **Note(s):** ▪ **The module needs to be reset after using ATSPL in order for the new power levels to take effect.** ▪ *Receive_Sensitivity is not adjustable on Dual Mode modules and will stay fixed at -95dBm.* ▪ *See the Module User's Guide or Module Datasheet for current consumption numbers at different power settings.*
	GET POWER LEVELS (Dual Mode) **Function:** Gets the LE and CB transmit power of a dual mode module. **Command Format:** ATSPL? **Response Format:** <LE_TX_Power>,<CB_TX_Power> **Example(s):** COMMAND: **ATSPL?<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **<cr_lf>** **10,10<cr_lf>**

8.9.2 UART Configuration (ATSUART)

SD **SET UART**

Function: Configures the module's UART. This command requires a reset for the new settings to take effect.

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

Command Parameter(s):

- **Baud_Rate:** 3-10 [9600bps – 921600]
  (230400, 460800, 921600 are only available on Dual Mode modules.)

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

WARNING: When streaming bi-directional data at the maximum rate over a CB connection at 921600 baud, data may be lost.

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

- **Stop_Bits:**
  0 = One
  1 = Two

- **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):

- ***Requires a reset for the setting to go into effect on Single Mode Modules.***

- *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 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>
```

8.9.3 PIO Configuration (ATSPIO)

SD **SET PIO**

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 direction and values of PIO's.

Command Format: ATSPIO,<PIO_Num>,<Direction>,<Value>

Command Parameter(s):

- **PIO_Num:**
  Single Mode: 0,1,2,4,5,7,8,9,10,11,12,13,14
  Dual Mode: 0,1,2,5,7,8,9,10,11,12,13,14,19,20,21,22

PIO_Num	Pin_Name	PIO_Num	Pin_Name
0	ADC_0	10	SPI_MISO
1	ADC_1	11	SPI_CSB
2	PIO_2	12	SPI_CLK
3	PIO_3	13	SPI_MOSI
4	PIO_4	14	PIO_14
5	PIO_5	19	PIO_19
6	PIO_6	20	PIO_20
7	PIO_7	21	PIO_21
8	PIO_8	22	PIO_22
9	PIO_9		

- **Direction:**
  0 = Input
  1 = Output
  2 = Special Function **(Single Mode modules only)**
  - PIO_4: If set, pulsing PIO_4 will control advertising instead of connecting to the last device.
  - PIO_8: If set, PIO_8 can be used to control the bypass line of the TI TPS62730 DC-DC Converter.

- **Value: (Must be 0 if direction is set to input)**

	0 = 0V 1 = VDD **Example(s):** COMMAND: **ATSPIO,7,1,1<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **Note(s):** ▪ *PIO_3, PIO_4 and PIO_6 cannot be modified.* ▪ *The value of PIO_2 and PIO_5 can be modified if Duty_Cycle is set to 0 in ATSLED.*
	GET PIO **Function:** Reads PIO settings. **Command Format:** ATSPIO?,<PIO_Num> **Command Parameter(s):** ▪ **PIO Number:** Single Mode: 0-18 Dual Mode: 0-22 **Response Format:** <Direction>,<Value> **Example(s):** COMMAND: **ATSPIO?,7<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **<cr_lf>** **1,1<cr_lf>**

8.9.4 LED Configuration (ATSLED)

SD	**SET LED** **Function:** Sets the status LED (PIO_2/PIO_5) parameters. **Command Format:** ATSLED,<Led_Num>,<Duty_Cycle>,<Period>,<Override_State> **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. ▪ **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

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

- **Override_State:**
  0 = Disabled - The leds will only pulse when in the states described above.
  1 = Enabled - The led state behavior will be overridden and the led will pulse immediately.

Example(s):

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

Note(s):

- *PIO_2 will pulse when the module is connected.*
- *PIO_5 will pulse when the module is not in the idle state, excluding the connected state as it is handled by PIO_2.*

GET LED

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

Command Format: ATSLED?,<Led_Num>,<Override_State>

Command Parameter(s):

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

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

Example(s):

```
COMMAND:  ATSLED?,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          010,02000,0<cr_lf>
```

8.9.5 Get ADC (ATADC?)

SM

GET ADC

Function: Takes a reading from one of the module's ADCs in mV using an internal 1.25V voltage reference.

Command Format: ATADC?,<ADC_Num>

Command Parameter(s):

- **ADC_Num:** 0-5,14,15

ADC_Num	PIO_Num	Pin_Name

0	0	ADC_0
1	1	ADC_1
2	10	SPI_MISO
3	11	SPI_MOSI
4	12	SPI_CSB
5	13	SPI_CLK
14	Internal	Temperature Sensor
15	Internal	VDD/3

Response Format: <ADC_Value>

Response Value(s):
- **ADC_Value:** 0-1250 [mV]

Example(s):

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

8.9.6 Get Battery Level (ATBL?)

SM

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>
```

8.9.7 Get Temperature (ATT?)

SM

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):

```
COMMAND:  ATT?<cr>
RESPONSE: <cr_lf>
          OK
          <cr_lf>
          026,079<cr_lf>
```

Note(s):

- *The initial accuracy of the internal temperature sensor is ±10°C, once calibrated using the ATCT command the accuracy is ±5°C.*

8.9.8 Calibrate Temperature Sensor (ATCT)

SM

CALIBRATE TEMPERATURE

Function: Calibrates the module's internal temperature sensor to an accuracy of ±5°C.

Command Format: ATCT,<Temp_Celsius>

Command Parameter(s):

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

Example(s):

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

Note(s):

- *The initial accuracy of the internal temperature sensor is ±10°C, once calibrated using the ATCT command the accuracy is ±5°C.*

8.10 Serial Profile Commands

8.10.1 Serial Profile Configuration (ATSSP)

SD

SET SERIAL PROFILE

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 from any other 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 default communication mode, 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>
          043,00000<cr_lf>
```

8.10.2 Command Mode (+++)

SD **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,ECFE7E000001,1<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,0,1,0,ECFE7E000001<cr_lf>
EVENT:     <cr_lf>
           BRSP,0,1<cr_lf>
TXDATA :   HELLO
COMMAND:   +++<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
COMMAND:   ATDH,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
           <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.*

8.10.3 Data Mode (ATMD)

SD **DATA MODE**

Function: Puts the module into Data Mode. In Data Mode all UART data is sent to the remote device using either the SPP service for CB connections or 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<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,0,1,0,ECFE7E000001<cr_lf>
COMMAND:   ATMD<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT    : <cr_lf>
           BRSP,0,1<cr_lf>
TXDATA   : HELLO
```

Note(s):

- *The module must be connected to use this command.*

8.10.4 Remote Command Mode (ATMRC)

SD **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 either the SPP service for CB connections or 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. **Events generated on the remote side will not be returned over LE connections due to limited packet size, but they will be returned over SPP connections on D2 devices, so the D2 can be used to bridge CB only devices to LE devices.**

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

Note(s):

- *The module must be connected to use this command.*
- ***Remote command mode is enabled by default for both LE and CB, 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 for LE and ATSMRC for CB.***
- *Due to connection timing constraints, changes to single mode module's (S2/S3) configuration cannot be stored in flash while the module is connected. Changes will still take effect immediately, but they will not be permanently stored until the module has disconnected. If configuration changes*

	are made or a new device is paired while connected and then power is lost before disconnecting, these changes will not have been saved. This does not apply to dual mode modules (D2), only single modes (S2/S3). ▪ *A D2 module can be put into remote command mode over a CB SPP connection by sending "ATMRC<cr>" immediately after connecting, prior to sending any other data. This allows non-BlueRadios devices to be able to put a D2 into remote command mode.*

8.10.5 CB Remote Command Mode Configuration (ATSMRC)

DM	**SET REMOTE COMMAND MODE CB** **Function:** This command is used to enable/disable remote command mode over the CB SPP connection. **Command Format:** ATSMRC,<Enable> **Command Parameter(s):** ▪ **Enable:** 0 = Disabled – Remote command mode disabled over SPP 1 = Enabled – Remote command mode enabled over SPP **Example(s):** COMMAND: **ATSMRC,1<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>**
	GET SPP REMOTE COMMAND MODE CB **Function:** Gets the CB remote command mode settings. **Command Format:** ATSMRC? **Response Format:** <Enable> **Example(s):** COMMAND: **ATSMRC?<cr>** RESPONSE: **<cr_lf>** **1<cr_lf>**

8.11 Cancel/Idle Command (ATDC)

SD	**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:**
  0 = Classic *Bluetooth* (CB)
  1 = Low Energy (LE)
  2 = General

- **Command:**
  Command_Type = 0 (CB):
  0 = ATDS
  1 = ATDI
  2 = ATDM
  3 = ATRRN
  4 = ATRRS

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

  Command_Type = 2 (General):
  0 = ATCS?
  1 = ATTXT
  2 = ATRXT

Example(s):

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

8.12 Disconnect Command (ATDH)

SD **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 connHandle 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>
```

8.13 Authentication Commands

8.13.1 Passkey Response (ATPKR)

SD

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):

- **Addr_Conn_Handle:** Address of device or connection handle if connected.
- **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<cr_lf>
COMMAND:  ATPKR,ECFE7E000001,123456<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,0,1<cr_lf>
```

8.14 Connection Bridge (ATB)

DM	**BRIDGE** **Function:** Bridges two connection handles. Connection bridging allows a module to act as a serial bridge between two other devices. All data received from Conn_Handle_1 will be passed through to Conn_Handle_2 and all data received from Conn_Handle_2 will be passed through to Conn_Handle_1. **Command Format:** ATB,<Conn_Handle_1>,<Conn_Handle_2> **Command Parameters:** ▪ **Conn_Handle_1:** Connection handle ▪ **Conn_Handle_2:** Connection handle **Example:** COMMAND: **ATB,0,1<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **Note(s):** ▪ *Set both connection handles to 0 to disable bridging.*
	GET BRIDGE **Function:** Gets the current bridge settings. **Command Format:** ATB? **Response Format:** <Conn_Handle_1>,<Conn_Handle_2> **Example:** COMMAND: **ATB?<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **<cr_lf>** **0,1<cr_lf>**

8.15 RF Test Commands

8.15.1 Transmitter Test (ATTXT)

SD	**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,<Channel>,<Mode> **Command Parameter(s):** ▪ **Channel:** Single Mode: RF channel [0-39] (Does not apply to mode 2, set to 0) *Frequency = 2402+(Channel*2) MHz* Dual Mode: RF channel [0-78] *Frequency = 2402+Channel MHz* ▪ **Mode:** Single Mode: [0-2], Dual Mode: [0] 0 = **Transmit** – Transmits using a modulated carrier, at the specified channel. 1 = **Unmodulated Transmi**t – SM Only. Transmits using an unmodulated carrier, at the specified channel. 2 = **Channel Hopping Transmit** – SM Only. Transmits a modulated carrier, transmitting a packet on a different channel every 625µs, continuously cycling linearly through all channels. **Example(s):** COMMAND: **ATTXT,0,0<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **Note(s):** ▪ *The module must be in either the Idle or Testing state before executing an ATTXT. Tests will run continuously until canceled with an ATDC command.* ▪ *Single mode modules will transmit at the maximum output power of 4dB (S2) and 0dB (S3). The D2 will transmit at the power level set by ATSPL.* ▪ *On D2 modules this command is identical to ATFCCT,0,0,<Channel>,15 (continuous wave, PN9 pattern, and maximum power).*

8.15.2 Receiver Test (ATRXT)

SD

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,<Channel>,<Sampling_Period>,<Test_Duration>,<Print_Samples>

Command Parameter(s):

- **Channel:** Single Mode: RF channel [0-39] (Does not apply to mode 2, set to 0)
  *Frequency = 2402+(Channel*2) MHz*

  Dual Mode: RF channel [0-78]
  Frequency = 2402+Channel MHz

- **Sampling_Period:** How often the RSSI will be sampled. [ms]

- **Test_Duration:** How long the test will run for, a value of 0 will run until cancelled. Test will always run till cancelled on Dual Mode modules.[ms]

- **Print_Samples:**
  0 = Do not print any samples, only print the average RSSI at the end of the test.

  1 = SM Only. Print all samples as they are taken in addition to the RSSI at the end of the test.

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,0,1000,3000,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          -042<cr_lf>
          <cr_lf>
          -043<cr_lf>
          <cr_lf>
          -044<cr_lf>
          <cr_lf>
          -043<cr_lf>
          <cr_lf>
          DONE,2,2<cr_lf>
```

Note(s):

- *The module must be in either the Idle or Testing state before executing an ATRXT. The test can be canceled with an ATDC command.*

- *The receiver gain can be set using the ATSPL command.*

8.15.3 Extended Transmitter/Receiver Test (ATTXTE)

DM

EXTENDED TRANSMITTER/RECEIVER TEST

Function: This command is used to start an RF transmitter test with extended options for Classic Bluetooth. It will enable continuous packet transmission/reception.

Command Format: ATTXTE,<Frequency_Mode>,<TX_Frequency>,<RX_Frequency>,<ACL_TX_Packet_Type>,<ACL_TX_Packet_Pattern>,<ACL_Packet_Length>,<Power_Level>,<Whitening>,<PRBS9_Init_Value>

Command Parameter(s):

- **Frequency_Mode:**
  0 = Hopping
  3 = Single Frequency

- **TX_Frequency:**
  0-39 - *Frequency = 2402+(2*i) MHz*
  40-78 - *Frequency = 2403+(2*(i-40)) MHz*

- **RX_Frequency:**
  0-39 - *Frequency = 2402+(2*i) MHz*
  40-78 - *Frequency = 2403+(2*(i-40)) MHz*
  255 - RX Disabled

- **ACL_TX_Packet_Type:**
  0 = DM1
  1 = DH1
  2 = DM3
  3 = DH3
  4 = DM5
  5 = DH5
  6 = 2-DH1
  7 = 2-DH3
  8 = 2-DH5
  9 = 3-DH1
  10 = 3-DH3
  11 = 3-DH5

- **ACL_TX_Packet_Pattern:**
  0 = All 0
  1 = All 1
  2 = 5555 (0101 0101 0101 0101b)
  3 = FOFO (1111 0000 1111 0000b)
  4 = Ordered
  5 = PRBS9 random

- **ACL_Packet_Length:** Length of packet in bytes. Max depends on ACL_TX_Packet_Type:
  DM1: 0-17
  DH1: 0-27
  DM3: 0-121
  DH3: 0-183
  DM5: 0-224
  DH5: 0-339

2-DH1: 0-54
2-DH3: 0-367
2-DH5: 0-679
3-DH1: 0-83
3-DH3: 0-552
5-DH3: 0-1021

- **Power_Level:** 0-15 (15 corresponds to the power level set in ATSPL)

- **Whitening:**
  0 = Enable Packet Whitening
  1 = Disable Packet Whitening

- **PRBS9_Init_Value:** 0-511. Used only with ACL_TX_Packet_Pattern 5 (PRBS9) to initialize data.

Example(s):

```
COMMAND:  ATTXTE,0,0,255,0,2,27,15,1,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *Prior to using the ATTXTE command, do an ATSDB,0,0 and an ATSDBLE,0,0 followed by an ATRST. This will put the board in the idle state by default.*

- *Do an ATRST prior to executing consecutive ATTXTE commands.*

8.15.4 FCC Continuous Transmitter Test (ATFCCT)

DM

FCC CONTINUOUS TRANSMITTER TEST

Function: This command is used to put the module in continuous transmission mode.

Command Format: ATFCCT,<Modulation>,<Test_Pattern>,<TX_Frequency>,
<Power_Level>

Command Parameter(s):

- **Frequency_Mode:**
  0 = Continuous Wave
  1 = GFSK (BR)
  2 = π/4-DQPSK (2-EDR)
  3 = 8DPSK (3-EDR)

- **Test_Pattern:**
  0 = PN9
  1 = PN15
  2 = 5555 (0101 0101 0101 0101b)
  3 = All 1
  4 = All 0
  5 = F0F0 (1111 0000 1111 0000b)
  6 = FF00

- **TX_Frequency:**
  0-39 - *Frequency = 2402+(2*i) MHz*
  40-78 - *Frequency = 2403+(2*(i-40)) MHz*

- **Power_Level:** 0-15 (15 corresponds to the power level set in ATSPL)

Example(s):

```
COMMAND:   ATFCCT,0,0,0,15<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

Note(s):

- *Prior to using the ATFCCT command, do an ATSDB,0,0 and an ATSDBLE,0,0 followed by an ATRST. This will put the board in the idle state by default.*

- *Do an ATRST prior to executing consecutive ATFCCT commands.*

8.15.5 Device Under Test Mode (ATSIGT)

DM **DEVICE UNDER TEST MODE**

Function: This command is used to put the module into the BR/EDR Device Under Test mode described in Volume 3 Part D of the Bluetooth 4.0 Specification. It is used for testing the module over the air using a Bluetooth tester such as the R&S CBT.

Command Format: ATSIGT

Example(s):

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

Note(s):

- *Prior to using the ATSIGT command, do an ATDC to put the module into the idle state.*

- *The output power used during the test can be set using the ATSPL command.*

- *Do an ATRST when done testing prior to executing any other commands.*

8.15.6 RF Observation (ATRFO)

SM **RF OBSERVATION**

Function: This command is used to enable RF observation. When enabled, PIO_8 will go high when the module is transmitting and PIO_9 will go high when the module is receiving.

Command Format: ATRFO,<Enable>

Command Parameter(s):

- **Enable:**
  0 = RF Observation Disabled
  1 = RF Observation Enabled

Example(s):

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

8.16 Utility Commands

8.16.1 Configuration Dump (ATCFG?)

SD

GET CONFIGURATION

Function: Dumps the current configuration.

Command Format: ATCFG?

Example(s):

```
COMMAND:  ATCFG?<cr>
RESPONSE: <cr_lf>
          OK
          <cr_lf>
          ATMT?        = BR-LE4.0-S2<cr_lf>
          ATV?         = 3.5.0.0-S2<cr_lf>
          ATA?         = ECFE7E000000<cr_lf>
          ATSN?        = BlueRadiosFFFFFF,0<cr_lf>
          ATSZ?        = 0,0,0<cr_lf>
          ATSFC?       = 1<cr_lf>
          ATSCL?       = 0<cr_lf>
          ATSRM?       = 0,0<cr_lf>
          ATSDIF?      = 65535,1,1<cr_lf>
          ATSPL?       = 3,1<cr_lf>
          ATSUART?     = 7,0,0,1<cr_lf>
          ATSPIO?,0    = 0,0<cr_lf>
          ATSPIO?,1    = 0,0<cr_lf>
          ATSPIO?,2    = 1,0<cr_lf>
          ATSPIO?,3    = 0,0<cr_lf>
          ATSPIO?,4    = 0,0<cr_lf>
          ATSPIO?,5    = 1,0<cr_lf>
          ATSPIO?,6    = 0,0<cr_lf>
          ATSPIO?,7    = 0,0<cr_lf>
          ATSPIO?,8    = 0,0<cr_lf>
          ATSPIO?,9    = 0,0<cr_lf>
          ATSPIO?,10   = 0,0<cr_lf>
          ATSPIO?,11   = 0,0<cr_lf>
          ATSPIO?,12   = 0,0<cr_lf>
          ATSPIO?,13   = 0,0<cr_lf>
          ATSPIO?,14   = 1,0<cr_lf>
          ATSLED?,0    = 100,65535,0<cr_lf>
          ATSLED?,1    = 010,02000,0<cr_lf>
```

```
ATSSP?       = 043,00000<cr_lf>
ATSPK?       = 0<cr_lf>
ATSAPP?      = 0<cr_lf>
ATSDBLE?     = 1,0,0<cr_lf>
ATSBRSP?     = 1,3,0<cr_lf>
ATSBAS?      = 1,0,100,0<cr_lf>
ATSDSLE?     = 0,0,7<cr_lf>
ATSDSTLE?    = 00000,00160,02048<cr_lf>
ATSDSDLE?,0 = 0201FF020AFF0512FFFFFFFF1107FFFFFFFFFFFFFFFFFFFFF
              FFFFFFFFFF<cr_lf>
ATSDSDLE?,1 = 05FFFFFFFFFF1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
              FFFFFF<cr_lf>
ATSDILE?     = 0,0,1,10,0<cr_lf>
ATSDITLE?    = 10240,00016,00016<cr_lf>
ATSDMTLE?    = 00000,00016,00016<cr_lf>
ATSDCP?      = 00008,00016,000,0400,0<cr_lf>
ATSPLE?      = 2,0,3,0,0<cr_lf>
```

Note(s):

- *This command can only be used when the module is in the Idle State.*

9 Low Energy Commands

9.1 Module Information Commands

9.1.1 Appearance (ATSAPP)

SD

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: http://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicViewer.aspx?u=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>
          00000<cr_lf>
```

9.1.2 Address Type (ATSAT)

SD

SET ADDRESS TYPE

Function: Sets the LE address type the module will use when communicating with other devices over LE.

Supported Address Types:

- **Public Address:** Unique IEEE *Bluetooth* Device Address. (The address returned by ATA?)
- **Static Random Address:** A randomly generated address that does not change until the module is reset or power cycled.
- **Resolvable Private Random Address:** A randomly generated address that is changed every 15 minutes. Paired devices are able to resolve the random address to the public address, allowing them to identify known devices even though their address is changing frequently.

Command Format: ATSAT,<Addr_Type>

Command Parameters:

- **Addr_Type:**
  0 = Public Address - The *Bluetooth* Device Address will be used.
  1 = Static Random Address - A random address will be automatically generated and a new address will be generated after each reset or power cycle.
  3 = Resolvable Private Random Address – A new resolvable random address will be automatically generated every 15 minutes.

Example:

```
COMMAND:  ATSAT,3<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used unless the module is in the Idle State.*
- *If using a random address, the generated address can be read back by using ATSAT? 10ms after setting the address type with ATSAT.*
- *Calling ATSAT,1 multiple times will generate a new static address each time, but a static address should never be changed other than with a reset or power cycle, so this is not recommened.*

GET ADDRESS TYPE

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

Command Format: ATSAT?

Response Format: <Addr_Type>

Example:

```
COMMAND:  ATSAT?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          3,6CE3C8582733<cr_lf>
```

Note(s):
- *ATA? Will still always return the Public Bluetooth Device Address.*

9.2 LE Status Commands

9.2.1 LE State (ATSLE?)

SD

GET STATE LE

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>
```

9.2.2 LE Last Connected Address (ATLCALE?)

SD

GET LAST CONNECTED ADDRESS LE

Function: Gets the last connected LE *Bluetooth* device address. If connected will be the address of the

current connected device.

Command Format: ATLCALE?

Response Format: <Addr>

Response Values:
- **Addr:** Last connected *Bluetooth* device address

Example(s):

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

9.3 LE Default Behavior (ATSDBLE)

SD **SET DEFAULT BEHAVIOR LE**

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>,<Bridge>

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
  2 = CB Connection – DM Only

- **Bridge:** (DM Only, Only Effective If Command = 4/5 and Trigger = 2)
  0 = No Bridging
  1 = Auto Bridge Data Between Incoming CB Connection and Outgoing LE Connection

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.*
- *Single Mode modules won't accept the command with the Bridge parameter included.*
- *When switching between different default behavior settings, the command from the previous default behavior will be cancelled, so a DONE event may print after the OK depending on the current state.*
- *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>
          1,0,0<cr_lf>
```

9.4 GATT Service Configuration

9.4.1 BRSP Service Configuration (ATSBRSP)

The ***n*Blue™** modules use a proprietary GATT profile developed by BlueRadios to stream data; it is not an official *Bluetooth* profile. BlueRadios serial port implementation simplifies the user experience, allowing users to stream data similar to the way the official *Bluetooth* 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 will currently provide a maximum theoretical throughput of 1.3kB/s at the minimum LE connection interval of 7.5ms.

SD	SET BRSP CONFIGURATION

Function: Sets the BRSP service parameters.

Command Format: ATSBRSP,<Service_Enable>,<Features_Mask>,<Security_Mode>

Command Parameter(s):

- **Service_Enable:**
  0 = BRSP Service Disabled
  1 = BRSP Service Enabled

- **Features_Mask:** Specifies what BRSP features are enabled.
  1 = Data Mode Supported
  2 = Remote Command Mode Supported
  4 = OTA Firmware Update Mode Supported – (SM Only. Can only be enabled if an MP45PE external flash is connected to the module, contact BlueRadios for more information.)

- **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)

Example(s):

1. Enables the BRSP service with support for Data Mode and Remote Command Mode.

```
COMMAND:  ATSBRSP,1,3,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *To set Security_Mode to 2, the IO_Capabilities of ATSPLE must not be set to 3 (No Input No Output.)*

- *In order to change the Service_Enable parameter, the module must be disconnected and not paired with any other devices.*

- *It is recommended to do a reset after changing the Service_Enable parameter to keep the service handles continuous.*

GET BRSP CONFIGURATION

Function: Gets the BRSP service parameters.

Command Format: ATSBRSP?

Response Format: <Service_Enable>,<Features_Mask>,<Security_Mode>

Example(s):

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

9.4.2 Battery Service Configuration (ATSBAS)

SD

SET BAS CONFIGURATION

Function: Configures the Battery Service (BAS).

The Battery Service specification can be found here:
https://www.bluetooth.org/en-us/specification/adopted-specifications

Command Format: ATSBAS,<Service_Enable>,<Mode>,<Level_Update_Interval>,<Critical_Level>

Command Parameter(s):

- **Service_Enable:**
  0 = BAS Service Disabled
  1 = BAS Service Enabled

- **Mode:**
  0 = Manual Mode – Level is set manually by the user. Use if not powering the module directly from a 3.0V coin cell and battery level is being monitored by an external controller.

  If BAS is enabled in Manual mode the BATT event will fire when a client device has registered/deregistered to receive notifications/indications from the Battery Level characteristic of the BAS service. When a client registers for notifications/indications the battery level should be updated using ATSBAS if it changes, which will cause the service to notify the client of the change in level.

  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.

- **Level_Update_Interval:**
  Mode = 0 [0-100%]
  Mode = 1 [1-687194 s]

- **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 with the level set to 100%.

```
COMMAND:  ATSBAS,1,0,100<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):

- *In order to change the Service_Enable parameter, the module must be disconnected and not paired with any other devices.*
- *It is recommended to do a reset after changing the Service_Enable parameter to keep the service handles continuous.*

GET BAS CONFIGURATION

Function: Gets the BAS service parameters.

Command Format: ATSBAS?

Response Format: <Service_Enable>,<Mode>,<Level_Update_Interval>

Example(s):

```
COMMAND:  ATSBAS?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1,0,100<cr_lf>
```

9.4.3 Device Information Service (ATSDIS)

SM PO

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 peripheral devices should implement the Device Information Service, so it cannot be disabled. All characteristics are optional though, so individual characteristics can be disabled if not needed. When disabled the characteristic will not be removed, but will instead return a null value.

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 and 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 contacting a set of values that shall be sued to create a device ID avalue that is unique for this device. Included in the characterstic are a Vedor 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-LE4.0-S2"
  - 2 = Serial Number String
    Default: Enabled, last six digits of IEEE addres ex: "000001"
  - 3 = Hardware Revision String
    Default: Disabled, "Hardware Revision"
  - 4 = Firmware Revision String
    Default: Enabled, "3.5.0.0-PO-S2"
  - 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,BlueRadios,Inc.<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>
```

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>
```

9.4.4 Heart Rate Service Configuration / Control (ATSHRS / ATHRM)

SM PO

SET HRS CONFIGURATION

Function: Configures the Heart Rate Service (HRS).

Enabling the Heart Rate Service will enable the Heart Rate Service Event which will be sent when a client device has registered/deregistered to receive notifications from the Heart Rate Measurement characteristic of the HRS service.

The Heart Rate Service specification can be found here:
https://www.bluetooth.org/en-us/specification/adopted-specifications

Command Format: ATSHRS,<Service_Enable>,<Sensor_Location>,
<Contact_Status_Support>,<Energy_Expended_Support>

Command Parameter(s):

- **Service_Enable:**
  0 = HRS Service Disabled
  1 = HRS Service Enabled

- **Sensor_Location:**
  0 = Other
  1 = Chest
  2 = Wrist
  3 = Finger
  4 = Hand
  5 = Ear Lobe
  6 = Foot

- **Contact_Status_Support:**
  0 = Sensor contact status is not supported
  1 = Sensor contract status is supported

- **Energy_Expended_Support:**
  0 = Energy expended is not supported
  1 = Energy expended is supported

Example(s):

1. Enables the HRS service for a chest sensor with contact status and expended energy support.

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

Note(s):

- *In order to change the Service_Enable parameter, the module must be disconnected and not paired with any other devices.*

- *It is recommended to do a reset after changing the Service_Enable parameter to keep the service handles continuous.*

GET HRS CONFIGURATION

Function: Gets the HRS service configuration.

Command Format: ATSHRS?

Response Format: <Service_Enable>,<Sensor_Location>,<Contact_Status_Support>,
<Energy_Expended_Support>

Example(s):

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

HEART RATE MEASUREMENT

Function: If a client is registered for notifications on the Heart Rate Measurement characteristic (see the Heart Rate Service Event), this command will notify the client of a new heart rate measurement. In typical applications the client should be notified with a new heart rate value approximately 1 time per second. If supported, the energy expended should be updated approximately 1 time per every 10 seconds.

Command Format: ATHRM,<Heart_Rate_Value>,<Sensor_Contact>,<New_Energy_Expended>

Command Parameter(s):

- **Heart_Rate_Value:** Integer value from 0 – 65535 [bpm (beats per minute)].
- **Sensor_Contact:**
  0 = Sensor contact not detected.
  1 = Sensor contact detected.
- **New_Energy_Expended:** Integer value from 0 – 65535 [joules]. Energy expended since the last time this parameter was updated. Internally a sum of all energy expended will be kept and this value will be added to the sum. The sum is sent with the heart rate measurement only if Energy_Expended is greater than 0. The sum can only be reset by the client as per the Heart Rate Profile.

Example(s):

1. Sends a new heart rate measurement of 150 bpm with sensor contact detected and 5 joules of new energy expended.

```
COMMAND:  ATHRM,150,1,5<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

9.4.5 Proximity Profile Services Configuration (ATSPXP)

SM PO

SET PXP CONFIGURATION

Function: Configures the Proximity Profile Services – Link Loss, Immediate Alert and TX Power. These services can be used to implement the Proximity Profile.

Enabling either the Link Loss or Immediate Alert Service will enable the Proximity Profile Service Event which will be sent when the Alert Level of either service changes.

The specifications for the Proximity Profile, as well as the Link Loss, Immediate Alert and TX Power Services can be found here:
https://www.bluetooth.org/en-us/specification/adopted-specifications

Command Format: ATSPXP,<LLS_Enable>,<IAS_Enable>,<TPS_Enable>

Command Parameter(s):

- **LLS_Enable:**
  0 = Link Loss Service Disabled
  1 = Link Loss Service Enabled
- **IAS_Enable:**
  0 = Immediate Alert Service Disabled

1 = Immediate Alert Service Enabled

- **TPS_Enable:**
  0 = TX Power Service Disabled
  1 = TX Power Service Enabled

Example(s):

1. Enables the Link Loss, Immediate Alert and TX Power Services.

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

Note(s):

- *In order to change any of the parameters, the module must be disconnected and not paired with any other devices.*
- *It is recommended to do a reset after changing any of the parameters to keep the service handles continuous.*

GET PXP PARAMETERS

Function: Gets the Proximity Profile Services configuration.

Command Format: ATSPXP?

Response Format: <LLS_Enable>,<IAS_Enable>,<TPS_Enable>

Example(s):

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

9.5 Advertising Commands

9.5.1 Advertise (ATDSLE)

SD **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.

Command Format: ATDSLE

Example(s):

```
COMMAND:  ATDSLE<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.*

9.5.2 Advertise Direct (ATDSDLE)

SD **DIAL AS SLAVE DIRECT (CONNECTABLE DIRECT ADVERTISING)**

Function: This command places the module in the connectable direct advertising mode, 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. Connectable direct advertising will automatically timeout after 1.28s regardless of the ATSDSTLE Ad_Timeout setting.

Command Format: ATDSDLE,<Addr>,<BRSP_Mode>,<Addr_Type>,<Duty_Cycle>

Command Parameter(s):

- **Addr:** Master address for connectable direct advertising.

- **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.

- **Address_Type:**
  0 = Public Address
  1 = Static Address
  2 = Non-Resolvable Private Address
  3 = Resolvable Private Address

- **Duty_Cycle:**
  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.
  1 = High Duty Cycle – A power intensive directed advertising mode for cases in which fast connection setup is essential.

Example(s):

```
COMMAND:  ATDSDLE,ECFE7E000000<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          CONNECT,0,1,0,ECFE7E000000<cr_lf>
```

Note(s):

- *This command overrides the ATSDSLE Ad_Type setting to connectable direct advertisement.*

- *If BRSP_Mode is not set to 0 and the master the module is connecting to was also set to connect with a BRSP_Mode of 1, both modules will attempt to initiate BRSP which will lead to a BRSP,05*

event.

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

9.5.3 Advertising Configuration (ATSDSLE)

SD

SET ADVERTISING

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

Command Format: ATSDSLE,<White_List_Filter>,<Ad_Type>,<Ad_Channel_Map>

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 indirect advertisement (Connectable + Scannable)
  2 = Scannable indirect advertisement (Scannable Only)
  3 = Non-connectable indirect advertisement (Neither Connectable or Scannable)

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

Example(s):

```
COMMAND:  ATSDSLE,0,0,7<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>

Example(s):

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

9.5.4 Advertising Timing Configuration (ATSDSTLE)

SD

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].
  0 = No Timeout/General Discoverable Mode / !0 = Timeout/Limited 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 160 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>
```

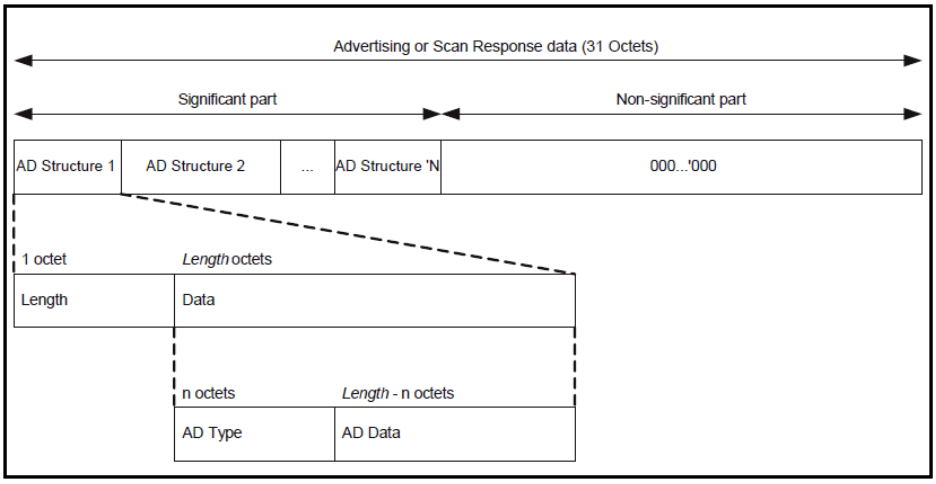
```
<cr_lf>
00000,00160,02048<cr_lf>
```

9.5.5 Advertising/Scan Response Data (ATSDSDLE)

SD **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 Bluetooth 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 if 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

- **Data:** Data is compromised 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: 0201FF020AFF0512FFFFFFFF1107FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

  This default data will auto populate to the same data used in previous version of AT.s:
  Flags-TX Power-Slave Connection Interval Range-128-bit UUID Service List
  020106020A0405120800100011077960 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 scan data will auto populate to the same data used in previous version of AT.s:
  BlueRadios Manufacturer Specific Data – Name
  05FF850000001109426C7565526164696F73303030303031
  05FF85000000
  For PAN172X builds the manufacturer data will not be included and the default will be:
  1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

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 in order 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.*

GET ADVERTISING DATA

Function: Gets the ATSDSDLE command settings.

Command Format: ATSDSDLE?,<Data_Type>

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>
```

9.6 LE Discovery Commands

- **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.

9.6.1 LE Discovery (ATDILE)

SD **DISCOVERY LE**

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

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<c _lf>
    EVENT:      <cr_lf>
                DISCOVERY,2,ECFE7E000001,0,-045,4,020106-020A04-051208000800-
                1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
    EVENT:      <cr_lf>
                DISCOVERY,6,ECFE7E000001,0,-045,1,
                1109426C7565526164696F73303030303031<cr_lf>
    EVENT:      <cr_lf>
                DISCOVERY,3,ECFE7E000002,0,-045,4,020106-020A04-051208000800-
                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.*

9.6.2 LE Discovery Configuration (ATSDILE)

SD **SET DISCOVERY LE**

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>

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 [0-10]
  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.
  1-10 = 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.)

Example(s):

```
COMMAND:  ATSDILE,0,0,1,10,0<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>

Example(s):

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

9.6.3 LE Discovery Timing Configuration (ATSDITLE)

SD

SET DISCOVERY TIMING LE

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 61440 [ms]. A value of 0 can be used on the Dual Mode to continuously discover.
  Default: 10240 (10240ms)

- **Interval:** Integer value from 4 to 16384 [.625ms].
  Default: 16 (10ms)

- **Window:** Integer value from 4 to 16384 [.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>
          10240,00016,00016<cr_lf>
```

9.7 LE Connect Commands

9.7.1 LE Connect (ATDMLE)

SD **DIAL AS MASTER LE (CONNECT)**

Function: This command initiates a connection to a slave device. If the command is an accepted an **"OK"** will be sent back immediately. Don't 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 *Bluetooth* connection is established.

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

Command Parameter(s):

- **Addr:** Slave device address. 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.

- **Address Type:**
  0 = Public Address
  1 = Static Address
  2 = Non-Resolvable Private Address
  3 = Resolvable Private Address

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:

```
COMMAND:  ATDMLE,ECFE7E000001,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<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,ECFE7E000001,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<cr_lf>
          <cr_lf>
          BRSP,0,1<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<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.*

9.7.2 LE Connect Last (ATDMLLE)

SD	**DIAL AS MASTER LAST LE (CONNECT LAST)**

Function: Initiates a connection to the last connected slave device.

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>
          <cr_lf>
          CONNECT,0,1,0,ECFE7E000001<cr_lf>
          <cr_lf>
          BRSP,1<cr_lf>
```

Note(s):

- *To verify the last address that will be connected to use the ATLCALE? Command.*
- *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.*

9.7.3 LE Connect Timing Configuration (ATSDMTLE)

SD **SET CONNECT TIMING LE**

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 0 to 61440 [ms]. (0 = No Timeout)
- **Interval:** Integer value from 4 to 16384 [.625ms].
  Default: 16 (10ms)

- **Window:** Integer value from 4 to 16384 [.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>
          00000,00016,00016<cr_lf>
```

9.8 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))*

9.8.1 Default Connection Parameters (ATSDCP)

SD **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 don't 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 don't 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<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>
          0008,00016,0000,0400,0<cr_lf>
```

9.8.2 Current Connection Parameters (ATSCCP)

SD **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>
          0008,0000,0400<cr_lf>
```

9.9 LE Pairing Commands

9.9.1 LE Pair Command (ATPLE)

SD

PAIR DEVICE LE

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 of device or connection handle if connected.
- **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,ECFE7E000001,1<cr_lf>
```

Note(s):

- *Up to 8 devices can be paired at a time on single mode modules.*
- *Up to 30 (CB and LE combined) devices can be paired at a time on dual mode modules.*

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 isn't 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-ECFE7E000002<cr_lf>
```

9.9.2 LE Pairing Configuration (ATSPLE)

SD	**SET PAIRING PARAMETERS LE** **Function:** This command is used to configure LE pairing. **Command Format:** ATSPLE,<Request_Handling>,<Authentication>,<IO_Capabilities>,

<Sync_White_List>,<Disconnect_Unpaired>

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 in order 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,03 if this option is enabled.

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

Example(s):

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

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**

GET PAIRING LE

Function: Gets the ATSPLE settings.

Command Format: ATSPLE?

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

Example(s):

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

9.9.3 LE Unpair Device (ATUPLE)

SD	UN PAIR DEVICE LE

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

Command Format: ATUPLE,<Addr>

Command Parameter(s):

- **Addr:** Device address

Example(s):

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

9.9.4 LE Clear Pair List (ATCPLE)

SD	CLEAR PAIR LIST LE

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>
```

9.9.5 Fixed Passkey (ATSPK)

SD	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>
```

9.10 White List Commands

9.10.1 White List Device (ATSWL)

SD	**SET 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 isn't used.

Command Format: ATSWL,<Addr>,<Addr_Type>

Command Parameter(s):

- **Addr:** Device address
- **Addr_Type:**
  0 = Public Device Address
  1 = Random Device Address

Example(s):

```
COMMAND:  ATSWL,ECFE7E000001<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *Up to 8 devices can be added to the whitelist on single mode modules, 30 on dual modes.*
- *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 doesn't 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-ECFE7E000002<cr_lf>
```

9.10.2 Un White List Device (ATUWL)

SD	**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

Example(s):

```
COMMAND:  ATUWL,ECFE7E000001<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.*

9.10.3 Clear White List (ATCWL)

SD	**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.*

9.11 GATT Commands

The GATT 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.

9.11.1 Discover All Primary Services (ATGDPS)

SD **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 Bluetooth defined service is discovered, the service acronym will be returned instead. All other UUIDs will still be returned as Hex UUIDs.

Example(s):

1. ATGDPS is issued for connection handle 0 and 4 primary services are discovered: GAP, GATT, BAS (Battery) and BRSP. When all of the primary services have been discovered a GATT_DONE event it triggered:

```
COMMAND:  ATGDPS,0cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00001,00011,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00012,00014,1801<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00016,00019,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00015,65535,DA2B84F1627948DEBDC0AFBEA0226079<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<cr_lf>
```

2. Same as example 1, but using the Sevice Acronym String format:

```
COMMAND:  ATGDPS,5cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00001,00011,GAP<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00012,00014,GATT<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00016,00019,BAS<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00015,65535,DA2B84F1627948DEBDC0AFBEA0226079<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<cr_lf>
```

9.11.2 Discover Primary Services By UUID (ATGDPSU)

SD	**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 11.

```
COMMAND:  ATGDPSU,0,1800<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00001,00011,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<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,0,00015,65535,DA2B84F1627948DEBDC0AFBEA0226079
          <cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<cr_lf>
```

9.11.3 Discover All Characteristics (ATGDC)

SD	**GATT DISCOVER ALL CHARACTERISTICS**

Function: This command is used to discover all the characteristics on a server or all of the characteristics within a specific attribute handle range. If Start_Att_Handle is specified, End_Att_Handle must also be specified.

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 11 (GAP Service), and 5 characteristics are found.

```
COMMAND:  ATGDC,0,1,11<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,00003,02,2A00<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,00005,02,2A01<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,00007,0A,2A02<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,00009,0A,2A03<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,00011,02,2A04<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,00<cr_lf>
```

Note(s):

- *If Start_Att_Handle is specified, End_Att_Handle must also be specified.*

9.11.4 Discover Characteristics By UUID (ATGDCU)

SD

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. If Start_Att_Handle is specified, End_Att_Handle must also be specified.

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,0,2A00,1,11<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DC,0,00003,02,2A00<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,00<cr_lf>
```

Note(s):

- *If Start_Att_Handle is specified, End_Att_Handle must also be specified.*

9.11.5 Characteristic Descriptor Discovery (ATGDCD)

SD

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 isn't specified correctly attribute types other than descriptors will be returned.

If a range isn't 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):

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 16 and an end handle of 19:

```
COMMAND:  ATGDPSU,0,180F<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,00016,00019,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<cr_lf>
```

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

```
COMMAND:  ATGDC,0,16,19<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
```

```
                GATT_DC,0,00018,12,2A19<cr_lf>
EVENT:          <cr_lf>
                GATT_DONE,0,2,00<cr_lf>
```

Last, ATGDCD is used to find the Battery Level characteristic descriptors. The Start_Att_Handle is set to 19, 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,0,19,19<cr>
RESPONSE:   <cr_lf>
            OK<cr_lf>
EVENT:      <cr_lf>
            GATT_DCD,0,00019,2902<cr_lf>
EVENT:      <cr_lf>
            GATT_DONE,0,3,00<cr_lf>
```

Note(s):

- *If Start_Att_Handle is specified, End_Att_Handle must also be specified.*

9.11.6 Characteristic Read (ATGR)

SD

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 only read values up to ATT_MTU-1 bytes in length. ATT_MTU is the Attribute Protocol Maximum Transmission Unit, which on Single Mode modules is 23, so the maximum read length is 22 bytes.

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 *Bluetooth* Device Name, formatted in Hex by default.

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

```
EVENT:    <cr_lf>
          GATT_VAL,0,00003,0,16,426C7565526164696F73303030303031<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,4,00<cr_lf>
```

9.11.7 Characteristic Read Long (ATGRL)

SD

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 ATT_MTU-1 bytes in length. ATT_MTU is the Attribute Protocol Maximum Transmission Unit, which on Single Mode modules is 23, so the maximum read length is 22 bytes.

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 secod 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,0,00054,000,22,
          11223344556677889900112233445566778899001122<cr_lf>
EVENT:    <cr_lf>
          GATT_LVAL,0,00054,022,11,33445566778899001122 33<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,6,00<cr_lf>
```

9.11.8 Characteristic Read By UUID (ATGRU)

SD

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 only read values up to ATT_MTU-1 bytes in length. ATT_MTU is the Attribute Protocol Maximum Transmission Unit, which on Single Mode modules is 23, so the maximum read length is 22 bytes.

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 *Bluetooth* Device Name, formatted in Hex by default.

```
COMMAND:  ATGRU,0,2A00<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_VAL,0,00003,0,16,426C7565526164696F73303030303031<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,4,00<cr_lf>
```

2. Reading the remote *Bluetooth* Device Name, formatted as an ASCII string.

```
COMMAND:  ATGRU,0,2A00,5<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_VAL,0,00003,0,16,BlueRadios000001<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,4,00<cr_lf>
```

9.11.9 Characteristic Write (ATGW)

SD

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.

This command could previously only write values up to ATT_MTU-3 bytes in length. ATT_MTU is the Attribute Protocol Maximum Transmission Unit, which on Single Mode modules is 23, so the maximum write length was 20 bytes. Now if more than 20 bytes are passed in the module will perform a long write and internally break the packet up into separate packets of 18 bytes. The amount of data that can be sent is currently limited by the module's UART RX buffer. If using the Hex Value_Format up to 36 bytes can be sent and if using the ASCII string format up to 72 bytes can be sent.

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,0,5,00<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,0,5,00<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,0,5,00<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,0,5,00<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,0,5,00<cr_lf>
```

9.11.10 Characteristic Write No Response (ATGWN)

SD

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 only write values up to ATT_MTU-3 bytes in length. ATT_MTU is the Attribute Protocol Maximum Transmission Unit, which on Single Mode modules is 23, so the maximum write length is 20 bytes.

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>
```

9.11.11 Characteristic Write Prepared (ATGWP/ATGWPE)

SD

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 Bluetooth 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<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,7,00<cr_lf>
COMMAND:  ATGWP,0,99,5,901234567890123456<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
```

	GATT_DONE,0,7,00<cr_lf> COMMAND: ATGWP,0,99,5,7890<cr> RESPONSE: <cr_lf> OK<cr_lf> EVENT: <cr_lf> GATT_DONE,0,7,00<cr_lf> COMMAND: ATGWPE,0<cr> RESPONSE: <cr_lf> OK<cr_lf> EVENT: <cr_lf> GATT_DONE,0,8,00<cr_lf>
SD	**GATT WRITE PREPARED EXECUTE** **Function:** This command is used to finalize a prepared write. **Command Format:** ATGWPE,<Conn_Handle> **Command Parameter(s):** ▪ **Conn_Handle:** Connection handle. **Example(s):** See ATGWP example above.

10 Classic Bluetooth Commands

10.1 Important Notes

- The CB Commands are only supported on **Dual Mode *n*Blue™ Modules**.

10.2 Module Information Commands

10.2.1 Class of Device (ATSCOD)

DM	**SET COD** **Function:** Sets the COD. **Command Format:** ATSCOD,<COD> **Command Parameters:** ▪ **COD*:*** 6 hex characters, based on the *Bluetooth* COD specification names published and maintained by the *Bluetooth* SIG. Default: 000000 (Undefined) **Example:** COMMAND: **ATSCOD,000000<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>**
	GET COD **Function:** Gets the COD. **Command Format:** ATSCOD? **Response Format:** <COD> **Example:** COMMAND: **ATSCOD?<cr>** RESPONSE: **<cr_lf>** **000000<cr_lf>**

10.2.2 Local Service Name / UUID (ATSSN)

DM	**SET SERVICE NAME / UUID** **Function:** Sets the name and UUID of the module's local service. **Command Format:** ATSSN,<Service_Name>,<Service_UUID> **Command Parameter(s):** ▪ **Service_Name:** 1-16 alphanumeric characters Default: COM1

- **Service_UUID:** UUID of service. UUID can be 16 or 128 bit (4 or 32 characters). Default: 1101 (SPP – Serial Port Profile)

Example(s):

```
COMMAND:  ATSSN,COM1,1101<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET SERVICE NAME

Function: Gets the name and UUID of the module's local service.

Command Format: ATSSN?

Response Format: <Service_Name>,<Service_UUID>

Example(s):

```
COMMAND:  ATSSN?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          COM1,1101<cr_lf>
```

10.3 CB Status Commands

10.3.1 CB State (ATS?)

DM

GET STATE CB

Function: Gets the current CB state of the module.

Command Format: ATS?

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

Response Values:

- **Idle_Testing:**
  0 = Not Idle
  1 = Idle
  2 = Testing
- **Scanning:**
  0 = Not Scanning
  1 = Scanning (ATDS)
- **Discovering:**
  0 = Not Discovering
  1 = Discovering (ATDI)
- **Connecting:**
  0 = Not Connecting

1 = Connecting (ATDM)
2 = ATRRN/ATRRS

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

Example(s):

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

10.3.2 CB Last Connected Address (ATLCA?)

DM **GET LAST CONNECTED ADDRESS CB**

Function: Gets the last connected device address, which if connected will be the address of the current connected device.

Command Format: ATLCA?

Response Format: <Addr>

Response Values:

- **Addr:** Slave device address

Example(s):

```
COMMAND:  ATLCA?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          ECFE7E000001<cr_lf>
```

10.3.3 Link Quality (ATLQ?)

DM **GET LINK QUALITY**

Function: Gets the link quality of a connection. Link Quality is a Hex value from 0-255, which represents the quality of the link between two Bluetooth devices. The higher the value, the better the link quality is.

Command Format: ATLQ?,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle of connection to read link quality from.

Response Format: <Link_Quality>

Response Values:
- **Link_Quality:** 0 – 255

Example(s):

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

Note(s):
- *The module must be connected to read the link quality.*

10.4 CB Default Behavior Commands

10.4.1 CB Default Behavior (ATSDB)

DM **SET DEFAULT BEHAVIOR CB**

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

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

Command Parameter(s):
- **Command:**
  0 = No Command (Idle)
  1 = ATDS (Scanning)
  2 = ATDI (Discovering) **(Cannot be used with Trigger = 1)**
  3 = ATDM,<ATSDBA_Addr>,1101 (Connecting to ATSDBA address SPP service)
- **Trigger: (Only Effective If Command != 0)**
  0 = Idle
  1 = Idle UART Data **–** Cannot be used if ATSDB Trigger = 1
  2 = LE Connection
- **Bridge: (Only Effective If Command = 3 and Trigger = 2)**
  0 = No Bridging
  1 = Auto Bridge Data Between Incoming LE Connection and Outgoing CB Connection

Factory Default: Action = 1, Trigger = 0, Bridge = 0

Example(s):

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

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

```
DONE,1,1<cr_lf>
```

Note(s):

- *When switching between different default behavior settings, the command from the previous default behavior will be cancelled, so a DONE event may print after the OK depending on the current state.*
- *A default behavior can be temporarily disabled by executing an ATDC command or by toggling PIO_4.*

GET DEFAULT BEHAVIOR CB

Function: Gets the module's CB default behavior.

Command Format: ATSDB?

Response Format: <Command>,<Trigger>

Example(s):

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

10.4.2 CB Default Behavior Address (ATSDBA)

DM

SET DEFAULT BEHAVIOR ADDRESS

Function: This command is used to set the address used by the ATDM default behaviors.

Command Format: ATSDBA,<Addr>

Command Parameter(s):

- **Addr:** Device address

Example(s):

```
COMMAND:  ATSDBA,ECFE7E000001<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

GET DEFAULT BEHAVIOR ADDRESS

Function: This command is used to get the address used by the ATDM default behaviors.

Command Format: ATSDBA?

Response Format: <Addr>

Example(s):

```
COMMAND:  ATSDBA?<cr>
RESPONSE: <cr_lf>
```

```
OK<cr_lf>
<cr_lf>
ECFE7E000001<cr_lf>
```

10.5 Scanning Commands

10.5.1 Scan Command (ATDS)

DM

DIAL AS SLAVE (SCAN)

Function: This command places the module in the scanning state where it can be connectable and/or discoverable.

Command Format: ATDS

Example(s):

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

10.5.2 Scan Configuration (ATSDS)

DM

SET SCAN

Function: This command is used to configure the ATDS command settings.

Command Format: ATSDS,<Connectable>,<Discoverable>

Command Parameter(s):

- **Connectable:**
  0 = Not connectable
  1 = Connectable (Page Scan)

- **Discoverable:**
  0 = Not discoverable
  1 = Discoverable (Inquiry Scan)

Example(s):

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

Note(s):

- *This command cannot be used when the module is in the scanning state.*

GET SCAN

Function: Gets the ATDS command settings.

Command Format: ATSDS?

Response Format: <Connectable>,<Discoverable>

Example(s):

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

10.5.3 Scan Timing Configuration (ATSDST)

DM

SET SCAN TIMING

Function: Sets the scan timing parameters.

Command Format: ATSDST,<Timeout>,<Connectable_Interval>,<Connectable_Window>,
<Discoverable_Interval>,<Discoverable_Window>

Command Parameter(s):

- **Timeout:** Integer value of 0 or from 30720 to 61440 [ms]. (0 = No Timeout, General Discoverable Mode, 30720-61440 = Limited Discoverable Mode)
- **Connectable_Interval:** Page Scan Interval, Integer value from 18 to 4096 [.625ms]. Default: 2048 (1280ms)
- **Connectable_Window:** Page Scan Window, Integer value from 17 to 4096 [.625ms]. Default: 18 (11.25ms)
- **Discoverable_Interval:** Inquiry Scan Interval, Integer value from 18 to 4096 [.625ms]. Default: 4096 (2560ms)
- **Discoverable_Window:** Inquiry Scan Window, Integer value from 17 to 4096 [.625ms]. Default: 18 (11.25ms)

Example(s):

```
COMMAND:  ATSDST,0,1024,512,1024,512<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

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

GET SCAN TIMING

Function: Gets the scan timing parameters.

Command Format: ATSDST?

Response Format: <Timeout>,<Connectable_Interval>,<Connectable_Window>,

<Discoverable_Interval>,<Discoverable_Window>

Example(s):

```
COMMAND:  ATSDST?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,1024,0512,1024,0512<cr_lf>
```

10.6 CB Discovery Commands

10.6.1 CB Discovery (ATDI)

DM **DISCOVERY CB**

Function: This command is used to discover discoverable CB devices.

Command Format: ATDI

Example(s):

```
COMMAND:  ATDI<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,0,ECFE7E000000,000000,-034,0,0
          <cr_lf>
          <cr_lf>
          DONE,1,2<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 a timeout has occurred while searching for the number of devices specified.*

10.6.2 CB Discovery Configuration (ATSDI)

DM **SET DISCOVERY CB**

Function: This command is used to configure the ATDI command settings.

Command Format: ATSDI,< Limited_Filter >,<Max_Devices>

Command Parameter(s):

- **Limited_Filter:**
  0 = Disabled (Discover all discoverable devices)
  1 = Discover only limited discoverable devices

- **Max_Devices:** Maximum number of devices to discover [1-32]
  Default: 32

Example(s):

```
COMMAND:  ATSDI,0,20<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

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

GET DISCOVERY CB

Function: Gets the ATSDI command settings.

Command Format: ATSDI?

Response Format: <Limited_Filter >,<Max_Devices>

Example(s):

```
COMMAND:  ATSDI?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,20<cr_lf>
```

10.6.3 CB Discovery Timing Configuration (ATSDIT)

DM

SET DISCOVERY TIMING CB

Function: Sets the module's CB discovery timing parameters.

Command Format: ATSDIT,<Timeout>

Command Parameter(s):

- **Timeout:** Integer value from 1 to 48 [1.28s].
  Default: 8 (10.25s)

Example(s):

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

Note(s):

- *If the module is already in the advertising state, advertising will need to be restarted by putting the module in the idle state using ATDC and then executing ATDS to use the new parameters.*
- *This command cannot be used when the module is in the Discovering State.*

GET DISCOVERY TIMING CB

Function: Gets the module's discovery timing parameters.

Command Format: ATSDIT?

Response Format: <Timeout>

Example(s):

```
COMMAND:  ATSDIT?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          8<cr_lf>
```

10.7 CB Connect Commands

10.7.1 CB Connect (ATDM)

DM **DIAL AS MASTER CB (CONNECT)**

Function: This command initiates a CB connection using the service UUID.

Command Format: ATDM,<Addr>,<Service_UUID_RFComm_Channel>

Command Parameter(s):

- **Addr:** Slave device address.
- **Service_UUID_RFComm_Channel:** UUID or RFComm channel of service to connect to. UUID can be 16, 32, or 128 bit (4, 8, or 32 characters). RFComm channel can be 1-31. 1101 (16-Bit UUID for Serial Port Profile)

Example(s):

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

COMMAND:  ATDM,ECFE7E000001,1101<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          CONNECT,0,0,0,ECFE7E000001<cr_lf>

COMMAND:  ATDM,ECFE7E000001,1101<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          DONE,1,2<cr_lf>
```

Note(s):

- *If the remote Slave device is not present, a DONE event will occur after the connect timeout.*

10.7.2 CB Connect Last (ATDML)

DM

DIAL AS MASTER LAST CB (CONNECT LAST)

Function: Initiates a CB connection to the last connected CB device.

Command Format: ATDML,<Service_UUID>

Command Parameter(s):

- **Service_UUID:** UUID of service to connect to. UUID can be 16, 32 or 128 bit (4, 8 or 32 characters).
- 1101 (Serial Port Profile)

Example(s):

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

Note(s):

- *To verify the last address use the ATLCA? Command.*

10.7.3 CB Connect Timing Configuration (ATSDMT)

DM

SET CONNECT TIMING CB

Function: Sets the CB connection initiation timing parameters.

Command Format: ATSDMT,<Timeout>

Command Parameter(s):

- **Timeout:** Integer value from 1 to 65535 [.625ms].
  Default: 8192 (5120ms)

Example(s):

```
COMMAND:  ATSDMT,64000<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *This command cannot be used when the module is in the Connecting State.*
- *The timeout in this command is also the timeout for ATP, ATRRN, and ATRRS.*

GET CONNECT TIMING CB

Function: Gets the CB connection initiation timing parameters.

Command Format: ATSDMT?

	Response Format: <Timeout> **Example(s):** COMMAND: **ATSDMT?<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **<cr_lf>** **64000<cr_lf>**

10.8 Link Supervision Timeout (ATSLST)

DM	**SET LINK SUPERVISION TIMEOUT** **Function:** Sets the link supervision timeout. **Command Format:** ATSLST,<Timeout> **Command Parameter(s):** ▪ **Timeout:** Integer value from 1 to 65535 [.625ms]. Default: 6400 (4000ms) **Example(s):** COMMAND: **ATSLST,6400<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>**
	GET LINK SUPERVISION TIMEOUT **Function:** Gets the link supervision timeout. **Command Format:** ATSLST? **Response Format:** <Timeout> **Example(s):** COMMAND: **ATSLST?<cr>** RESPONSE: **<cr_lf>** **OK<cr_lf>** **<cr_lf>** **06400<cr_lf>**

10.9 CB Pairing Commands

10.9.1 CB Pair (ATP)

DM	**PAIR DEVICE CB** **Function:** This command is used to initiate CB 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: ATP,<Addr_Conn_Handle>,<Accept_Request>

Command Parameter(s):

- **Addr_Conn_Handle:** Address of device or connection handle if connected.
- **Accept_Request: (Only needed when responding to a PAIR_REQ event.)**
  0 = Reject pairing request.
  1 = Accept pairing request.

Example(s):

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

Note(s):

- *Up to 30 (CB and LE combined) devices can be paired at a time.*

GET PAIRED DEVICES CB

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

Command Format: ATP?

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):

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

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

10.9.2 CB Pairing Configuration (ATSP)

DM

SET PAIRING CB

Function: This command is used to configure CB pairing.

Command Format: ATSP,<Request_Handling>,<Authentication>,<IO_Capabilities>

Command Parameter(s):

- **Request_Handling:**
  0 = Reject all pairing requests.
  1 = Prompt for all pairing requests with PAIR_REQ event.
  2 = 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.

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

Example(s):

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

Note(s):

- *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 in order 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.*

- *Legacy Android applications cannot connect to the module if the ATSP IO_Capabilities are set to No Input or Output. To guarantee that the module can connect to any Android application, set the IO_Capabilities in ATSP to Display Only or Display With Yes/No Buttons.*

IO Capability/Authentication Mapping:

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

Display With Yes/No Buttons	Just Works: **No Auth**	Numeric Comparison: Both display, both confirm **Authenticated**	Passkey Entry: responder displays, initiator inputs **Authenticated**	Just Works: **No Auth**
Numeric Keyboard Only	Passkey Entry: initiator displays, responder inputs **Authenticated**	Passkey Entry: initiator displays, responder inputs **Authenticated**	Passkey Entry: initiator and responder input **Authenticated**	Just Works: **No Auth**
No Input or Output	Just Works: **No Auth**	Just Works: **No Auth**	Just Works: No **Auth**	Just Works: **No Auth**

GET PAIRING PARAMETERS CB

Function: Gets the ATSP settings.

Command Format: ATSP?

Response Format: <Request_Handling>,<Authentication>,<IO_Capabilities>

Example(s):

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

10.9.3 Unpair Device CB (ATUP)

DM

UN PAIR DEVICE CB

Function: This command is used to delete the pairing information for a CB device.

Command Format: ATUP,<Addr>

Command Parameter(s):

- **Addr:** Device address

Example(s):

```
COMMAND:  ATUP,ECFE7E000001<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

10.9.4 Clear Pair List CB (ATCP)

DM **CLEAR PAIR LIST CB**

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

Command Format: ATCP

Example(s):

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

Note(s):

- *This command can only be used when the module is in the Idle State.*

10.10 User Confirmation Response (ATUCR)

DM **USER CONFIRMATION RESPONSE**

Function: Responds to a UC_REQ authentication event.

Command Format: ATUCR,<Addr>,<Yes_No>

Command Parameter(s):

- **Addr:** The address of the remote device.

- **Yes_No:**
  0 = No, reject numeric value.
  1 = Yes, confirm numeric value.

Example(s):

1. CB pairing is initiated using the ATP command to connection handle 0. Authentication is required by one side or the other and based on the module's IO_Capabilities (ATSP), UC_REQ is sent to the local module. The Numeric_Value is confirmed using the ATUCR command. Once both sides confirm the Numeric_Value, pairing is completed and the PAIRED event is sent:

```
COMMAND:  ATP,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          UC_REQ,ECFE7E000001,123456,1<cr_lf>
COMMAND:  ATUCR,ECFE7E000001,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,ECFE7E000001,1<cr_lf>
```

10.11 CB Legacy Pairing

The following PIN commands will only be used when pairing with legacy *Bluetooth* 2.0 and earlier devices that do not support Secure Simple Pairing.

10.11.1 PIN Response (ATPINR)

DM

PIN Response

Function: Responds to a PIN_REQ event.

Command Format: ATPINR,<Addr_Conn_Handle>,<PIN>

Command Parameter(s):

- **Addr_Conn_Handle:** Address of device or connection handle if connected.
- **PIN:** 4-16 alphanumeric characters (Case sensitive, includes spaces).

Example(s):

```
COMMAND:  ATPINR,ECFE7E000000,default<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

10.11.2 Fixed PIN (ATSPIN)

DM

SET PIN

Function: Sets a fixed PIN to be used with Bluetooth Core Specification 2.0 and earlier devices. All newer devices use Secure Simple Pairing, which uses a passkey instead of a PIN and can be configured through ATSP.

Command Format: ATSPIN,<PIN>

Command Parameter(s):

- **PIN:** 4-16 alphanumeric characters (Case sensitive, includes spaces). If set to 0, a fixed PIN will not be used and a PIN_REQUEST event will be sent, which can be responded to using the ATPR command.
  Default: 0 (Fixed PIN disabled)

Example(s):

```
COMMAND:  ATSPIN,default<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

Note(s):

- *For compatibility with devices with numeric keypads, fixed PINs shall be composed of only numeric characters.*

GET PIN

Function: Gets the fixed PIN.

Command Format: ATSPIN?

Response Format: <PIN>

Example(s):

```
COMMAND:  ATSPIN?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          default<cr_lf>
```

10.12 Remote Device Information

10.12.1 Read Remote Name (ATRRN)

DM

READ REMOTE NAME

Function: Gets the *Bluetooth* device name of a remote device.

Command Format: ATRRN,<Addr>

Command Parameter(s):

- **Addr:** Remote device address.

Example(s):

```
COMMAND:  ATRRN,ECFE7E000001<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          RN,BlueRadiosECFE7E000001<cr_lf>
EVENT:    <cr_lf>
          DONE,0,3<cr_lf>
```

Note(s):

- *The timeout for this command is controlled by the ATDMT timeout.*

10.12.2 Read Remote Services (ATRRS)

DM

READ REMOTE SERVICES

Function: Discovers services on a remote device.

Command Format: ATRRS,<Addr>,<Service_UUID>,<Service_Mask>

Command Parameter(s):

- **Addr:** Remote device address.

- **Service_UUID:** UUID of service to search for. UUID can be 16, 32, or 128 bit (4, 8, or 32 characters). To discover multiple service types set to 0. (16-Bit UUID for Serial Port Profile)

- **Service_Mask:** Services to search for when discovering multiple service types. (65535)
  1 (0x0001) = Serial Port (SPP)
  2 (0x0002) = Headset (HSP)
  4 (0x0004) = Dial Up Networking (DUN)
  8 (0x0008) = Fax (FAX)
  16 (0x0010) = LAN Access (LAP)
  32 (0x0020) = Object Push (OPP)
  64 (0x0040) = File Transfer (FTP)
  128 (0x0080) = Synchronization (SYNCH)
  256 (0x0100) = Hands free (HFP)
  512 (0x0200) = SIM Access (SAP)
  1024 (0x0400) = Basic Printing (BPP)
  2048 (0x0800) = Basic Imaging (BIP)
  4096 (0x1000) = Video Distribution (VDP)
  8192 (0x2000) = Phonebook Access (PBAP)
  16384 (0x4000) = Message Access (MAP)
  32768 (0x8000) = IOS Device Accessory

Example(s):

Two SPP Services Found:

```
COMMAND:  ATRRS,ECFE7E000001,1101<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          RS,1101,01,COM1<cr_lf>
EVENT:    <cr_lf>
          RS,1101,02,COM2<cr_lf>
EVENT:    <cr_lf>
          DONE,0,4<cr_lf>
```

No Services Found:

```
COMMAND:  ATRRS,ECFE7E000001,1101<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          RS,0,0,0<cr_lf>
EVENT:    <cr_lf>
          DONE,0,4<cr_lf>
```

Device Not Found:

```
COMMAND:  ATRRS,ECFE7E000001,1101<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
(ATDM TIMEOUT, default=4s)
EVENT:    <cr_lf>
          DONE,0,4<cr_lf>
```

Note(s):

- *The timeout for this command is controlled by the ATSDMT timeout.*

11 Extended Examples

The examples in this section will demonstrate using various different 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.

11.1 D2 Bridging From a CB Phone to an LE Device

1. Connect to the “COM0” SPP service on the D2 from the phone.

2. Send the command “ATMRC<cr>”. If any other data is received prior to “ATMRC<cr>” the module will enter data mode instead of command mode.

```
PHONE:     ATMRC<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           SPP,1,2<cr_lf>
```

3. Now that the D2 is in remote command over SPP, we can send commands to the D2 and all events generated on the D2 will be sent back to the phone. Send the ATDMLE command to connect to an LE device. Set BRSP_Mode to 0 in ATDMLE so the D2 doesn’t try to start the BRSP service on the remote device; this will keep the D2 in command mode so it can interact with the LE device using the GATT commands.

```
PHONE:     ATDMLE,ECFE7E000001,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,2,1,0,ECFE7E000001<cr_lf>
```

4. Once the D2 is connected, the GATT commands can be used to interact with the LE device from the phone. Here we use ATGDPS to discover all of the primary services on the LE device.

```
PHONE:     ATGDPS,2cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_DPS,0,00001,00011,1800<cr_lf>
EVENT:     <cr_lf>
           GATT_DPS,0,00012,00014,1801<cr_lf>
EVENT:     <cr_lf>
           GATT_DPS,0,00016,00019,180F<cr_lf>
EVENT:     <cr_lf>
           GATT_DPS,0,00015,65535,DA2B84F1627948DEBDC0AFBEA0226079<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,0,00<cr_lf>
```

5. When finished the connection can be terminated using the ATDH command.

```
PHONE:     ATDH,2<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
           <cr_lf>
           DISCONNECT,2,0<cr_lf>
```

11.2 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'll connect to an S2 module.

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

2. Discover the service you're looking, in this example we'll look for the battery service (BAS, UUID 0x180F). The service is found at attribute handles 16 – 19.

```
COMMAND:   ATGDPSU,0,180F<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_DPS,0,00016,00019,180F<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,0,00<cr_lf>
```

3. Discover the characteristic you're looking for within the service, in this example we'll look for the battery level (UUID = 0x2A19) within the battery service handle range of 16-19. The battery level characteristic is found at handle 18, with Read/Notify properties.

```
COMMAND:   ATGDCU,0,2A19,16,19<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_DC,0,00018,12,2A19<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,0,00<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 19, 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,0,19,19<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_DCD,0,00019,2902<cr_lf>
EVENT:     <cr_lf>
```

```
GATT_DONE,0,3,00<cr_lf>
```

4. Enable notifications by writing the value 0x0001 to the characteristic configuration descriptor.

```
COMMAND:  ATGW,0,19,2,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,5,00<cr_lf>
```

5. 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)

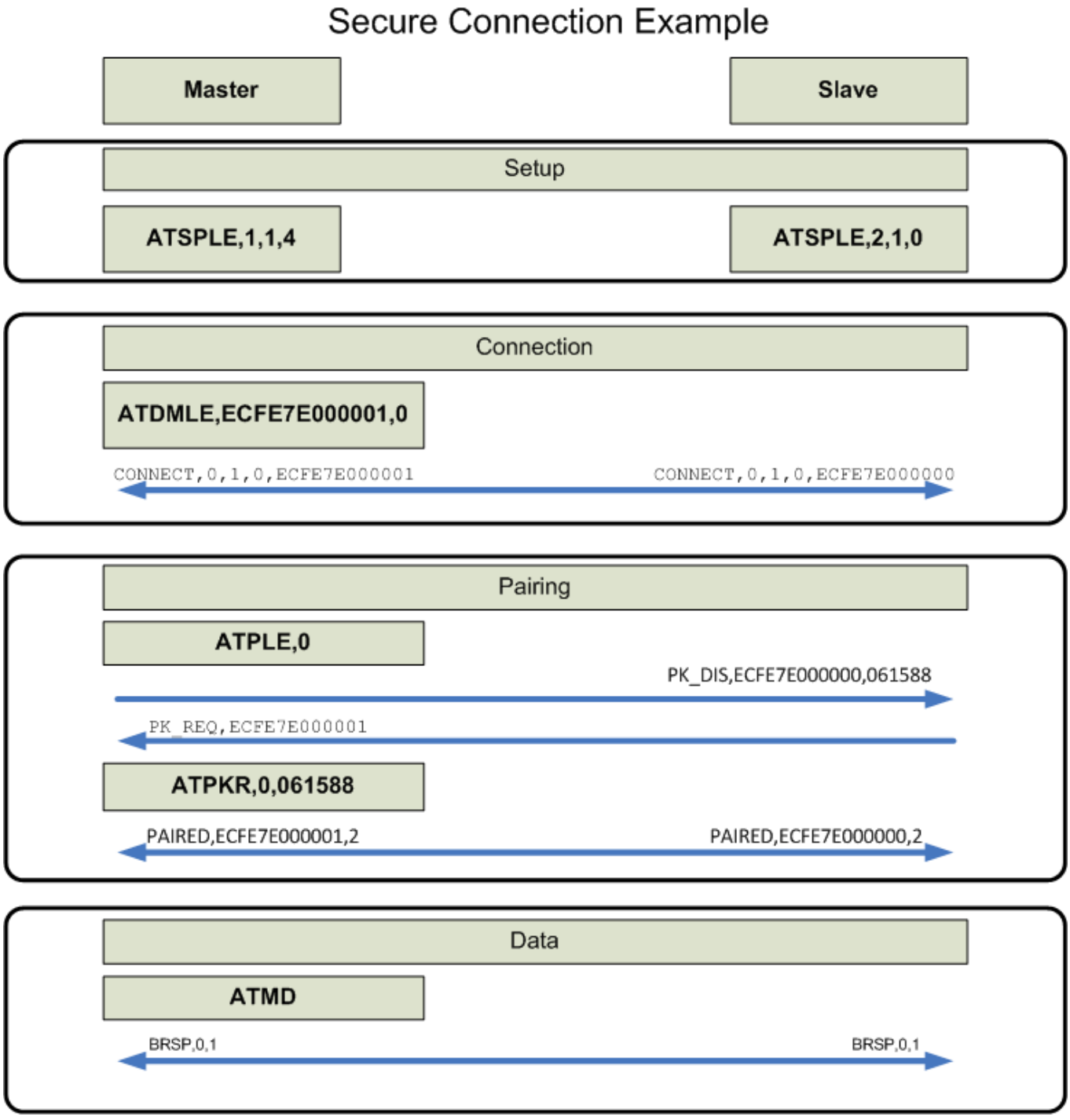
```
COMMAND:  ATSBAS,1,0,99<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

(On Master Module)

```
EVENT:    <cr_lf>
          GATT_VAL,0,00018,1,01,63<cr_lf>
```

6. 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.

11.3 Secure Connection Example



11.4 iBeacon Example

AT.s modules can be easily configured to advertise as iBeacons.

11.4.1 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]

11.4.2 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 or 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)

11.4.3 House Example

In this example we will create a set of beacons for a house. First generate a unique Proximity UUID to group all of 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

12 Command Set Summary Table

12.1 Command Status Responses

	Response	Description
	OK	
SD	OK	Command Accepted
	Error	
SD	ERROR	Command Not Accepted
	No Response	
SD		Invalid Command

12.2 Events

12.2.1 General Events (SM/DM)

M	Event	Description
	Reset	
SD	BR-LE4.0-XX	Reset
	Done	
SD	DONE	Background Task Complete
	Connection	
SD	CONNECT	Connect
SD	DISCONNECT	Disconnect
	Discovery	
SD	DISCOVERY	Device Discovery
	Pairing	
SD	PAIR_REQ	Pairing Request
SD	PAIRED	Pairing Successful
SD	PAIR_FAIL	Pairing Failed
	Authentication	
SD	PK_REQ	Passkey Request
SD	PK_DIS	Passkey Display
SD	PIN_REQ	PIN Request

12.2.2 Low Energy Events (SM/DM)

M	Event	Description
	Conn Param Update	
SD	SCCPS	Set Current Connection Parameter Status
SD	CPU	Connection Parameter Update
	GATT	
SD	GATT_DONE	GATT Done

SD	GATT_DPS	Discovered Primary Service
SD	GATT_DC	Discovered Characteristic
SD	GATT_DCD	Discovered Characteristic Descriptor
SD	GATT_VAL	Characteristic/Descriptor Value
SD	GATT_LVAL	Long Characteristic/Descriptor Value
	BRSP	
SD	BRSP	BRSP Status
	Resolved	
SD	RESOLVED	Private Resolvable Address Resolved
	Battery	
SD	BATT	Battery Event
	Heart Rate Service	
SMPO	HRS	Heart Rate Service Event
	Proximity Profile	
SMPO	PXP	Proximity Profile Service Event

12.2.3 Classic Bluetooth Events (DM Only)

M	Event	Description
	Remote Device Information	
DM	RN	Remote Name
DM	RS	Remote Service
	Authentication	
DM	UC_REQ	User Confirmation Request
DM	PIN_REQ	PIN Request
	SPP	
DM	SPP	SPP Status
	iOS Mfi Status	
DM	IOSTAT	iOS Mfi Authentication Status Event

12.3 General Commands (SM/DM)

M	Command	Description	Factory Default	Stored?
	AT			
SD	AT	Attention	-	-
	Help			
SD	ATHELP	Help	-	-
	Reset			
SD	ATRST	Reset	-	-
SD	ATFRST	Factory Reset	-	-

	Sleep			
SD	ATZ	Enable Sleep Mode	-	-
SD	ATSZ/ATSZ?	Set/Get Sleep Configuration	0,0,0	X
	Module Information			
SD	ATMT?	Get Module Type	-	-
SD	ATST?	Get Stack Type	-	-
SD	ATV?	Get Version	-	-
SD	ATA?	Get Address	-	-
SD	ATSN/ATSN?	Set/Get Name	BlueRadios######,0	X
	Status			
SD	ATCS?	Get Connection Status	-	-
SD	ATRSSI?	Get RSSI Value	-	-
	Config Control			
SD	ATSCL/ATSCL?	Set/Get Configuration Lock	0	X
SD	ATFC	Flash Config (Manual)	-	-
SD	ATSFC/ATSFC?	Set/Get Flash Configuration Setting	1	X
	Command Response			
SD	ATSRM/ATSRM?	Set/Get Response Mode	0,0	X
SD	ATSDIF/ATSDIF?	Set/Get Discovery Formatting	65535,1,1	X
	Hardware Config			
SD	ATSPL/ATSPL?	Set/Get Power Levels	3/2,1 (SM) / 10,10 (DM)	X
SD	ATSUART/ATSUART?	Set/Get UART Settings	7,0,0,1	X
SD	ATSPIO/ATSPIO?	Set/Get PIO	0,0 (All)	X
SD	ATSLED/ATSLED?	Set/Get LED	Led 0 = 100,65535,1 Led 1 =10,2000,1	X
SM	ATADC?	Get ADC		
SM	ATBL?	Get Battery Level		
SM	ATT?	Get Temperature		
SM	ATCT	Calibrate Temperature Sensor		X
	Serial Profile			
SD	ATSSP/ATSSP?	Set/Get Serial Profile Configuration	43,0	X
SD	+++	Escape To Command Mode	-	-
SD	ATMD	Data Mode	-	-
SD	ATMRC	Remote Command Mode	-	-
DM	ATSMRC/ATSMRC?	Set Remote Command Mode CB	1	X
	Cancel Command			
SD	ATDC	Cancel	-	-
	Disconnect			
SD	ATDH	Disconnect	-	-
SD	ATDX	Disconnect Immediate	-	-
	Authentication			
SD	ATPKR	Passkey Response	-	-
SD	ATSPK/ATSPK?	Set/Get Fixed Passkey	0	X

	Connection Bridging			
DM	ATB/ATB?	Bridge/Get Bridge	0,0	-
	Utilities			
SD	ATTXT	Transmitter Test	-	-
SD	ATRXT	Receiver Test	-	-
DM	ATTXTE	Extended Transmitter/Receiver Test	-	-
DM	ATFCCT	FCC Continuous Transmitter Test	-	-
DM	ATSIGT	Device Under Test Mode	-	-
SM	ATRFO	RF Observation	-	-
SD	ATCFG?	Configuration Dump	-	-

12.4 Low Energy Commands (SM/DM)

M	Command	Description	Factory Default	Stored?
	LE Status			
SD	ATSAPP/ATSAPP?	Set/Get Appearance	0	X
SD	ATSAT/ATSAT?	Set/Get Address Type	0	X
	LE Status			
SD	ATSLE?	Get State LE	-	-
SD	ATLCALE?	Get Last Connected Address LE	FFFFFFFFFFFF	X
	LE Default Behavior			
SD	ATSDBLE/ATSDBLE?	Set/Get Default Behavior LE	1,0	X
	BRSP Configuration			
SD	ATSBRSP/ATSBRSP?	Set/Get BRSP Parameters	1,3,0	X
	BAS Configuration			
SD	ATSBAS/ATSBAS?	Set/Get BAS Parameters	1,0,100,0	X
	DIS Configuration			
SMPO	ATSDIS/ATSDIS?	Set/Get DIS Parameters	See ATSDIS	X
	HRS Configuration			
SMPO	ATSHRS/ATSHRS?	Set/Get HRS Parameters	0,0,0,0	X
SMPO	ATHRM	Send Heart Rate Measurement		
	PXP Configuration			
SMPO	ATSPXP/ATSPXP?	Set/Get PXP Parameters	0,0,0	X
	Advertising			
SD	ATDSLE	Dial As Slave LE	-	-
SD	ATDSDLE	Dial As Slave Direct LE	-	-
SD	ATSDSLE/ATSDSLE?	Set/Get ATDSLE Parameters	0,0,7	X
SD	ATSDSTLE/ATSDSTLE?	Set/Get ATDSLE Timing Parameters	0,160,2048	X
SD	ATSDSDLE/ATSDSDLE?	Set/Get ATDSLE Advertising Data	Advertising Data = 0201FF020AFF0512FFFFFF FF1107FFFFFFFFFFFFFFFF	X

			FFFFFFFFFFFFFFF Scan Response Data = 05FFFFFFFFF1509FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	
	LE Discovery			
SD	ATDILE	LE Discovery	-	-
SD	ATSDILE/ATSDILE?	Set/Get ATDILE Parameters	0,0,1,10,0	X
SD	ATSDITLE/ATSDITLE?	Set/Get ATDILE Timing Parameters	10240,16,16	X
	LE Connect			
SD	ATDMLE	Dial As Master LE	-	-
SD	ATDMLLE	Dial As Master Last LE	-	-
SD	ATSDMTLE/ATSDMTLE?	Set/Get ATDMLE Timing Parameters	0,16,16	X
	Connection Parameters			
SD	ATSDCP/ATSDCP?	Set/Get Default Connection Params	16,16,0,400,0	X
SD	ATSCCP/ATSCCP?	Set/Get Current Connection Params	-	-
	LE Pairing			
SD	ATPLE/ATPLE?	Pair Device/Get Paired Device List LE	0,0	X
SD	ATSPLE/ATSPLE?	Set/Get Pairing Parameters LE	2,0,3	X
SD	ATUPLE	Un Pair Device LE	-	X
SD	ATCPLE	Clear Pair List LE	-	X
	White List			
SD	ATSWL/ATSWL?	Set/Get White List	0,0	X
SD	ATUWL	Un White List Device	-	X
SD	ATCWL	Clear White List	-	X
	GATT Commands			
SD	ATGDPS/ATGDPSU	GATT Discover Primary Services	-	-
SD	ATGDC/ATGDCU	GATT Discover Characteristics	-	-
SD	ATGDCD	GATT Discover Char Descriptors	-	-
SD	ATGR/ATGRU	GATT Read	-	-
SD	ATGRL	GATT Read Long	-	-
SD	ATGW/ATGWN	GATT Write	-	-
SD	ATGWP/ATGWPE	GATT Write Prepared	-	-

12.5 Classic Bluetooth Commands (DM Only)

M	Command	Description	Factory Default	Stored?
	Module Information			
DM	ATSCOD/ATSCOD?	Set/Get Class of Device	000000	X
DM	ATSSN/ATSSN?	Set/Get Default Service Name / UUID	COM1,1101	X
	CB Status			
DM	ATS?	Get State CB	-	-
DM	ATLCA?	Get Last Connected Address CB	FFFFFFFFFFFF	X
DM	ATLQ?	Get Link Quality	-	-

	CB Default Behavior			
DM	ATSDB/ATSDB?	Set/Get Default Behavior CB	1,0	X
DM	ATSDBA/ATSDBA?	Set/Get Default Behavior Address	FFFFFFFFFFFF	X
	Scanning			
DM	ATDS	Dial As Slave CB	-	-
DM	ATSDS/ATSDS?	Set/Get ATDS Parameters	1,1	X
DM	ATSDST/ATSDST?	Set/Get ATDS Timing Parameters	1024,512,1024,512	X
	CB Discovery			
DM	ATDI	Discovery CB	-	-
DM	ATSDI/ATSDI?	Set/Get ATDI Parameters	0,0,0,20	X
DM	ATSDIT/ATSDIT?	Set/Get ATDI Timing Parameters	10240	X
	CB Connect			
DM	ATDM	Dial As Master CB	-	-
DM	ATDML	Dial As Master Last CB	-	-
DM	ATSDMT/ATSDMT?	Set/Get ATDM Timing Parameters	8192	X
	Link Supervision TO			
DM	ATSLST/ATSLST?	Set/Get Link Supervision Timeout	6400	X
	CB Pairing			
DM	ATP/ATP?	Pair Device/Get Paired Devices CB	0,0	X
DM	ATSP/ATSP?	Set/Get Pairing Parameters CB	2,0,3	X
DM	ATUP	Un Pair Device CB	-	X
DM	ATCP	Clear Pair List CB	-	X
	CB Authentication			
DM	ATUCR	User Confirmation Response	-	-
	CB Legacy Security			
DM	ATPINR	PIN Response	-	-
DM	ATSPIN/ATSPIN?	Set/Get Fixed PIN	0	X
	Remote Device Information			
DM	ATRRN	Read Remote Name	-	-
DM	ATRRS	Read Remote Service	-	-

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)
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
LF – Line Feed
MCU – Microcontroller Unit
MISO – Master In Slave Out
MOSI – Master Out Slave In
NC – No Connect
PC – Personal Computer
PCB – Printed Circuit Board
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