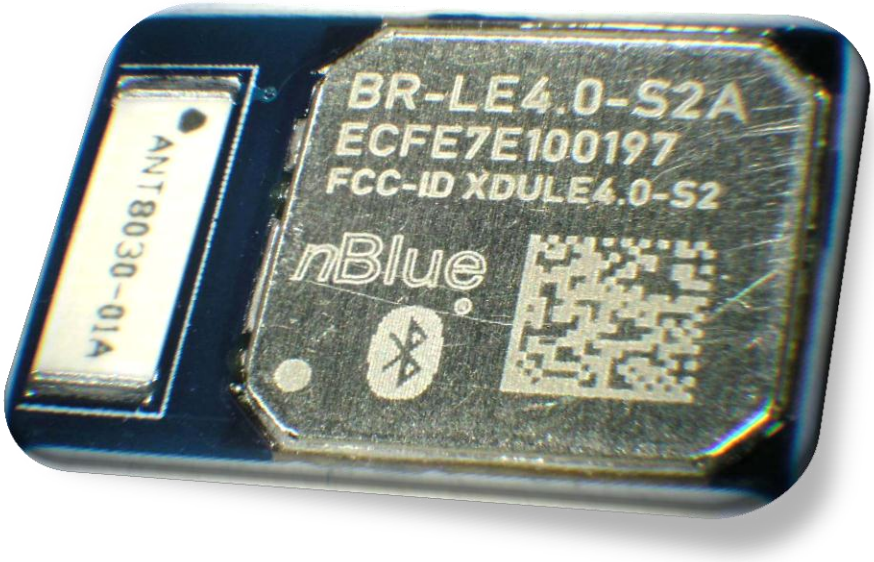
nBlue™ Bluetooth® 4.0
AT.s Command Set
V3.1.0



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
2.0.0	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.
2.1.0	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 BD_Addr instead of a Conn_Handle, since CB pairing can take place without being connected. ▪ Pairing commands can now accept a BD_Addr 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.
2.2.0	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.
3.0.0	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.
3.1.0	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 wihout 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 causes 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.

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.

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.
- Make sure the last three digits of the module's firmware version match the version of this document.
- Many changes have been made to the Single Mode LE AT command set to provide a single unified command set for both our Single Mode and Dual Mode modules. Command parameters such as <Conn_Handle> have been added to many commands to accommodate support for multiple connections.
- 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.
- V2.0.0 modules will not be compatible with modules with earlier firmware when trying to send data using BRSP as the BRSP service has been re-architected.
- V2.1.0 or greater modules will not be compatible with modules with earlier firmware when trying to send data using BRSP as the BRSP service has been modified to use 128-bit UUIDs in order to be fully compliant.
- 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.

2.2 Important Notes on Single Mode Combined Master/Slave Architecture

- The modules will now default to the slave role with their default behavior set to advertising, with an Ad_Type of connectable indirect advertisement (connectable + scannable)zx.
- The 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.
- The module 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

- The ATSCCP command can sometimes cause a DISCONNECT,02 to occur.
- 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.

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.

2.4 Related Documents

- nBlue Module User's Guide
- nBlue BR-EVAL-4.0-X2A Quick Start Guide

3 Hardware Notes

3.1 Electrical Specifications Summary

- The modules operate at a supply voltage (VDD) of 2.0-3.6V, 3.0V is recommended.
- VDD ripple should not exceed 100mV.
- Minimum logic high input voltage is 2.5V
- Maximum voltage level on any pin should not exceed 3.6V. **The I/O is not 5V tolerant.**
- All PIOs have a 4mA drive capability, except for PIO_2 and PIO_5, which can drive 20mA.
- Applying VDD to a PIO set to an output may permanently damage the module.
- All inputs are pulled low by an internal 20kΩ resistor.

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 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 PIO Functions

3.3.1 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 the 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 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.

3.3.2 PIO_3 - Sleep Mode Toggle Input

To toggle the device in and out of sleep mode PIO_3 needs to be pulsed high for at least 20ns. **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 AT commands.** Since the UART is shutdown when the device enters sleep mode, RTS will be set high, and upon waking up RTS will go low.

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.

When sleep mode is enabled and the module is in the idle state it will enter a deep sleep state which consumes only 0.4µA at 3.0V. If the module is connected or advertising, it will toggle between a sleep state of consuming only 0.9µA at 3.0V, and waking up to handle connection events. The power consumption in this case will be based on the connection parameters used.

3.3.3 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/disconnect when pulsed high for at least 5µs. If not connected the module will perform an ATDMLLE, and if connected the module will perform and ATDH. The module will respond with the same response as if the command was entered through the UART.

3.3.4 PIO_6 – BRSP Comm Mode Toggle Input

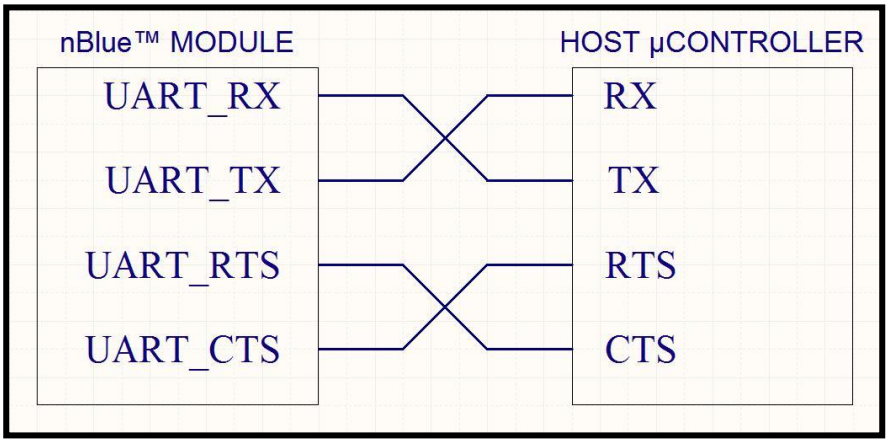
PIO_6 can be used as a hardware shortcut for toggling between command mode and data mode when connected by pulsing it high for at least 5µs. The module will respond with the same response as if the command was entered through the UART.

3.3.5 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.

3.4 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 Command Usage Guidelines

4.1 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.
- All response lines come back in the following format <cr_lf>“RESPONSE”<cr_lf>.
- All parameters are in decimal format, unless otherwise noted.
- Numerical data in responses will print with leading zeroes to keep the response length consistent. For example the maximum connection interval value is 3200, so a value of 16 will read back as 0016.
- 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.
- 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**
DM	**DUAL MODE ONLY COMMAND/EVENT**

4.2 Common Parameter/Response Value Descriptions

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

4.3 Common Terms/Abbreviations

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

5 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.

5.1 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>**

5.2 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>**

6 Events

6.1 General Events

6.1.1 Reset

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	Single Mode LE Module
BR-LE4.0-D2	Dual Mode BR/EDR and LE Module

Example(s):

1. Following a reset, 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 BR-LE4.0-S2 and ~880ms after reset on the BR-LE-4.0D2.*

6.1.2 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.*

6.1.3 Connect

SD

CONNECT EVENT

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

Event Format: CONNECT,<Conn_Handle>,<Conn_Type>,<Pairing_Status>,<BD_Addr>

Event Values:

- **Conn_Handle:** Connection handle
- **Conn_Type:**
  0 = Classic *Bluetooth*
  1 = Low Energy
- **Pairing_Status:**
  0 = Not Paired
  1 = Paired, Unauthenticated
  2 = Paired, Authenticated
- **BD_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>
```

6.1.4 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>
```

6.1.5 Discovery

SD

DISCOVERY EVENT

Function: The discovery event will occur when a device is discovered after the ATDI or ATDILE command has been issued. Descriptions of the different AD Structure Types can be found in Appendix B of the User's Guide.

Event Format: DISCOVERY,<Discovery_Type>,<BD_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)
  5 = Low Energy Non-connectable Indirect Advertisement (Not Connectable or Scannable)
  6 = Low Energy Scan Response

- **BD_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 Address
  3 = Resolvable Private Address

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

- **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>
```

6.1.6 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,<BD_Addr>,<Pairing_Type>

Event Values:

- **BD_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>
```

6.1.7 Paired

SD	**PAIRED EVENT**

Function: The paired event will be sent when pairing is successful.

Event Format: PAIRED,<BD_Addr>,<Pairing_Status>

Event Values:

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

- **Pairing Status:**
  1 = Paired, Unauthenticated
  2 = 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>
```

6.1.8 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,<BD_Addr>,<Reason>

Event Values:

- **BD_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>
```

6.1.9 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,<BD_Addr>

Event Values:

- **BD_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,0,1<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.*

6.1.10 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,<BD_Addr>,<Passkey>

Event Values:

- **BD_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.*

6.2 Low Energy Events

6.2.1 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, letting the user know if the update request was accepted or rejected. 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. 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.*

6.2.2 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. 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,0,1<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.*

6.2.3 GATT Done (GATT_DONE)

SM	**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

- **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>
```

6.2.4 GATT Discovered Primary Service (GATT_DPS)

SM	**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>
```

6.2.5 GATT Discovered Characteristic (GATT_DC)

SM **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>
```

6.2.6 GATT Discovered Characteristic Descriptor (GATT_DCD)

SM

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>
```

6.2.7 GATT Characteristic/Descriptor Value (GATT_VAL)

SM

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,0,3cr>
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>
```

6.2.8 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.*

6.3 Classic Bluetooth Events

6.3.1 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>
```

6.3.2 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_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 = Handsfree Audio Gateway	24 = Message Access Notification Server
11 = Handsfree	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 RCComm 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>
```

6.3.3 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,<BD_Addr>,<Numeric_Value>,<Response_Needed>

Event Values:

- **BD_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>
```

6.3.4 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,<BD_Addr>

Response Values:

- **BD_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.*

6.3.5 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

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 General Commands

7.1 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>**

7.2 Reset Commands

7.2.1 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.*

7.2.2 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.*

7.3 Sleep Commands

7.3.1 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 the User's Guide for more detailed information on sleep mode.*

7.3.2 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,1,1,1<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>
          1,1,1<cr_lf>
```

7.4 Module Information Commands

7.4.1 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	Single Mode LE Module
BR-LE4.0-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>
```

7.4.2 Stack Type (ATST?)

SD **GET STACK TYPE**

Function: Gets the stack type.

Command Format: ATST?

Response Format: BlueRadios nBlue

Example(s):

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

7.4.3 Firmware Version (ATV?)

SD	**GET FIRMWARE VERSION**

Function: Gets the radio's firmware version.

Command Format: ATV?

Response Format: <Firmware_Version>

Response Values:

- **Firmware_Version:** Module Firmware Version. The last three digits should match the command set version.

Example(s):

```
COMMAND:  ATV?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1.2.0.3.0.0-S2<cr_lf>
```

7.4.4 *Bluetooth* Device Address (ATA?)

SD	**GET BLUETOOTH DEVICE ADDRESS**

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

Command Format: ATA?

Response Format: <BD_Addr>

Response Values:

- **BD_Addr:** Local device address

Example(s):

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

7.4.5 *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>

Command Parameter(s):

- **Name:** *Bluetooth* Device Name - SM: 1-20 characters, DM: 1-32 characters

Factory Default: BlueRadios<*Bluetooth* Device Address Lower Address Part>

Example(s):

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

7.5 Module Status Commands

7.5.1 Connection Status (ATCS?)

SD

GET CONNECTION STATUS

Function: Gets the details of an existing connection.

Command Format: CS?,<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>,<BD_Addr>

Response Values:

- **Conn_Handle:** Connection handle
- **Conn_Type:**
  0 = Classic *Bluetooth* Connection
  1 = Low Energy Connection
- **Pairing Status:**
  0 = Not Paired
  1 = Paired, Unauthenticated
  2 = Paired, Authenticated

	▪ **BD_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>**

7.5.2 RSSI (ATRSSI?)

SD	**GET RSSI VALUE** **Function:** This command is used to obtain the RSSI value of the last packet received when connected. The radio has a built-in received signal-strength indication (RSSI), 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 an absolute receiver signal strength value in dBm to ± 6 dBm accuracy. It is a signed number on a logarithmic scale with 1-dB steps. **Command Format:** ATRSSI?,<Conn_Handle> **Command Parameter(s):** ▪ **Conn_Handle:** Connection handle of connection to read RSSI from. **Response Format:** <RSSI_Value> **Response Values:** ▪ **RSSI_Value:** The RSSI of the last packet received from the device. [-127-+20] **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.*

7.6 Configuration Control Commands

7.6.1 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:** 1-16 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.

Factory Default: Lock = 0

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>
          0<cr_lf>
```

7.6.2 Configuration Flashing Commands

7.6.3 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.*

7.6.4 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 **Factory Default:** Auto_Flash = 1 **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>
           0<cr_lf>
```

7.7 Response Configuration Commands

7.7.1 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.

Factory Default: Response_Mode = 0, Disconnected_Mode = 0

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>
```

7.7.2 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
  - 32768 (0x8000) = Manufacturer Specific Data

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

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

Factory Default: Data_Structure_Mask = 65535 (All Data Structures Enabled), Name_Format = 0, Hyphens = 1

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>
```

7.8 Hardware Configuration / Control Commands

7.8.1 Set Power Level (ATSPL)

SD

SET POWER LEVELS

Function: Sets the transmit power and receive sensitivity of the module.

Command Format: ATSPL,<Transmit_Power>,<Receive_Sensitivity>

Command Parameter(s):

- **Transmit_Power:** Integer value from 0 to 3.
  0 = -23 dBm
  1 = -6 dBm [~9mW more power consumption than at -23]
  2 = 0 dBm [~18mW more power consumption than at -23]
  3 = 4 dBm [~33mW more power consumption than at -23]

- **Receive_Sensitivity:** Integer value from 0 to 1.
  0 = -89 dBm Standard Mode
  1 = -95 dBm High-Gain Mode [~7.5mW more power consumption than at -89]

Factory Default: Transmit_Power = 3 (4dBm), Receive_Sensitivity = 1 (-95dBm)

Example(s):

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

Note(s):

- *Receive_Sensitivity is not adjustable on Dual Mode modules and will stay fixed at -95dBm.*

GET POWER LEVELS

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

Command Format: ATSPL?

Response Format: <Transmit_Power>,<Receive_Sensitivity>

Example(s):

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

7.8.2 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 – 1000000bps], enter Value from table below.
  (230400, 460800 and 1000000 are only available on Dual Mode modules.)

Baud rate	Value	Error (%)
9600	3	0.14
19200	4	0.14
38400	5	0.14
57600	6	0.03
115200	7	0.03
230400	**8**	**0.03**
460800	**9**	**0.03**
1000000	**10**	**0.03**

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

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

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

Factory Default: Baud_Rate = 7, Parity = 0, Stop_Bits = 0, Flow_Control = 1 (115200, 8-N-1, FC)

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>
```

7.8.3 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: 0,1,2,5,7,8,9,10,11,12,13,14
  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

- **Value: (Must be 0 if direction is set to input)**
  0 = 0V
  1 = VDD

Factory Default: Direction = 0, Value = 0 (For All PIO_Num Values)

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 using the ATSLED command, but they must remain outputs.*

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>
```

7.8.4 LED Configuration (ATSLED)

SD

SET LED

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

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

Command Parameter(s):

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

- **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.

- **Period:** Integer decimal value from 1ms to 65,535ms

Factory Default (PIO_2): Duty Cycle = 100, Period = 65535 (PIO_2),

Factory Default (PIO_5): Duty Cycle = 10, Period = 2000 (PIO_5)

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>

Command Parameter(s):

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

Response Format: <Duty Cycle>,<Period>

Example(s):

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

7.8.5 Get ADC (ATADC?)

SM	**GET ADC** **Function:** Takes a reading from one of the module's ADCs in mV using an internal 1.25 voltage reference. **Command Format:** ATADC?,<ADC_Num>

Command Parameter(s):

- **ADC_Num:** 0-5

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

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>
```

7.8.6 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>
```

7.8.7 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.*

7.8.8 Calibrate Temperature Sensor (ATCT)

SM	CALIBRATE TEMPERATURE

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

Command Format: ATST,<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.*

7.9 Serial Profile Commands

7.9.1 Serial Profile Configuration (ATSSP)

SD	SET SERIAL PROFILE

	Function: Sets the default communication mode, escape character and no data timeout parameters. **Command Format:** ATSSP,<Escape_Char>,<No_Data_Timeout> **Command Parameter(s):** ▪ **Escape_Char:** The escape character used to put the radio into back 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.** ▪ **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. **Factory Default:** Escape_Char = 43 ('+'), No_Data_Timeout = 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>**

7.9.2 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.*

7.9.3 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.*

7.9.4 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 RF. Events on the remote side will not be returned over RF.

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):

1. 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.*

7.10 Cancel Command (ATDC)

SD	**CANCEL**

Function: The DC 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.

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>
```

7.11 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>
```

7.12 Authentication Commands

7.12.1 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,<BD_Addr_Conn_Handle>,<Passkey>

Command Parameter(s):

- **BD_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>
```

7.13 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>** **0,1<cr_lf>**

7.14 Utility Commands

7.14.1 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.*
- *The module will transmit at the maximum output power of 4dB.*

7.14.2 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 an 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.*

7.14.3 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>
```

7.14.4 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?         = 1.2.0.2.2.5-S2<cr_lf>
          ATA?         = ECFE7E000000<cr_lf>
          ATSN?        = BlueRadiosFFFFFF<cr_lf>
          ATSZ?        = 0,0,0<cr_lf>
          ATSFC?       = 1<cr_lf>
          ATSCL?       = 0<cr_lf>
          ATSRM?       = 0,0<cr_lf>
          ATSDIF?      = 65535,0,1<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<cr_lf>
          ATSLED?,1    = 010,02000<cr_lf>
          ATSSP?       = 043,00000<cr_lf>
          ATSPK?       = 0<cr_lf>
          ATSAPP?      = 0<cr_lf>
          ATSDBLE?     = 0,0<cr_lf>
          ATSBRSP?     = 1,3,0<cr_lf>
          ATSBAS?      = 1,0,100<cr_lf>
          ATSDSLE?     = 0,0,7<cr_lf>
          ATSDSTLE?    = 00000,00160,02048<cr_lf>
          ATSDSDLE?,0  = 0<cr_lf>
          ATSDSDLE?,1  = 0<cr_lf>
          ATSDILE?     = 0,0,1,10<cr_lf>
          ATSDITLE?    = 10240,00016,00016<cr_lf>
          ATSDMTLE?    = 00000,00016,00016<cr_lf>
          ATSDCP?      = 00008,00016,000,0400,1792<cr_lf>
          ATSPLE?      = 1,0,3<cr_lf>
```

Note(s):

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

8 Low Energy Commands

8.1 Module Information Commands

8.1.1 Appearance (ATSAPP)

SM	**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:** 0 = Unknown !0 = 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 **Factory Default:** 0 (Unknown) **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>** **00000<cr_lf>**

8.2 LE Status Commands

8.2.1 LE State (SLE?)

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

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

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

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

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

Example(s):

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

8.2.1 LE Last Connected Address (ATLCALE?)

SD **GET LAST CONNECTED ADDRESS LE**

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

Command Format: ATLCALE?

Response Format: <BD_Addr>

Response Values:

- **BD_Addr:** Slave device address

Example(s):

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

8.3 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

Factory Default (SM): Action = 1, Trigger = 0

Factory Default (DM): Action = 0, Trigger = 0

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 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. For an idle trigger behavior,*

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>
```

8.4 GATT Service Configuration

8.4.1 BRSP Service Configuration (ATSBRSP)

SD

SET BRSP

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 (SM Only)
  4 = OTA Firmware Update Mode Supported – (SM Only. Can only be enabled if a Numonyx 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)

Factory Default (SM): Service_Enable = 1, Features_Mask = 3, Security_Mode = 0

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

Function: Sets 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>
```

8.4.2 Battery Service Configuration (ATSBAS)

SM

SET BAS CONFIGURATION

Function: Sets the Battery Service (BAS) parameters.

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

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.
  1 = Automatic Mode – Level is automatically read at the specified update interval. Use if a 3.0V coin cell is connected directly to VDD.

- **Level_Update_Interval:**

Mode = 0 [0-100 %]
Mode = 1 [1-687194 s]

Factory Default: Service_Enable = 1, Mode = 0, Battery_Level = 100

Example(s):

1. Enables the BAS service in manual mode with the level set to 100%.

```
COMMAND:  ATSBAS,1,0,00<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>
```

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 PARAMETERS

Function: Sets 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>
```

8.5 Advertising Commands

8.5.1 Advertise (ATDSLE)

SD	**DIAL AS SLAVE (ADVERTISE)**

Function: This command places the module in the advertising state, allowing it to be discovered by other LE devices. Depending on the ATSDS Ad_Type, this can also make 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.*

8.5.2 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 connect to a master in auto connect mode, which is always trying to connect to devices in its whitelist. For this reason connectable direct advertising will automatically timeout after 1.28s regardless of the ATSDSTLE Ad_Timeout setting.

Command Format: ATDSDLE,<BD_Addr>,<BRSP_Mode>,<Addr_Type>

Command Parameter(s):

- **BD_Addr:** Master address for connectable direct advertising. If not specified the module will advertise using the Ad_Type set in ATSDSLE.

- **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):

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

COMMAND:  ATDMLE,ECFE7E000000<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          DONE,1,0<cr_lf>
```

Note(s):

- *This command overrides the ATSDS Ad_Type setting to 1, 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 initate BRSP which will lead to a BRSP,05 event.*

8.5.3 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

Factory Default: White_List_Filter = 0, Ad_Type = 0, Ad_Channel_Map = 7

Example(s):

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

Note(s):

- *To use connectable direct advertising, use ATSDSLE.*

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>
```

8.5.4 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 30720 [ms].
  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.
- **Connected_Ad_Interval:** Integer value from 160 to 16384 [.625ms]. This is the interval that advertisements will be sent when the module is connected.

Factory Default: Ad_Timeout = 0 (No Timeout), Unconnected_Ad_Interval = 160 (100ms),
Connected_Ad_Interval = 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.*

GET ADVERTISING TIMING

Function: Sets 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>
```

8.5.5 Advertising Data (ATSDSDLE)

SD **SET ADVERTISING DATA**

Function: This command is used to configure the module's advertising and scan response data. Since the Flags ad structure is required to be in the Advertising Data it will automatically be populated in the first 3 bytes, so the advertising data is limited to a maximum of 28 bytes. See the BLE Basics section of the User's Guide for detailed information about the data format. Descriptions of the different AD Structure Types can be found in Appendix B.

The advertising 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 invalid, but it is up to the user to make sure the ad data meets the requirements of the Bluetooth 4.0 specification.

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

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

Command Parameter(s):

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

- **Data:**
  Advertising Data - 1-28 bytes (2-56 characters, 3 bytes reserved for Flags)
  Scan Response Data - 1-31 bytes (2-62 characters)

  0 = Use Default Data
  !0 = Custom data 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.

Factory Default: 0 (Use Default Data)

Example(s):

1. 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). 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>
```

2. 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,0,07FF8500000102<cr>
RESPONSE: <cr_lf>
          ERROR,01<cr_lf>
```

3. This example fails because the length was set to 7, but only 5 bytes of data were passed.

```
COMMAND:  ATSDSDLE,0,07FF8500FF0102<cr>
RESPONSE: <cr_lf>
          ERROR,01<cr_lf>
```

4. 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,1,07FF8500FF0102020A04051208001000<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

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

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

GET ADVERTISING DATA

Function: Gets the ATSDSDLE command settings.

Command Format: ATSDSLE?,<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:  ATSDSLE?,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 3 above.

```
COMMAND:  ATSDSLE?,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          07FF8500FF0102020A04051208001000<cr_lf>
```

8.6 LE Discovery Commands

8.6.1 LE Discovery (ATDILE)

SD **DISCOVERY LE**

Function: This command is used to discover nearby advertising LE devices. See the BLE Basics section of the User's Guide for detailed information about the parameters and responses. Descriptions of the different AD Structure Types can be found in Appendix B.

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.*

8.6.2 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>,<Disc_Mode_Filter>,<Active_Scan>,

<Max_Devices>

Command Parameter(s):

- **White_List_Filter:** The whitelist can be configured using the ATSWL command.
  0 = Disabled
  1 = Discover White List devices only.

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

- **Active_Scan:**
  0 = Disabled
  1 = Will 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.

Factory Default: White_List_Filter = 0, Disc_Mode_Filter = 0, Active_Scan = 1, Max_Devices = 10

Example(s):

```
COMMAND:  ATSDILE,0,0,1,10<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>,<Disc_Mode_Filter>,<Active_Scan>,<Max_Devices>

Example(s):

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

8.6.3 LE Discovery Timing Configuration (ATSDITLE)

SD	**SET DISCOVERY TIMING LE** **Function:** Sets the module's discovery (ATDILE) timing parameters.

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.
- **Interval:** Integer value from 4 to 16384 [.625ms].
- **Window:** Integer value from 4 to 16384 [.625ms].

Factory Default: Timeout = 10240 (10240ms), Interval = 16 (10ms), Window = 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: Sets 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>
```

8.7 LE Connect Commands

8.7.1 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,<BD_Addr>,<BRSP_Mode>,<Addr_Type>

Command Parameter(s):

- **BD_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):

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>
```

2. 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>
```

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

```
COMMAND:   ATDMLE,ECFE7E000001<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.*

- *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.*

8.7.2 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.*

8.7.3 LE Connect Timing Configuration (ATSDMTLE)

SD	**SET CONNECT TIMING LE**

Function: Sets the connection initiation timing parameters.

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].
- **Window:** Integer value from 4 to 16384 [.625ms].

Factory Default: Timeout = 0 (No Timeout), Interval = 16 (10ms), Window = 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>
```

8.8 Connection Parameters

8.8.1 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.
- **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)).
- **Slave_Auto_Update:**
  0 = If connected to as a slave, the module will accept any connection parameters.
  1 = 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.

Factory Default: Min_Conn_Interval = 8 (10ms), Max_Conn_Interval = 16 (20ms), Slave_Latency = 0, Supervision_Timeout: 400 (4s), Slave_Auto_Update = 0

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)*
- *The settings will not take effect on 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>
```

8.8.2 Current Connection Parameters (ATSCCP)

SD

SET CURRENT CONNECTION PARAMETERS

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

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

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.
- **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. 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>
```

8.9 LE Pairing Commands

8.9.1 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,<BD_Addr_Conn_Handle>,<Accept_Request>

Command Parameter(s):

- **BD_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>,<BD_Addrs>

Response Value(s):

- **Count:** Number of paired devices
- **BD_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>
```

8.9.2 LE Pairing Configuration (ATSPLE)

SD

SET PAIRING LE

Function: This command is used to configure LE pairing.

Command Format: ATSPLE,<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 = 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.

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

Factory Default: Mode = 1, Authentication = 0, IO_Capabilities = 3

Example(s):

```
COMMAND:  ATSPLE,1,0,3<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.*

GET PAIRING LE

Function: Gets the ATSPLE settings.

Command Format: ATSPLE?

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

Example(s):

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

8.9.3 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,<BD_Addr>

Command Parameter(s):

- **BD_Addr:** Device address

Example(s):

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

8.9.4 LE Clear Pair List (ATCPLE)

SD

CLEAR PAIR LIST LE

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

Command Format: ATCPLE

Example(s):

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

Note(s):

- *This command can only be used if the module is not connected.*

8.9.5 Fixed Passkey (ATSPK)

SD

SET PASSKEY

Function: Sets a fixed passkey to be used for Low Energy pairing authentication. For a fixed passkey to be used, Authentication needs to be enabled in ATSPLE and the IO_Capabilities should not be set to No Input or Output, or the Passkey Entry method of authentication will not be used and authentication will not occur.

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.

Factory Default: Passkey = 0 (Fixed passkey disabled)

Example(s):

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

GET PASSKEY

Function: Gets the fixed passkey.

Command Format: ATSPK?

Response Format: <Passkey>

Example(s):

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

8.10 White List Commands

8.10.1 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,<BD_Addr>,<Addr_Type>

Command Parameter(s):

- **BD_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.*
- *Up to 25 devices can be added to the whitelist on dual mode modules.*

GET WHITE LIST

Function: This command is used to read the White List.

Command Format: ATSWL?

Response Format: <Count>,<BD_Addrs>

Response Value(s):

- **Count:** Number of device in the White List
- **BD_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>
```

8.10.2 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,<BD_Addr>

Command Parameter(s):

- **BD_Addr:** Device address

Example(s):

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

8.10.3 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 can only be used if the module is not connected.*

8.11 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.

8.11.1 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>

Command Parameter(s):

- **Conn_Handle:** Connection handle.

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>
```

8.11.2 Discovery 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 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>
```

8.11.3 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_Add_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]

- **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. ATGDC is used to discover all characterisics 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 Att_Handle is specified, End_Att_Handle must also be specified.*

8.11.4 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_Add_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]
- **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. 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 Att_Handle is specified, End_Att_Handle must also be specified.*

8.11.5 Characteristic Descriptor Discovery (ATGDCD)

SD

GATT DISCOVER ALL CHARACTERISTIC DESCRIPTORS

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

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 plus 1 of a specific characteristic. Start_Att_Handle is included in the search. [1-65535]
- **End_Att_Handle:** Attribute handle to stop searching at, typically the Char_Value_Att_Handle of the next characteristic minus 2. End_Att_Handle is included in the search. Must be greater than or equal to Start_End_Handle. [1-65535]

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>
```

Note(s):

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

8.11.6 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 = ASCII 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>
```

8.11.7 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_Add_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 = ASCII 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>
```

8.11.8 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 DONE event.

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: 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 = ASCII String
- **Value:** Value data.

Example(s):

1. Writing a 16-bit value of 1 using format 1.

```
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 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 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 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>
```

8.11.9 Characteristic Write No Response (ATGWN)

SD

GATT WRITE NO RESPONSE

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.

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 = ASCII String
- **Value:** Value data.

Example(s):

1. Writing a 16-bit value of 1 using format 1.

```
COMMAND:  ATGWN,0,99,2,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

2. Writing a 16-bit value of 1 using 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 format 5.

```
COMMAND:  ATGWN,0,99,5,HELLO<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

4. Writing the string “HELLO” using format 0. This is equivalent to example 3.

```
COMMAND:  ATGWN,0,99,0,48454C4C4F<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

9 Classic Bluetooth Commands

9.1 Important Notes

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

9.2 Module Information Commands

9.2.1 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. **Factory 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>**

9.2.2 Local Service Name (ATSSN)

DM	**SET SERVICE NAME** **Function:** Sets the *Bluetooth* service name of the module's Serial Port Profile service. **Command Format:** ATSSN,<Service Name> **Command Parameter(s):** ▪ **Service_Name:** 1-16 alphanumeric characters

Factory Default: Service_Name = “COM1”

Example(s):

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

GET SERVICE NAME

Function: Gets the *Bluetooth* service name of the module’s Serial Port Profile service.

Command Format: ATSSN?

Response Format: <Service_Name>

Example(s):

```
COMMAND:  ATSSN?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          COM0<cr_lf>
```

9.3 CB Status Commands

9.3.1 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>
```

9.3.2 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: <BD_Addr>

Response Values:

- **BD_Addr:** Slave device address

Example(s):

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

9.3.3 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.*

9.4 CB Default Behavior Commands

9.4.1 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_BD_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

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. For an idle trigger behavior,*

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>
```

9.4.2 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,<BD_Addr>

Command Parameter(s):

- **BD_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: <BD_Addr>

Example(s):

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

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

9.5 Scanning Commands

9.5.1 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>
```

9.5.2 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)

Factory Default: Connectable = 1, Discoverable = 1

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>
```

9.5.3 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].
- **Connectable_Window:** Page Scan Window, Integer value from 17 to 4096 [.625ms].
- **Discoverable_Interval:** Inquiry Scan Interval, Integer value from 18 to 4096 [.625ms].
- **Discoverable_Window:** Inquiry Scan Window, Integer value from 17 to 4096 [.625ms].

Factory Default: Timeout = 0 (No Timeout, General Discoverable Mode),
Connectable_Interval = 2048 (1280ms), Connectable_Window = 18 (11.25ms),
Discoverable_Interval = 4096 (2560ms), Discoverable_Window = 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 Advertising 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>
```

9.6 CB Discovery Commands

9.6.1 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.*

9.6.2 CB Discovery Configuration (ATSDI)

DM

SET DISCOVERY CB

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

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

Command Parameter(s):

- **Disc_Mode_Filter:**
  0 = Disabled (Discover all discoverable devices)

1 = Discover only limited discoverable devices

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

Factory Default: Disc_Mode_Filter = 0, Max_Devices = 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: <Disc_Mode_Filter>,<Max_Devices>

Example(s):

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

9.6.3 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].

Factory Default: Timeout = 8 (10.24s)

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: Sets 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>
```

9.7 CB Connect Commands

9.7.1 CB Connect (ATDM)

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

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

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

Command Parameter(s):

- **BD_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-Buit 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.*

9.7.2 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):
- **BD_Addr:** Slave device address.
- **Service_UUID:** 4 hex characters, UUID of service to connect to. 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.*

9.7.3 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].

Factory Default: Timeout = 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>
```

9.8 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].

Factory Default: Timeout = 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>
```

9.9 CB Pairing Commands

9.9.1 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,<BD_Addr_Conn_Handle>,<Accept_Request>

Command Parameter(s):

- **BD_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>,<BD_Addrs>

Response Value(s):

- **Count:** Number of paired devices
- **BD_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>
```

9.9.2 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
  2 = Numeric Keyboard Only
  3 = No Input or Output

Factory Default: Mode = 2, Authentication = 0, IO_Capabilities = 3

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.*

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>
```

9.9.3 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,<BD_Addr>

Command Parameter(s):

- **BD_Addr:** Device address

Example(s):

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

9.9.4 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.*

9.10 User Confirmation Response (ATUCR)

DM	USER CONFIRMATION RESPONSE

Function: Responds to a UC_REQ authentication event.

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

Command Parameter(s):

- **BD_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 ATPL 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:   ATPL,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>
```

9.11 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.

9.11.1 PIN Response (ATPINR)

DM

PIN Response

Function: Responds to a PIN_REQ event.

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

Command Parameter(s):

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

Example(s):

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

9.11.2 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.

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.

Factory Default: PIN = 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>
```

9.12 Remote Device Information

9.12.1 Read Remote Name (ATRRN)

DM

READ REMOTE NAME

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

Command Format: ATRRN,<BD_Addr>

Command Parameter(s):

- **BD_Addr:** Remote device address.

Example(s):

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

Note(s):

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

9.12.2 Read Remote Services (ATRRS)

DM

READ REMOTE SERVICES

Function: Discovers services on a remote device.

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

Command Parameter(s):

- **BD_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-Buit 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) = Handsfree (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.*

10 Command Set Summary Table

10.1 Command Status Responses

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

10.2 Events

10.2.1 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

10.2.2 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
	BRSP	
SD	BRSP	BRSP Status

10.2.3 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

10.3 General Commands (SM/DM)

M	Command	Description	Factory Default	Stored?
	AT			
SD	AT	Attention	-	-
	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	BlueRadiosXXXXXX	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	65535,0,1	X
	Hardware Config			
SD	ATSPL/ATSPL?	Set/Get Power Levels	3,1	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	0 = 100,65535 / 1 =10,2000	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	-	-
	Cancel Command			
SD	ATDC	Cancel	-	-
	Disconnect Command			
SD	ATDH	Disconnect	-	-
	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	-	-
SM	ATRFO	RF Observation	-	-
SD	ATCFG?	Configuration Dump	-	-

10.4 Low Energy Commands (SM/DM)

M	Command	Description	Factory Default	Stored?
	LE Status			
SD	ATSAPP/ATSAPP?	Set/Get Appearance	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 (SM) / 0,0 (DM)	X
	BRSP Configuration			
SD	ATSBRSP/ATSBRSP?	Set/Get BRSP Parameters	1,3,0	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	0	X
	LE Discovery			
SD	ATDILE	LE Discovery	-	-
SD	ATSDILE/ATSDILE?	Set/Get ATDILE Parameters	0,0,1,10	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	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	1,0,3	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	ATGW/ATGWN	GATT Write	-	-

10.5 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 SPP Service Name	COM1	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	-	-