1BLE™ *Bluetooth*® 4.0 AT Command Set V1.1.0

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
1.0.0	02/07/2013	**Initial Document** ▪ First version of AT Command set for single mode.
1.0.1	04/29/2013	• Changed formatting of version to track document version. • Fixed issues some commands and events not being documented correctly.
1.1.0	03/07/2014	• Modified document to be specific to 1BLE modules • Corrected DONE event display when using ATCS? • Corrected SCCPS event display when using ATSCCP • ATSPK now stores the Passkey correctly • ATSDCP now stores default connection parameters correctly • ATPLE now allows pairing in the master or slave role • ATSWL? now displays the white list correctly after using ATUWL • ATPLE? now displays the paired list correctly after using ATUPLE • ATGRU will now display in any valid format • Updated the transmit power level steps (ATSPL) • The pairing configuration (ATSPLE) now defaults to automatically accept all pairing requests • The discovery format (ATSDIF) now defaults the Name Format as a string • Added ATLEENDT command to end LE Receiver or Transmitter testing

1 Introduction

This document describes the protocol used to control and configure ***Stonestreet One 1BLE™*** 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.

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 ***Stonestreet One 1BLE™*** modules can be configured, commanded, and controlled through simple ASCII strings through the hardware serial UART or over a remote Bluetooth RF connection.

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.
- For complete firmware/documentation compatibility, make sure the three digits after the "d" of the module's firmware version (displayed by the ATV? command) match the version of this document.
- Certain command parameters such as <Conn_Handle> are required even if the device (such as the BR-SS-S2x) can be connected to only 1 other device at a time and thus <Conn_Handle> will always be set to 0. This is to support potential future products that may support multiple connections.
- The BR-SS-S2x module uses an EM9301. The EM9301 is a **single** state machine Bluetooth LE Controller. The Link Layer state machine allows only one state to be active at a time. In addition, there are restrictions on state to state transitions (see Bluetooth Specification Version 4.0 [Vol 6], Link Layer States). For instance the module cannot be advertising and scanning at the same time, nor can it transition directly from Advertising State to Scanning State, but must transition to Standby State in between. For example; if the module is commanded to advertise (ATDSLE), the advertising must first be cancelled (e.g. ATDC) before it can be commanded to scan (ATDILE).

2.2 Known Issues in This Version

- None

2.3 Related Documents

- Quick Start Guide TWRPI-BLE-DEMO

3 Hardware Notes

3.1 Electrical Specifications Summary

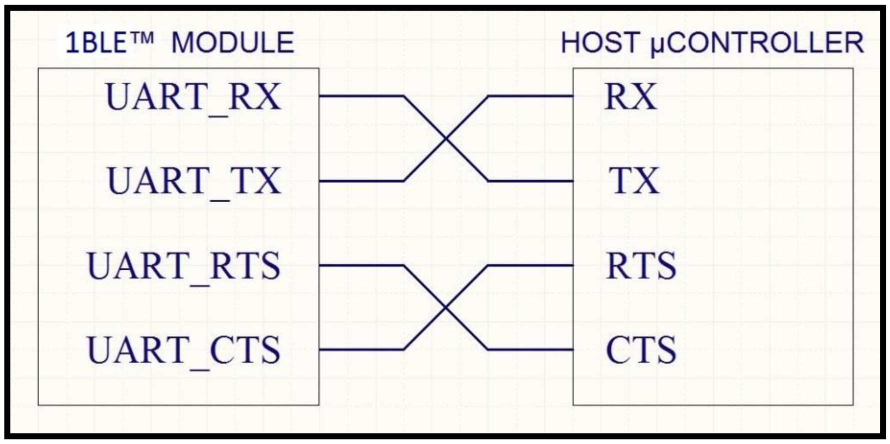
- The module operates at a supply voltage (VDD) of 1.8-3.6V, 3.0V is recommended.
- VDD ripple should not exceed 100mV.
- Minimum logic high input voltage is 2.5V
- Maximum voltage level on any pin should not exceed 3.6V. **The I/O is not 5V tolerant.**
- All PIOs have a 25mA drive capability.
- Applying VDD to a PIO set to an output may permanently damage the module.
- All inputs are pulled low by an internal 22k-50kΩ 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 235ms after power on. To reset the module, the RESET line must be pulsed low for at least 1µS.

3.3 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 TBD

5 Command Usage Guidelines

5.1 Command Usage

- All commands are entered in the following format: “COMMAND”<cr>. The command parser does not accept a line feed <lf> after the carriage return <cr>. If using HyperTerminal the following option should be disabled, or commands will not be submitted correctly: Send line ends with line feeds.
- All commands are typed exactly as shown in the examples. The commands themselves are not case sensitive, but the arguments may be, depending on the command.
- All response lines come back in the following format <cr_lf>“RESPONSE”<cr_lf>.
- All parameters are in decimal format, unless otherwise noted.
- The factory default setting of any stored parameters will be identified by a blue background like this.
- Some command parameters are optional and will be identified by a gray background like this. The default value that the parameter will take if not specified will also be identified by a gray background or a blue background if the value is stored in flash.

5.2 Common Parameter/Response Value Descriptions

- <cr> = 0x0D (carriage return)
- <cr_lf> = 0x0D0A (carriage return, linefeed)
- <Addr> = Device Address, 12 hex characters.
- <Conn_Handle> = Connection Handle, 0-9 (always 0 for BR-SS-S2x module).
- <Att_Handle> = Attribute Handle, specific to the GATT commands and events, 1-65535.

5.3 Common Terms/Abbreviations

- LE = Low Energy
- Discovery/Discovering is used interchangeably with Scan/Scanning

6 Command Status Responses

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

6.1 OK (OK)

OK STATUS RESPONSE

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

Example(s):

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

6.2 Error (ERROR)

ERROR STATUS RESPONSE

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

Response Format: ERROR,<Error Code>

Response Values:

- **Error Code:**
  01 = Invalid Parameters
  02 = Invalid Role
  03 = Invalid State
  04 = Invalid Password
  05 = Invalid Connection Handle
  06 = Configuration Locked
  07 = List Error
  08 = Hardware Error
  09 = No Address Stored
  10 = Bluetooth Error
  11 = Memory Allocation Error
  12 = GATT Request Pending

Example(s):

```
    RESPONSE: <cr_lf>
              ERROR,01<cr_lf>
```

7 Events

7.1 General Events

7.1.1 Reset (SS1-LE4.0-1BLE)

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:**

SS1-LE4.0-1BLE	1BLE Single Mode LE Module

Example(s):

1. Following a reset on a 1BLE single mode module, the Module_Type is sent:

```
EVENT: <cr_lf>
       SS1-LE4.0-1BLE<cr_lf>
```

Note(s):

- *This event will occur ~235ms after reset.*

7.1.2 Done (DONE)

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:**
  1 = Low Energy (LE)
  2 = General

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

  Command_Type = 2 (General):
    0 = ATCS?
    1 = ATLETXT

2 = ATLERXT

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 BR-SS-S2x module transitioning from the Advertising (ATDSLE) State to the Connected State will not send a DONE event when advertising is automatically cancelled to allow connection, but will just send the CONNECT event instead.*

7.1.3 Connect (CONNECT)

CONNECT EVENT

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

Event Format: CONNECT,<Conn_Handle>,<Pairing_Status>,<Addr>

Event Values:

- **Conn_Handle:** Connection handle
- **Pairing_Status:**
  0 = Not Paired
  1 = Paired, Unauthenticated
  2 = Paired, Authenticated
- **Addr:** The address of the connected device.

Example(s):

1. The ATDMLE command is used to initiate an LE connection and once connected the CONNECT event is sent. The connected device is an LE device, that is not paired with the module and has an address of D093F8FF0001:

```
COMMAND:   ATDMLE,D093F8FF0001,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,0,0,D093F8FF0001<cr_lf>
```

7.1.4 Disconnect (DISCONNECT)

DISCONNECT EVENT

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

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

Event Values:

- **Conn_Handle:** Connection handle

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

Example(s):

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

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

7.1.5 Discovery (DISCOVERY)

DISCOVERY EVENT

Function: The discovery event will occur when a device is discovered after the 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>,<Addr>,<Addr_Type>,<RSSI>,
<Data_Structure_Count>,<Data_Structures>

Event Values:

- **Discovery_Type:**
  2 = Low Energy Connectable Indirect Advertisement (Connectable + Discoverable)
  3 = Low Energy Connectable Direct Advertisement (Connectable Only)
  4 = Low Energy Discoverable Indirect Advertisement (Discoverable Only)
  5 = Low Energy Non-connectable Indirect Advertisement (Not Connectable or Discoverable)
  6 = Low Energy Scan Response

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

- **Addr_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 ATDILE command is used to start an LE discovery and two devices are discovered, D093F8FF0001 and D093F8FF0002. D093F8FF0001 is advertising Connectable + Discoverable, 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 3 ad data structures (Flags, TX Power, Slave Connection Interval Range), and the scan data contains 1 ad structure (Complete Local Name). D093F8FF0002 is advertising Connectable only, so only one DISCOVERY event is sent for advertising data:

```
COMMAND:   ATDILE<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           DISCOVERY,2,D093F8FF0001,0,-80,3,020106-020A00-05120A001400
           <cr_lf>
EVENT:     <cr_lf>
           DISCOVERY,6,D093F8FF0001,0,-80,1,
           0F095353312B31424C45464630303031<cr_lf>
EVENT:     <cr_lf>
           DISCOVERY,3,D093F8FF0002,0,-80,3,020106-020A00-05120A001400
           <cr_lf>
EVENT:     <cr_lf>
           DONE,1,1<cr_lf>
```

7.1.6 Pairing Request (PAIR_REQ)

PAIRING REQUEST EVENT

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

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

Event Values:

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

- **Pairing_Type:**

1 = Low Energy

Example(s):

1. An LE pairing request is received from D093F8FF0001 and ATPLE is used to accept the request:

```
EVENT:    <cr_lf>
          PAIR_REQ,D093F8FF0001,1<cr_lf>
COMMAND:  ATPLE,0,1<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,D093F8FF0001,1<cr_lf>
```

7.1.7 Paired (PAIRED)

PAIRED EVENT

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

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

Event Values:

- **Addr:** The address of the 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,D093F8FF0001,1<cr_lf>
```

7.1.8 Pairing Failed (PAIR_FAIL)

PAIRING FAILED EVENT

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

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

Event Values:

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

- **Reason:**
  0 = Pairing Timeout
  1 = Invalid Passkey
  3 = IO Capabilities Cannot Meet Authentication Requirements
  5 = Pairing Not Supported
  6 = Encryption Key Size (If previously paired, then the remote device has unpaired, and pairing will need to be reinitiated.)
  7 = Pairing List Error
  8 = 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,D093F8FF0001,5<cr_lf>
```

7.1.9 Passkey Request (PK_REQ)

PASSKEY REQUEST EVENT

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

Event Format: PK_REQ,<Addr>

Event Values:

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

Example(s):

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

```
COMMAND:   ATPLE,0<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           PK_REQ,D093F8FF0001<cr_lf>
COMMAND:   ATPKR,D093F8FF0001,123456<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           PAIRED,D093F8FF0001,1<cr_lf>
```

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

7.1.10 Passkey Display (PK_DIS)

PASSKEY DISPLAY EVENT

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

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

Event Values:
- **Addr:** The address of the remote device.
- **Passkey:** The passkey to be displayed. 6 numeric characters.

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

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PK_DIS,D093F8FF0001,123456<cr_lf>
EVENT:    <cr_lf>
          PAIRED,D093F8FF0001,1<cr_lf>
```

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

7.2 Low Energy Events

7.2.1 Connection Parameter Update Status (SCCPS)

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.*

7.2.2 Connection Parameter Update (CPU)

CONNECTION PARAMETER UPDATE EVENT

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

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

Event Values:

- **Conn_Handle:** Connection handle

- **Conn_Interval:** Integer value from 6 to 3200 [1.25ms].

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

- **Supervision_Timeout:** Integer value from 10 to 3200 [10ms].

Example(s):

1. A connection parameter update is requested on connection handle 0. 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.*

7.2.3 GATT Done (GATT_DONE)

GATT DONE EVENT

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

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

Event Values:

- **Conn_Handle:** Connection handle

- **Completed_Command:**
  0 = ATGDPS/ATGDPSU
  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>
```

7.2.4 GATT Discovered Primary Service (GATT_DPS)

GATT DISCOVERED PRIMARY SERVICE EVENT

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

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

Event Values:

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

Example(s):

1. ATGDPS is issued for connection handle 0 and 3 primary services are discovered: GAP, GATT, and BAS (Battery). 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,0,0001,0011,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,0012,0014,1801<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,0016,0019,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<cr_lf>
```

7.2.5 GATT Discovered Characteristic (GATT_DC)

GATT DISCOVERED CHARACTERISTIC EVENT

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

Event Format: GATT_DC,<Conn_Handle>,<Char_Value_Att_Handle>,<Char_Properties_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,0,0003,02,2A00<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,2,00<cr_lf>
```

7.2.6 GATT Discovered Characteristic Descriptor (GATT_DCD)

GATT DISCOVER CHARACTERISTIC DESCRIPTOR EVENT

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

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

Event Values:

- **Conn_Handle:** Connection handle
- **Char_Value_Att_Handle:** Characteristic descriptor attribute handle. [1-65535]
- **Char_Desc_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,0,0015,2902<cr_lf>
EVENT:     <cr_lf>
           GATT_DCD,0,0019,2902<cr_lf>
EVENT:     <cr_lf>
           GATT_DCD,0,0029,2902<cr_lf>
EVENT:     <cr_lf>
           GATT_DCD,0,0024,2902<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,3,00<cr_lf>
```

7.2.7 GATT Characteristic/Descriptor Value (GATT_VAL)

GATT CHARACTERISTIC/DESCRIPTOR VALUE EVENT

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

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

Event Values:

- **Conn_Handle:** Connection handle.
- **Value_Att_Handle:** Value attribute handle. [1-65535]
- **Value_Event_Type:**
  0 = Read Response
  1 = Notification
  2 = Indication
- **Value_Length:** Length of characteristic value in bytes.
- **Value:** Value formatted in hex.

Example(s):

1. ATGR is used to read a value from handle 3. When the operation is complete a GATT_VAL is

triggered, returning a 16 byte value:

```
COMMAND:   ATGR,0,3<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_VAL,0,0003,0,16,426C7565526164696F73303030303031<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,4,00<cr_lf>
```

7.2.8 GATT Read Request (GRREQ)

GATT READ REQUEST EVENT

Function: This event will be sent when a remote device attempts to read a characteristic or descriptor that is part of a published GATT service.

Event Format: GRREQ,<GRS_Handle>,<Conn_Handle>,<RequestID>,
<Characteristic_Descriptor_Offset>,<Attribute_Value_Offset>

Event Values:

- **GRS_Handle:** The handle used to add Includes, Characteristics and Descriptors to the GATT Service. This is also used to Commit and De-commit the Service.

- **Conn_Handle:** Connection Handle of the device making the Connection Request.

- **RequestID:** Request Identifier that is used to respond to the read request.

- **Characteristic_Descriptor_Offset:** Offset of the Characteristic where the request is being made. Note that this is the offset into the table created using ATGRS. As such the Service Declaration is always offset 0, and Characteristics use two attribute entries (1 for the Characteristic Declaration, 1 for the Characteristic Value). Note that this event will only be made to read Characteristic Values and Descriptors. All other read requests will be handled internally. Descriptors use 1 attribute entry. For a Service with 1 Characteristic and 1 Descriptor, the offsets would be as follows:
  Attribute Offset 0 = Service Declaration
  Attribute Offset 1 = Characteristic Declaration
  Attribute Offset 2 = Characteristic Value
  Attribute Offset 3 = Characteristic Descriptor

- **Attribute_Value_Offset:** Number of Bytes into the Attribute Value to return.

Example(s):

1. Remote device is attempting to read from the service with GRS Handle 0 from Offset 2. The RequestID of 0 is used to respond to the Read Request.

```
EVENT:     <cr_lf>
           GRREQ,0,0,2<cr_lf>
```

Note(s):

- *This event is dispatched whenever a Client attempts to do a GATT Read Request unless a remote device tries to read a characteristic or descriptor that is not readable.*

- *This event can be responded to with the ATGRR command.*

7.2.9 GATT Write Request (GWREQ)

GATT WRITE REQUEST EVENT

Function: This event will be sent when a remote device attempts to write to characteristic or descriptor that is part of a published GATT service.

Event Format: GWREQ,<GRS_Handle>,<Conn_Handle>,<RequestID>,
<Characteristic_Descriptor_Offset>,<HexValue>

Event Values:

- **GRS_Handle:** The handle used to add Includes, Characteristics and Descriptors to the GATT Service. This is also used to Commit and De-commit the Service.

- **Conn_Handle:** Connection Handle of the device making the Connection Request

- **RequestID:** Request Identifier that is used to respond to the Write Request.

- **Characteristic_Descriptor_Offset:** Offset of the Characteristic where the request is being made. Note that this is the offset into the table created using ATGRS. As such the Service Declaration is always offset 0, and Characteristics use two attribute entries (1 for the Characteristic Declaration, 1 for the Characteristic Value). Note that this event will only be made to write Characteristic Values and Descriptors. All other read requests will be handled internally. Descriptors use 1 attribute entry. For example, a Service with 1 Characteristic and 1 Descriptor, the offsets would be as follows:
  Attribute Offset 0 = Service Declaration
  Attribute Offset 1 = Characteristic Declaration
  Attribute Offset 2 = Characteristic Value
  Attribute Offset 3 = Characteristic Descriptor

- **HexValue:** The value that the Client is attempting to write, formatted as a Hex String.

Example(s):

1. Remote device is attempting to write from the service with GRS Handle 0 at Offset 2. The RequestID of 0 is used to respond to the Write Request.

```
EVENT:    <cr_lf>
          GWREQ,0,0,2,48656C6C6F20576F726C64<cr_lf>
```

Note(s):

- *This event is dispatched whenever a Client attempts to do a GATT Write Request unless the Client attempts to write to a characteristic or descriptor that is not writable.*

- *This event can be responded to with the ATGWR command.*

7.2.10 GATT Write Without Response (GWORP)

GATT WRITE WITHOUT RESPONSE EVENT

Function: This event will be sent when a remote device attempts to write to characteristic or descriptor that is part of a published GATT service using the GATT Write Without Response procedure. There is no response to this event and the host is not required to actually perform the write.

Event Format: GWORP,<GRS_Handle>,<Conn_Handle>,
<Characteristic_Descriptor_Offset>,<HexValue>

Event Values:

- **GRS_Handle:** The handle used to add Includes, Characteristics and Descriptors to the GATT Service. This is also used to Commit and De-commit the Service.

- **Conn_Handle:** Connection Handle of the device making the Connection Request

- **Characteristic_Descriptor_Offset:** Offset of the Characteristic where the request is being made. Note that this is the offset into the table created using ATGRS. As such the Service Declaration is always offset 0, and Characteristics use two attribute entries (1 for the Characteristic Declaration, 1 for the Characteristic Value). Note that this event will only be made to write Characteristic Values and Descriptors. All other read requests will be handled internally. Descriptors use 1 attribute entry. For example, a Service with 1 Characteristic and 1 Descriptor, the offsets would be as follows:
  Attribute Offset 0 = Service Declaration
  Attribute Offset 1 = Characteristic Declaration
  Attribute Offset 2 = Characteristic Value
  Attribute Offset 3 = Characteristic Descriptor

- **HexValue:** The value that the Client is attempting to write, formatted as a Hex String.

Example(s):

2. Remote device is attempting to write to the service with GRS Handle 0 at Offset 2.

```
EVENT:    <cr_lf>
          GWORP,0,2,48656C6C6F20576F726C64<cr_lf>
```

Note(s):

- *This event is dispatched whenever a Client attempts to do a GATT Write Request unless the Client attempts to write to a characteristic or descriptor that is not writable.*

- *This event can be responded to with the ATGWR command.*

7.2.11 GATT Server Indication Confirmation (GSCONF)

GATT CONFIRMATION EVENT

Function: This event will be dispatched when a Confirmation from a previously sent Indication is received from a remote device.

Event Format: GSCONF,<GRS_Handle>,<Conn_Handle>,<Status>

Event Values:

- **GRS_Handle:** Handle of the Service containing the Characteristic Value that was indicated.
- **Conn_Handle:** Connection Handle of the device that send the Confirmation.
- **Status:** Status of the Confirmation. Will be one of the following values:
  0 = Confirmation was received successfully
  1 = Confirmation was not received within the timeout window

Example(s):

1. Remote device with Connection Handle 0 sent Confirmation for Indication on GRS Handle 0 (so the confirmation is successful.

```
EVENT:     <cr_lf>
           GSCONF,0,0,0<cr_lf>
```

7.2.12 Address Resolved (RESOLVED)

RESOLVED EVENT

Function: The address change event will occur when the address of device that is using a random address is resolved to their public or static address.

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

Event Values:

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

Example(s):

1. The ATDILE command is used to start an LE discovery. A single device is discovered, with a resolvable private address of 7E06AC26DB6A. Since this device was previously paired the resolvable private address is resolved to the public address of D093F8FF0000.

```
COMMAND:   ATDILE<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           DISCOVERY,3,7E06AC26DB6A,0,-45,4,020106-020A04-051208000800-
           1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
           <cr_lf>
           RESOLVED,7E06AC26DB6A,D093F8FF0000<cr_lf>
           <cr_lf>
           DONE,1,1<cr_lf>
```

8 General Commands

8.1 AT (AT)

AT

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

Example(s):

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

8.2 Reset Commands

8.2.1 Reset (ATRST)

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>
               SS1-LE4.0-1BLE<cr_lf>
```

Note(s):

- *This command does not return an OK.*
- *During a reset, the TX line of the UART will pulse low, which may be read as a NULL character showing up before the <cr_lf> on some receivers.*

8.2.2 Factory Reset (ATFRST)

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>
           SS1-LE4.0-1BLE<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.*

8.3 TBD

8.4 Module Information Commands

8.4.1 Module Type (ATMT?)

GET MODULE TYPE

Function: Gets the module type.

Command Format: ATMT?

Response Format: <Module_Type>

Response Values:

- **Module_Type:**

SS1-LE4.0-1BLE	Single Mode LE Module

Example(s):

```
COMMAND:   ATMT?<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
           <cr_lf>
           SS1-LE4.0-1BLE<cr_lf>
```

8.4.2 Stack Type (ATST?)

GET STACK TYPE

Function: Gets the stack type.

Command Format: ATST?

Response Format: Stonestreet One 1BLE

Example(s):
```
    COMMAND:   ATST?<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
               <cr_lf>
               Stonestreet One 1BLE<cr_lf>
```

8.4.3 Firmware Version (ATV?)

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 three digits after "d" should match this document's version.

Example(s):
```
   COMMAND:   ATV?<cr>
   RESPONSE:  <cr_lf>
              OK<cr_lf>
              <cr_lf>
              4.0.1.9-d1.1.0-1BLE<cr_lf>
```

8.4.4 Bluetooth Device Address (ATA?)

GET BLUETOOTH DEVICE ADDRESS

Function: Gets the module's Bluetooth Device Address.

Command Format: ATA?

Response Format: <Addr>

Response Values:
- **Addr:** Local device address

Example(s):
```
    COMMAND:   ATA?<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
               <cr_lf>
               D093F8FF0001<cr_lf>
```

8.4.5 Bluetooth Device Name (ATSN)

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 - 1-20 characters
  Default: SS1+1BLE<Bluetooth Device Address Lower Address Part>

Example(s):

```
COMMAND:   ATSN,Stonestreet One123456<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>
           OK<cr_lf>
           Stonestreet One123456<cr_lf>
```

8.5 Module Status Commands

8.5.1 Connection Status (ATCS?)

GET CONNECTION STATUS

Function: Gets the details of an existing connection.

Command Format: ATCS?,<Conn_Handle>

Command Parameter(s):

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

Response Format: <Conn_Handle>,<Pairing_Status>,<Addr>

Response Values:

- **Conn_Handle:** Connection handle

- **Pairing Status:**
  0 = Not Paired
  1 = Paired, Unauthenticated
  2 = Paired, Authenticated

- **Addr:** Address of remote device

Example(s):

1. One active connection is found, it is an unpaired LE connection to D093F8FF000. Then DONE event is triggered to signal the end of the connection list.

```
COMMAND:  ATCS?<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0,0,D093F8FF0000<cr_lf>
          <cr_lf>
          DONE,2,0<cr_lf>
```

8.5.2 RSSI (ATRSSI?)

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>
              -34<cr_lf>
```

Note(s):

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

8.6 Configuration Control Commands

8.6.1 Configuration Lock (ATSCL)

SET CONFIGURATION LOCK

Warning: Be careful when locking the configuration. The password cannot be obtained after it has been changed and a reload of the application software 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.

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 and cannot be used to reset the password. Only reloading the application software will do so.*

GET CONFIGURATION LOCK

Function: Gets the configuration lock setting.

Command Format: ATSCL?

Response Format: <Lock>

Example(s):
```
    COMMAND:   ATSCL?<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
               <cr_lf>
               0<cr_lf>
```

8.6.2 Flash Configuration (ATFC)

FLASH CONFIG

- **Function:** Stores the current configuration in flash. This command is intended to be used after all settings have been configured. If 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 it is to disable Auto_Flash, make all of the changes, then use ATFC to store them.

Command Format: ATFC

Example(s):
```
    COMMAND:   ATFC<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
```

Note(s):
- *ATFC can still be used once the configuration has been locked.*

8.6.3 Configure Auto Configuration Flashing (ATSFC)

SET FLASH CONFIGURATION

Function: Sets the configuration flashing settings.

Command Format: ATSFC,<Auto_Flash>

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

Example(s):

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

GET FLASH CONFIGURATION

Function: Gets the configuration flashing settings.

Command Format: ATSFC?

Response Format: <Auto_Flash>

Example(s):

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

8.7 Response Configuration Commands

8.7.1 Discovery Event Formatting (ATSDIF)

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
65535 (0xFFFF)	= All Data Structures Enabled

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

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

Example(s):

```
COMMAND:   ATSDIF,65535,1,1<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 advertising 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>
           65535,1,1<cr_lf>
```

8.8 Hardware Configuration / Control Commands

8.8.1 Set Power Level (ATSPL)

SET POWER LEVEL

Function: Sets the transmit power of the module.

Command Format: ATSPL,<Transmit_Power>

Command Parameter(s):

- **Transmit_Power:** Integer value from 0 to 7.
  0 = -18 dBm
  1 = -15 dBm
  2 = -12 dBm
  3 = -9 dBm
  4 = -6 dBm
  5 = -3 dBm
  6 = 0 dBm
  7 = 3 dBm

Example(s):

```
    COMMAND:   ATSPL,6<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
```

GET POWER LEVEL

Function: Gets the transmit power of the module.

Command Format: ATSPL?

Response Format: <Transmit_Power>

Example(s):

```
    COMMAND:   ATSPL?<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
               <cr_lf>
               6<cr_lf>
```

8.8.2 UART Configuration (ATSUART)

SET UART

Function: Configures the module's UART.

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

Command Parameter(s):

- **Baud_Rate:** 0-10 [300bps – 921600bps], enter Value from table below.

Baud rate	Value
300	0
1200	1
2400	2
9600	3
19200	4
38400	5
57600	6
115200	7
230400	8
460800	9
921600	10

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

- **Stop_Bits:**

0 = One
1 = Two

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

Example(s):

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

Note(s):

- *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.*

GET UART

Function: Gets the UART configuration.

Command Format: ATSUART?

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

Example(s):

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

8.8.3 PIO Configuration (ATSPIO)

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:** 2-19

PIO_Num	Pin Name	PIO_Num	Pin Name
2	PIO_2	11	UART_CTS
3	PIO_3	12	UART_RTS
4	PIO_4/TDI	13	IC2_SCL

5	PIO_5	14	PIO_14
6	PIO_6/TMS	15	SPI_MISO
7	PIO_7/TCK	16	SPI_CSB
8	PIO_8/TDO	17	SPI_CLK
9	PIO_9/NMI	18	I2C_SDA
10	SPI_MOSI	19	PIO_20

- **Direction:**
  0 = Input
  1 = Output

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

Example(s):

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

Note(s):

- *PIO_Num 6-12, 15, and 17 cannot be modified.*

GET PIO

Function: Reads PIO settings.

Command Format: ATSPIO?,<PIO_Num>

Command Parameter(s):

- **PIO_Num:** 2-19

Response Format: <Direction>,<Value>

Example(s):

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

8.9 Cancel Command (ATDC)

CANCEL

Function: This command tells the radio to cancel commands that run in the background such as ATDMLE, ATDSLE, or ATDILE. This command can come in handy for a quick exit from commands like ATDILE 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:**
  1 = Low Energy (LE)
  2 = General

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

  Command_Type = 2 (General):
  0 = ATCS?
  1 = ATLETXT
  2 = ATLERXT

Example(s):

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

8.10 Disconnect Command (ATDH)

DISCONNECT

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

Command Format: ATDH,<Conn_Handle>

Command Parameter(s):

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

Example(s):

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

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

8.11 Authentication Commands

8.11.1 Passkey Response (ATPKR)

PASSKEY RESPONSE

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

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

Command Parameter(s):
- **Addr_Conn_Handle:** Address of device or the connection handle
- **Passkey:** 6 numeric characters

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

```
COMMAND:  ATPLE,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PK_REQ,D093F8FF0001<cr_lf>
COMMAND:  ATPKR,D093F8FF0001,123456<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          PAIRED,0,1<cr_lf>
```

8.12 Utility Commands

8.12.1 LE Receiver Test (ATLERXT)

LE RECEIVER TEST

Function: Issues the HCI_LE_Receiver_Test Command to the Bluetooth Controller. This command is used to start a test where the Device Under Test (DUT) receives test reference packets at a fixed interval. The tester generates the test reference packets.

Command Format: ATLERXT,<RX_Frequency>

Command Parameter(s):
- **RX_Frequency:** 0-39 (N) which are the LE channels representing frequencies 2402 MHz – 2480 MHz (F) using the formula: N = (F – 2402) / 2.

Example(s):

1. Test receiving reference packets at 2458 MHz.

```
COMMAND:  ATLERXT,28<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

8.12.2 LE Transmitter Test (ATLETXT)

LE TRANSMITTER TEST

Function: Issues the HCI_LE_Transmitter_Test Command to the Bluetooth Controller. This command is used to start a test where the Device Under Test (DUT) generates test reference packets at a fixed interval.

Command Format: ATLETXT,<TX_Frequency>,<Length_Of_Test_Data>,<Packet_Payload>

Command Parameter(s):

- **TX_Frequency:** 0-39 (N) which are the LE channels representing frequencies 2402 MHz – 2480 MHz (F) using the formula: N = (F – 2402) / 2.
- **Length_Of_Test_Data:** 0-37. Length in bytes of payload data in each packet.
- **Packet_Payload:** Format of the payload in each packet. Must be one of the following:
  0 = Pseudo-Random bit sequence 9
  1 = Pattern of alternating bits 11110000
  2 = Pattern of alternating bits '10101010'
  3 = Pseudo-Random bit sequence 15
  4 = Pattern of All '1' bits
  5 = Pattern of All '0' bits
  6 = Pattern of alternating bits '00001111
  7 = Pattern of alternating bits '0101'

Example(s):

1. Test transmitting reference packets of payload length 8 bytes with an alternating bit pattern at 2480 MHz.

```
COMMAND:  ATLETXT,39,8,2<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

8.12.3 LE End Test (ATLEENDT)

LE END TEST

Function: Issues the HCI_LE_Test_End command to the Bluetooth Controller, stopping any test which is in progress (i.e. an ongoing LE Receiver or Transmitter Test).

Command Format: ATLEENDT

Response Format: <Packets_Received>

Example(s):

1. Module #1 (serving as a tester) starts a transmitter test with 1 byte length packets with a pseudo-random bit sequence 9 pattern at 2412 MHz. Module #2 starts a receiver test to receive packets at 2412 MHz. Module #2 then ends its testing and reports the number of packets it received. Module #1 then ends its testing (as a transmitter, it will report 0 packets received).

```
Module #1 COMMAND:  ATLETXT,5,1,0<cr>
Module #1 RESPONSE: <cr_lf>
                    OK<cr_lf>

Module #2 COMMAND:  ATLERXT,5<cr>
Module #2 RESPONSE: <cr_lf>
                    OK<cr_lf>

Module #2 COMMAND:  ATLEENDT<cr>
Module #2 RESPONSE: <cr_lf>
                    OK<cr_lf>
                    <cr_lf>
                    <Packets_Received><cr_lf>

Module #1 COMMAND:  ATLEENDT<cr>
Module #1 RESPONSE: <cr_lf>
                    OK<cr_lf>
                    <cr_lf>
                    0<cr_lf>
```

Note(s):

- *The number of packets received will depend on how long the test is run and cannot exceed 65535.*

9 Low Energy Commands

9.1 Module Information Commands

9.1.1 TBD

9.2 LE Status Commands

9.2.1 LE State (ATSLE?)

GET STATE LE

Function: Gets the current LE state of the module.

Command Format: ATSLE?

Response Format: <Standby_Testing>,<Advertising>,<Scanning>,<Initiating>,<Connected>

Response Values:

- **Standby_Testing:**
  0 = Not Standby
  1 = Standby
  2 = Testing
- **Advertising:**
  0 = Not Advertising
  1 = Advertising (ATDSLE)
- **Scanning:**
  0 = Not Scanning/Discovering
  1 = Scanning/Discovering (ATDILE)
- **Connecting:**
  0 = Not Initiating
  1 = Initiating (ATDMLE)
- **Connected:**
  0 = Not Connected
  1 = Connected

Example(s):

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

9.2.1 LE Last Connected Address (ATLCALE?)

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: <Addr>

Response Values:

- **Addr:** Slave device address

Example(s):

```
COMMAND:   ATLCALE?<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
           <cr_lf>
           D093F8FF0001<cr_lf>
```

9.3 TBD

9.4 Advertising Commands

9.4.1 Advertise (ATDSLE)

DIAL AS SLAVE (ADVERTISE)

Function: This command places the module in the Advertising State, allowing it to be discovered by other LE capable devices. Depending on the ATSDSLE Ad_Type, this can also make the module discoverable 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 1 master at a time.*
- *The BR-SS-S2x Bluetooth chip's Link Layer supports a single state machine. Thus this command can only be executed when in the Standby State (i.e. it will fail if the module is in the Advertising, Scanning, Initiating, or Connected State).*

9.4.2 Advertise Direct (ATDSDLE)

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 ATSDSLE Ad_Timeout setting.

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

Command Parameter(s):

- **Addr:** Master address for connectable direct advertising. If not specified the module will advertise using the Ad_Type set in ATSDSLE.
- **Address Type:**
  0 = Public Address
  1 = Static Address
  2 = Non-Resolvable Private Address
  3 = Resolvable Private Address

Example(s):

```
    COMMAND:   ATDSDLE,D093F8FF0000<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
               <cr_lf>
               CONNECT,0,0,D093F8FF0000<cr_lf>

    COMMAND:   ATDMLE,D093F8FF0000<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
               <cr_lf>
               DONE,1,0<cr_lf>
```

Note(s):

- *This command overrides the ATSDSLE Ad_Type setting to 1, connectable direct advertisement.*

9.4.3 Advertising Configuration (ATSDSLE)

SET ADVERTISING

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

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

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 + Discoverable)
  1 = *This setting is overridden by ATDSDLE which is used for connectable direct advertising*
  2 = Discoverable indirect advertisement (Discoverable Only)
  3 = Non-connectable indirect advertisement (Neither Connectable or Discoverable)

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

Example(s):

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

Note(s):

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

GET ADVERTISING

Function: Gets the advertising (ATDSLE) settings.

Command Format: ATSDSLE?

Response Format: <White_List_Filter>,<Ad_Type>,<Ad_Channel_Map>

Example(s):

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

9.4.4 Advertising Timing Configuration (ATSDSTLE)

SET ATDSLE TIMING PARAMETERS

Function: Sets a slave module's advertising 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].
  Default: 160 (100ms)
- **Connected_Ad_Interval:** Integer value from 160 to 16384 [.625ms].
  Default: 2048 (1280ms)

Example(s):

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

Note(s):

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

GET ATDSLE TIMING PARAMETERS

Function: Sets the module's advertising timing parameters.

Command Format: ATSDSTLE?

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

Response Values: See ATSDSTLE.

Example(s):
```
    COMMAND : ATSDSTLE?<cr>
    RESPONSE: <cr_lf>
              OK<cr_lf>
              <cr_lf>
              0,160,2048<cr_lf>
```

9.4.5 Advertising Data (ATSDSDLE)

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 Stonestreet One, 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 Stonestreet One 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.

Example(s):

1. An ad structure of 7 bytes, with an Ad Type of FF (Manufacturer Specific Data) and 6 bytes of data (Stonestreet One 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 Stonestreet One CID of 0x0085, but the next byte was not set to 0xFF to specify Stonestreet One 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: ATSDSDLE?,<Data_Type>

Response Format: <Data>

Example(s):

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

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

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

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

9.5 LE Discovery Commands

9.5.1 LE Discovery (ATDILE)

DISCOVERY LE

Function: This command is used to scan for 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 scanning for LE devices and two devices are discovered, D093F8FF0001 and D093F8FF0002. D093F8FF0001 is advertising Connectable + Discoverable, 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, and Interval Range), and the scan data contains 1 ad structure (Complete Local Name). D093F8FF0002 is advertising Connectable only, so only one DISCOVERY event is sent for advertising data:

```
COMMAND:  ATDILE<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,2,D093F8FF0001,0,-045,4,020106-020A04-051208000800-
          1107796022A0BEAFC0BDDE487962F1842BDA<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,6,D093F8FF0001,0,-045,1,
          1109426C7565526164696F7330303030303031<cr_lf>
EVENT:    <cr_lf>
          DISCOVERY,3,D093F8FF0002,0,-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 discovered, or an inquiry timeout has occurred while scanning 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.*

- *The BR-SS-S2x Bluetooth chip's Link Layer supports a single state machine. Thus this command can only be executed when in the Standby State (i.e. it will fail if the module is in the Advertising, Scanning, Initiating, or Connected State).*

9.5.2 LE Discovery Configuration (ATSDILE)

SET DISCOVERY CONFIGURATION 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 White List can be configured using the ATSWL command.
  0 = Disabled
  1 = Scan for White List devices only.

- **Disc_Mode_Filter:**
  0 = Disabled (Scan for all discoverable devices)
  1 = Scan for 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.

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 Scanning 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>
           0,0,1,10<cr_lf>
```

9.5.3 LE Discovery Timing Configuration (ATSDITLE)

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] or 0. If 0 is used the module with continuously scan.
  Default: 10240 (10240ms)

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

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

Example(s):

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

Note(s):

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

GET DISCOVERY TIMING LE

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

Command Format: ATSDITLE?

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

Example(s):

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

9.6 LE Connect Commands

9.6.1 LE Connect (ATDMLE)

DIAL AS MASTER LE (CONNECT)

Function: This command initiates a connection to a slave device. If the command is accepted, an **"OK"** will be sent back the module will be in the Initiating State (not yet connected). A completed connection will return a CONNECT event sometime after the command was sent.

Command Format: ATDMLE,<Addr_WL>,<Addr_Type>

Command Parameter(s):

- **Addr_WL:** Slave device address or set to "WL" to use the White List instead of a direct connection.

- **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 D093F8FF0001:

```
COMMAND:   ATDMLE,D093F8FF0001<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           CONNECT,0,0,D093F8FF0001<cr_lf>
```

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

```
COMMAND:   ATDMLE,D093F8FF0001<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.*

- *The BR-SS-S2x module can connect (in the LE master role) to 1 LE slave.*

- *The BR-SS-S2x Bluetooth chip's Link Layer supports a single state machine. Thus this command can only be executed when in the Standby State (i.e. it will fail if the module is in the Advertising, Scanning, Initiating, or Connected State).*

9.6.2 LE Connect Last (ATDMLLE)

DIAL AS MASTER LAST LE (CONNECT LAST)

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

Command Format: ATDMLLE

Example(s):

```
COMMAND:   ATDMLLE<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
           <cr_lf>
           CONNECT,0,0,D093F8FF0001<cr_lf>
```

Note(s):

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

9.6.3 LE Connect Timing Configuration (ATSDMTLE)

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

Example(s):

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

Note(s):

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

GET CONNECT TIMING LE

Function: Gets the connection initiation timing parameters.

Command Format: ATSDMTLE?

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

Example(s):

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

9.7 Connection Parameters

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

- **Connection Interval –** The amount of time between two connection events. A shorter connection interval will result in higher throughput, lower latency, but will use more power, and a longer connection interval will result in lower throughput, higher latency and less power consumption.

- **Slave Latency –** Allows a slave to skip a set number of connection events. This allows the slave to save power if it has no data to send. This will not affect the latency of data sent from slave to master, but will result in increased latency from master to slave. The slave latency must not make the effective connection interval greater than 32s.

- **Supervision Timeout –** Amount of time allowed since the last successful connection event. If this timeout occurs the connection will be dropped.

- **Effective Connection Interval –** Amount of time between connection events, assuming the slave always skips [slave latency] connection events. It can be calculated with the following formula:

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

9.7.1 Default Connection Parameters (ATSDCP)

SET DEFAULT CONNECTION PARAMETERS

Function: Sets the module's default connection parameters.

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

Command Parameter(s):

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

- **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 after the link is encrypted or after a configurable delay (configured using ATSAU.

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>
               8,16,0,400,0<cr_lf>
```

9.7.2 Current Connection Parameters (ATSCCP)

SET CURRENT CONNECTION PARAMETERS

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

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

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

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

Command Parameter(s):

- **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>
          8,0,400<cr_lf>
```

9.8 LE Pairing Commands

9.8.1 LE Pair Command (ATPLE)

PAIR DEVICE LE

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

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

Command Parameter(s):

- **Addr_Conn_Handle:** Address of device or the connection handle.

- **Accept_Request:** (Only used 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,D093F8FF0001,1<cr_lf>
```

Note(s):

- *Up to 30 devices can be paired at a time.*

GET PAIRED DEVICES LE

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

Command Format: ATPLE?

Response Format: <Count>,<Addrs>

Response Value(s):

- **Count:** Number of paired devices

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

Example(s):

1. The module isn't paired to any devices.

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

2. The module has been paired to two devices.

```
COMMAND:   ATPLE?<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
           <cr_lf>
           2,D093F8FF0001-D093F8FF0002<cr_lf>
```

9.8.2 LE Pairing Configuration (ATSPLE)

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

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>
```

9.8.3 LE Unpair Device (ATUPLE)

UN PAIR DEVICE LE

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

Command Format: ATUPLE,<Addr>

Command Parameter(s):

- **Addr:** Device address

Example(s):

```
    COMMAND:   ATUPLE,D093F8FF0001<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
```

9.8.4 LE Clear Pair List (ATCPLE)

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.*

9.8.5 Fixed Passkey (ATSPK)

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.
  Default: 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>
```

9.9 White List Commands

9.9.1 White List Device (ATSWL)

SET WHITE LIST

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

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

Command Parameter(s):

- **Addr:** Device address

- **Addr_Type:**

0 = Public Device Address
1 = Random Device Address

Example(s):

```
COMMAND:   ATSWL,D093F8FF0001<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

Note(s):

- *Up to 4 devices can be added to the White List.*

GET WHITE LIST

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

Command Format: ATSWL?

Response Format: <Count>,<Addrs>

Response Value(s):

- **Count:** Number of device in the White List
- **Addrs:** List of addresses in the White List, separated by '-' characters. 0 if Count is 0.

Example(s):

1. The module doesn't have any device in its White List.

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

2. The module has two devices in its White List.

```
COMMAND:   ATSWL?<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
           <cr_lf>
           2,D093F8FF0001-D093F8FF0002<cr_lf>
```

9.9.2 Un White List Device (ATUWL)

UN WHITE LIST DEVICE

Function: This command is used to remove a device from the White List.

Command Format: ATUWL,<Addr>

Command Parameter(s):

- **Addr:** Device address

Example(s):

```
    COMMAND:   ATUWL,D093F8FF0001<cr>
    RESPONSE:  <cr_lf>
               OK<cr_lf>
```

Note(s):

- *This command cannot be used if any of the following is true:*
  - *The advertising filter policy uses the white list and advertising is enabled*
  - *The initiator filter policy uses the white list and a create connection command is outstanding*

9.9.3 Clear White List (ATCWL)

CLEAR WHITE LIST

Function: This command is used to remove all devices from the White List.

Command Format: ATCWL

Example(s):

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

Note(s):

- *This command cannot be used if any of the following is true:*
  - *The advertising filter policy uses the white list and advertising is enabled*
  - *The initiator filter policy uses the white list and a create connection command is outstanding*

9.10 GATT Commands

The GATT commands allow the module to use the Generic Attribute Profile (GATT) to discover and use the services on a remote device. GATT commands cannot be cancelled using the ATDC command.

9.10.1 Discover All Primary Services (ATGDPS)

GATT DISCOVER ALL PRIMARY SERVICES

Function: This command is used to discover all the primary services on a server.

Command Format: ATGDPS,<Conn_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle.

Example(s):

1. ATGDPS is issued for connection handle 0 and 3 primary services are discovered: GAP, GATT, and BAS (Battery). When all of the primary services have been discovered a GATT_DONE event is triggered:

```
COMMAND:  ATGDPS,0cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,0001,0011,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,0012,0014,1801<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,0016,0019,180F<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<cr_lf>
```

9.10.2 Discovery Primary Services By UUID (ATGDPSU)

GATT DISCOVER PRIMARY SERVICES BY UUID

Function: This command is used to discover a specific primary service on a server.

Command Format: ATGDPSU,<Conn_Handle>,<UUID>

Command Parameter(s):

- **Conn_Handle:** Connection handle.

- **UUID:** UUID of the service to discover. [16-bit UUID = 4 chars, 128-bit UUID = 32 chars]

Example(s):

1. ATGDPSU is issued for connection handle 0 and the GAP Service. The service is found with a start handle of 1 and an end handle of 11.

```
COMMAND:  ATGDPSU,0,1800<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_DPS,0,0001,0011,1800<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,0,00<cr_lf>
```

9.10.3 Discover All Characteristics (ATGDC)

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.

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_Att_Handle. [1-65535]

Example(s):

1. ATGDC is used to discover all characteristics between handles 1 and 11 (GAP Service), and 5 characteristics are found.

```
COMMAND:   ATGDC,0,1,11<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
EVENT:     <cr_lf>
           GATT_DC,0,0003,02,2A00<cr_lf>
EVENT:     <cr_lf>
           GATT_DC,0,0005,02,2A01<cr_lf>
EVENT:     <cr_lf>
           GATT_DC,0,0007,0A,2A02<cr_lf>
EVENT:     <cr_lf>
           GATT_DC,0,0009,0A,2A03<cr_lf>
EVENT:     <cr_lf>
           GATT_DC,0,0011,02,2A04<cr_lf>
EVENT:     <cr_lf>
           GATT_DONE,0,2,00<cr_lf>
```

Note(s):

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

9.10.4 Discover Characteristics By UUID (ATGDCU)

GATT DISCOVER CHARACTERISTICS BY UUID

Function: This command is used to discover specific characteristics on a server or specific characteristics within a specific attribute handle range.

Command Format: ATGDCU,<Conn_Handle>,<UUID>,<Start_Att_Handle>,<End_Att_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **UUID:** UUID of the characteristic to discover. [16-bit UUID = 4 chars, 128-bit UUID = 32 chars]
- **Start_Att_Handle:** Attribute handle to start searching at, typically the Svc_Att_Handle of a specific service. Start_Att_Handle is included in the search. [1-65535]
- **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_Att_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,0,0003,02,2A00<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,2,00<cr_lf>
```

Note(s):

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

9.10.5 Characteristic Descriptor Discovery (ATGDCD)

GATT DISCOVER ALL CHARACTERISTIC DESCRIPTORS

Function: This command is used to discover all the characteristic descriptors within a characteristic definition. It will return all attributes within the specified handle range. If a range is isn't specified then all attributes will be returned, allowing the entire GATT table to be seen.

Command Format: ATGDCD,<Conn_Handle>,<Start_Att_Handle>,<End_Att_Handle>

Command Parameter(s):

- **Conn_Handle:** Connection handle.
- **Start_Att_Handle:** Attribute handle to start searching at, typically the Char_Value_Att_Handle+1 of a specific characteristic. Start_Att_Handle is included in the search. [1-65535]
- **End_Att_Handle:** Attribute handle to stop searching at, typically the ending handle of the

characteristic. End_Att_Handle is included in the search. Must be greater than or equal to Start_Att_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,0,0015,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DCD,0,0019,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DCD,0,0029,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DCD,0,0024,2902<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,3,00<cr_lf>
```

Note(s):

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

9.10.6 Characteristic Read (ATGR)

GATT READ

Function: This command is used to read specific characteristic values or descriptors from a server when the attribute handle of the characteristic value or descriptor is known.

This command can 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,0,7<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
EVENT:    <cr_lf>
          GATT_VAL,0,0007,0,14,5353312B31424C45303030303031<cr_lf>
EVENT:    <cr_lf>
          GATT_DONE,0,4,00<cr_lf>
```

9.10.7 Characteristic Read By UUID (ATGRU)

GATT READ BY UUID

Function: This command is used to read specific characteristic values or descriptors from a server when the handle of the characteristic value or descriptor is not known.

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_Att_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,0,0007,0,14,5353312B31424C45303030303031<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,0,0007,0,14,SS1+1BLE000001<cr_lf>
    EVENT:     <cr_lf>
               GATT_DONE,0,4,00<cr_lf>
```

Note(s):

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

9.10.8 Characteristic Write (ATGW)

GATT WRITE

Function: This command is used to write a specific characteristic value or descriptor from a server when the attribute handle of the characteristic value or descriptor is known. The server will return a response confirming the value was written, which will trigger a 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>
```

9.10.9 Characteristic Write No Response (ATGWN)

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>
```

9.10.10 Register Service (ATGRS)

GATT REGISTER SERVICE

Function: This command is used to register a dynamic GATT service. The OK from this command will return the GRS Handle that can be used to add attributes to the service before it is published.

Command Format: ATGRS,<Service Type>,<Number of Attributes>,<UUID>

Command Parameter(s):

- **Service Type:** Type of Service to be registered
  0 = Primary Service
  1 = Secondary Service

- **Number of Attributes:** Number of Required Attributes
  Number Of Attributes = <Number Of Includes> +
  <Number Of Characteristics> * 2 + <Number Of Descriptors>

- **UUID:** Service UUID
  16 bit or 128 bit UUID

Example(s):

1. Registering a Primary Service with a 16 bit UUID of 0x1811 that contains 12 attributes

```
COMMAND:  ATGRS,0,12,1811<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          0<cr_lf>
```

2. Registering a Primary Service with a 128 bit UUID of 0x796022A0BEAFC0BDDE487962F1842BDA that contains 12 attributes.

```
COMMAND:  ATGRS,0,12,796022A0BEAFC0BDDE487962F1842BDA<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          1<cr_lf>
```

Note(s):

- *Once this command is sent then Includes, Characteristics and Descriptors may be added to the service using the GRS Handle that is returned in the OK response.*

9.10.11 Add Include (ATGAI)

GATT ADD INCLUDE

Function: This command is used to add an Include Definition to a registered (but not published) service. An Include Definition is a way to reference another service from the service with the Include Definition (so as to create a service hierarchy).

Command Format: ATGAI,<GRS_Handle>,<Attribute Offset>,<GRS_Handle_Included_Service>

Command Parameter(s):

- **GRS_Handle:** GRS_Handle of the Service to add the Include Attribute to.

- **Attribute Offset:** Offset into the Attribute Table to place the descriptor. Note that this is the offset into the table created using ATGRS. As such the Service Declaration is always offset 0, and Characteristics use two attribute entries (1 for the Characteristic Declaration, 1 for the Characteristic Value). Descriptors use 1 attribute entry. For a Service with 1 Characteristic and 1 Descriptor, the offsets would be as follows:
  Attribute Offset 0 = Service Declaration
  Attribute Offset 1 = Characteristic Declaration
  Attribute Offset 2 = Characteristic Value
  Attribute Offset 3 = Characteristic Descriptor

- **GRS_Handle_Included_Service:** GRS_Handle of the Service to be included

Example(s):

1. Adding an Include Definition to Service with GRS Handle 1 at Attribute Offset 1. The included service has a GRS Handle of 0.

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

9.10.12 Add Characteristic (ATGAC)

GATT ADD CHARACTERISTIC

Function: This command is used to add a characteristic to a registered (but not published) service. A characteristic uses two attributes (one for the characteristic declaration and 1 for the characteristic) so the Attribute Offset directly after the one used as the parameter for this command will be used for the characteristic value. This function accepts the properties of the characteristic and also the security properties of the characteristic which can be used to control which connected devices can read/write to the characteristic value (if the characteristic is readable or writable).

Command Format: ATGAC,<GRS_Handle>,<Attribute Offset>,<Characteristic Properties Mask>, <Security Properties>, <UUID>

Command Parameter(s):

- **GRS_Handle:** GRS_Handle of the Service to add the Include Attribute to.

- **Attribute Offset:** Offset into the Attribute Table to place the characteristic. Note that this is the offset into the table created using ATGRS. As such the Service Declaration is always offset 0, and Characteristics use two attribute entries (1 for the Characteristic Declaration, 1 for the Characteristic Value). Descriptors use 1 attribute entry. For a Service with 1 Characteristic and 1 Descriptor, the offsets would be as follows:

  Attribute Offset 0 = Service Declaration
  Attribute Offset 1 = Characteristic Declaration
  Attribute Offset 2 = Characteristic Value
  Attribute Offset 3 = Characteristic Descriptor

  The highlighted offset would be the offset used to add the characteristic to the table.

- **Characteristic Properties Mask:** Bit mask made up of the following bits.

  0x01 = Broadcast Allowed
  0x02 = Read Allowed
  0x04 = Write Without Response Allowed
  0x08 = Write Allowed
  0x10 = Notifications Allowed
  0x20 = Indicate Allowed
  0x80 = Extended Properties Present

- **Security Properties:** Bit mask that specifies the security properties of the attribute.

  0x00 = No Security Required
  0x01 = Unauthenticated Pairing with Encryption required for write

0x02 = Authenticated Pairing with Encryption required for write
0x10 = Unauthenticated Pairing with Encryption required for read
0x20 = Authenticated Pairing with Encryption required for read

- **UUID:** Characteristic UUID
  16 bit or 128 bit UUID

Example(s):

1. Adding characteristic with UUID AABB to service with GRS Handle of 0 at Attribute Offset 1. The characteristic is Read and Writable with no security requirements.

```
COMMAND:  ATGAC,0,1,10,0,AABB<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

9.10.13 Add Descriptor (ATGAD)

GATT ADD DESCRIPTOR

Function: This command is used to a descriptor to a registered (but not published) service. A descriptor is a attribute that is used to modify/describe a characteristic value or that allows a characteristic descriptor to be configured.

Command Format: ATGAD,<GRS_Handle>,<Attribute Offset>,<Descriptors Property Mask>,
<Security Properties>, <UUID>

Command Parameter(s):

- **GRS_Handle:** GRS_Handle of the Service to add the Include Attribute to.

- **Attribute Offset:** Offset into the Attribute Table to place the descriptor. Note that this is the offset into the table created using ATGRS. As such the Service Declaration is always offset 0, and Characteristics use two attribute entries (1 for the Characteristic Declaration, 1 for the Characteristic Value). Descriptors use 1 attribute entry. For a Service with 1 Characteristic and 1 Descriptor, the offsets would be as follows:
  Attribute Offset 0 = Service Declaration
  Attribute Offset 1 = Characteristic Declaration
  Attribute Offset 2 = Characteristic Value
  Attribute Offset 3 = Characteristic Descriptor

- **Descriptors Properties Mask:** Bit mask made up of the following bits.
  0x01 = Read Allowed
  0x02 = Write Allowed

- **Security Properties:** Bit mask that specifies the security properties of the attribute.
  0x00 = No Security Required
  0x01 = Unauthenticated Pairing with Encryption required for write
  0x02 = Authenticated Pairing with Encryption required for write
  0x10 = Unauthenticated Pairing with Encryption required for read
  0x20 = Authenticated Pairing with Encryption required for read

- **UUID:** Descriptor UUID
  16 bit or 128 bit UUID

Example(s):

1. Adding a descriptor with UUID BBCC at Attribute Offset 3 that is writable and has no security properties. The GRS Handle of the service that the descriptor is being added to is 0.

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

9.10.14 Publish Service (ATGPS)

GATT PUBLISH SERVICE

Function: This command is used to publish a previously registered service. Once a service is published no Includes/Characteristics/Descriptors can be added to it. A published service will be visible to any connected device. This command will return the handle range of the published service in the GATT database.

Command Format: ATGPS,0

Command Parameter(s):

- **GRS_Handle:** GRS_Handle of the Service to publish.

Example(s):

1. Publishing the service with GRS Handle of 0.

```
COMMAND:  ATGPS,0<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
          <cr_lf>
          <Service Start Handle>,<Service End Handle><cr_lf>
```

Note:

- *Service Start Handle – Handle of the Service Declaration for registered service. This is the first GATT Handle that belongs to the registered Service.*

- *Service End Handle – Last handle that belongs to the registered service in the GATT Database.*

9.10.15 Delete Service (ATGDS)

GATT DELETE SERVICE

Function: This command is used to delete a previously registered or published service. One this command is sent to delete a service no Includes/Characteristics/Descriptors may be added to the service and the service will not be visible to any connected client.

Command Format: ATGDS,<GRS_Handle>

Command Parameter(s):

- **GRS_Handle:** GRS_Handle of the Service to un-commit.

Example(s):

1. Deleting service with GRS Handle of 0.

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

9.10.16 Read Response (ATGRR)

GATT READ RESPONSE

Function: This command is used to respond to a read request by a remote device.

Command Format: ATGRR,<Request_ID>,<Value_Format>,<Value>

Command Parameter(s):

- **Request_ID:** Request Identifier for the Read Request.

- **Value_Format:** Value Format. One of the following:
  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 to respond with.

Example(s):

1. Test example…

```
COMMAND:   ATGRR,1,5,HelloWorld<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

9.10.17 Write Response (ATGWR)

GATT WRITE RESPONSE

Function: This command is used to accept a write request from a remote device.

Command Format: ATGWR,<Request_ID>

Command Parameter(s):

- **Request_ID:** Request Identifier for the Write Request.

Example(s):

1. Test example…

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

9.10.18 Error Response (ATGER)

GATT ERROR RESPONSE

Function: This command is used to send an error response to a request from a remote device.

Command Format: ATGER,<Request_ID>,<Error_Code>

Command Parameter(s):

- **Request_ID:** Request Identifier for the Request that is being rejected.
- **Error_Code:** Value ATT Error Code to respond to the Request with.

Example(s):

1. Example:

```
COMMAND:   ATGER,0,6<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

9.10.19 Server Notification (ATGSN)

GATT SERVER NOTIFICATION

Function: This command is used to send a Server Notification of a characteristic.

Command Format: ATGSN,<GRS_Handle>,<Conn_Handle>,<Characteristic_Attribute_Offset>,
<Value_Format>,<Value>

Command Parameter(s):

- **GRS_Handle:** GRS Handle of the Service to send the notification.

- **Conn_Handle:** Connection Handle of the device to send the notification to.

- **Characteristic_Attribute_Offset:** Attribute Offset of the Characteristic Value that is to be notified.

- **Value_Format:** Value Format. One of the following:
  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 to respond with.

Example(s):

1. Sending a notification with a ASCII string data.

```
COMMAND:  ATGSN,0,0,2,5,Hello World<cr>
RESPONSE: <cr_lf>
          OK<cr_lf>
```

9.10.20 Server Indication (ATGSI)

GATT SERVER INDICATION

Function: This command is used to send a Server Indication of a characteristic.

Command Format: ATGSI,<GRS_Handle>,<Conn_Handle>,<Characteristic_Attribute_Offset>,
<Value_Format>, <Value>

Command Parameter(s):

- **GRS_Handle:** GRS Handle of the Service to send the notification.

- **Conn_Handle:** Connection Handle of the device to send the notification to.

- **Characteristic_Attribute_Offset:** Attribute Offset of the Characteristic Value that is to be notified.

- **Value_Format:** Value Format. One of the following:
  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 to respond with.

Example(s):

1. Sending an indication with a ASCII string data.

```
COMMAND:   ATGSI,0,0,2,5,Hello World<cr>
RESPONSE:  <cr_lf>
           OK<cr_lf>
```

Note:

- *A GSCONF event will be dispatched when the remote device acknowledges the Indication.*

10 Command Set Summary Table

10.1 Command Status Responses

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

10.2 Events

10.2.1 General Events

	Event	Description
	Reset	
	SS1-LE4.0-1BLE	Reset
	Done	
	DONE	Background Task Complete
	Connection	
	CONNECT	Connect
	DISCONNECT	Disconnect
	Discovery	
	DISCOVERY	Device Discovery
	Pairing	
	PAIR_REQ	Pairing Request
	PAIRED	Pairing Successful
	PAIR_FAIL	Pairing Failed
	Authentication	
	PK_REQ	Passkey Request
	PK_DIS	Passkey Display

10.2.2 Low Energy Events

	Event	Description
	Conn Param Update	
	SCCPS	Set Current Connection Parameter Status
	CPU	Connection Parameter Update
	GATT	
	GATT_DONE	GATT Done
	GATT_DPS	Discovered Primary Service
	GATT_DC	Discovered Characteristic
	GATT_DCD	Discovered Characteristic Descriptor
	GATT_VAL	Characteristic/Descriptor Value
	GRREQ	Read Request
	GWREQ	Write Request
	GWORP	Write Without Response
	GSCONF	Server Indication Confirmation
	Resolved	
	RESOLVED	Address Resolved

10.3 General Commands

	Command	Description	Factory Default	Stored?
	AT			
	AT	Attention	-	-
	Reset			
	ATRST	Reset	-	-
	ATFRST	Factory Reset	-	-
	Module Information			
	ATMT?	Get Module Type	-	-
	ATST?	Get Stack Type	-	-
	ATV?	Get Version	-	-
	ATA?	Get Address	-	-
	ATSN/ATSN?	Set/Get Name	SS1+1BLE*xxxxxx*	X
	Status			
	ATCS?	Get Connection Status	-	-
	ATRSSI?	Get RSSI Value	-	-
	Config Control			
	ATSCL/ATSCL?	Set/Get Configuration Lock	0	X
	ATFC	Flash Config (Manual)	-	-
	ATSFC/ATSFC?	Set/Get Flash Configuration Setting	1	X

	Response Config			
	ATSDIF/ATSDIF?	Set/Get Discovery Formatting	65535,1,1	X
	Hardware Config			
	ATSPL/ATSPL?	Set/Get Power Level	6	X
	ATSUART/ATSUART?	Set/Get UART Settings	7,0,0,1	X
	ATSPIO/ATSPIO?	Set/Get PIO		X
	Cancel			
	ATDC	Cancel	-	-
	Disconnect			
	ATDH	Disconnect	-	-
	Authentication			
	ATPKR	Passkey Response	-	-
	Utilities			
	ATLETXT	LE Transmitter Test	-	-
	ATLERXT	LE Receiver Test	-	-

10.4 Low Energy Commands

	Command	Description	Factory Default	Stored?
	LE Status			
	ATSLE?	Get State LE	-	-
	ATLCALE?	Get Last Connected Address LE	FFFFFFFFFFFF	X
	Advertising			
	ATDSLE	Dial As Slave LE	-	-
	ATDSDLE	Dial As Slave Direct LE	-	-
	ATSDSLE/ATSDSLE?	Set/Get ATDSLE Parameters	0,0,7	X
	ATSDSTLE/ATSDSTLE?	Set/Get ATDSLE Timing Parameters	0,160,2048	X
	ATSDSDLE/ATSDSDLE?	Set/Get ATDSLE Advertising Data	-	X
	LE Discovery			
	ATDILE	LE Discovery	-	-
	ATSDILE/ATSDILE?	Set/Get ATDILE Parameters	0,0,1,10	X
	ATSDITLE/ATSDITLE?	Set/Get ATDILE Timing Parameters	10240,16,16	X
	LE Connect			
	ATDMLE	Dial As Master LE	-	-
	ATDMLLE	Dial As Master Last LE	-	-
	ATSDMTLE/ATSDMTLE?	Set/Get ATDMLE Timing Parameters	0,16,16	X
	Connection Parameters			
	ATSDCP/ATSDCP?	Set/Get Default Connection Parameters	8,16,0,400	X
	ATSCCP/ATSCCP?	Set/Get Current Connection Parameters	-	-
	LE Pairing			

🟦	ATPLE/ATPLE?	Pair Device/Get Paired Device List LE	0,0	X
🟦	ATSPLE/ATSPLE?	Set/Get Pairing Parameters LE	2,0,3	X
🟦	ATUPLE	Un Pair Device LE	-	X
🟦	ATCPLE	Clear Pair List LE	-	X
🟦	ATSPK/ATSPK?	Set/Get Fixed Passkey	0	X
	White List			
🟦	ATSWL/ATSWL?	Set/Get White List	0,0	X
🟦	ATUWL	Un White List Device	-	X
🟦	ATCWL	Clear White List	-	X
	GATT Commands			
🟦	ATGDPS/ATGDPSU	GATT Discover Primary Services	-	-
🟦	ATGDC/ATGDCU	GATT Discover Characteristics	-	-
🟦	ATGDCD	GATT Discover Char Descriptors	-	-
🟦	ATGR/ATGRU	GATT Read	-	-
🟦	ATGW/ATGWN	GATT Write	-	-
🟦	ATGRS	GATT Register Service		
🟦	ATGAI	GATT Add Include		
🟦	ATGAC	GATT Add Characteristic		
🟦	ATGAD	GATT Add Descriptor		
🟦	ATGPS	GATT Publish Service		
🟦	ATGDS	GATT Delete Service		
🟦	ATGRR	GATT Read Response		
🟦	ATGWR	GATT Write Response		
🟦	ATGER	GATT Error Response		
🟦	ATGSN	GATT Server Notification		
🟦	ATGSI	GATT Server Indication		