WICED Studio

EZ-Serial WICED Firmware Platform User Guide for EZ-BT Modules

Associated Part Family: EZ-BT CYBT-4130XX-02

Document Number. 002-30728 Rev. **

Cypress Semiconductor

An Infineon Technologies Company

198 Champion Court

San Jose, CA 95134-1709

www.cypress.com, www.infineon.com

Contents

About This Document

This document provides a complete guide to the EZ-Serial™ platform for the CYBT-4130XX-02 EZ-BT modules.

Purpose and Audience

This document provides information on the EZ-Serial text and binary communication protocol through the PUART transport interface. The protocol is an implementation example of how a host MCU can communicate with the EZ-Serial module.

Scope

This document covers the following concepts related to EZ-Serial and provides all information required to interface to the EZ-Serial FW platform on the target Cypress modules:

- System description and functional overview (Introduction and Getting Started)
- Firmware configuration examples (Operational Examples)
- Complete design examples (Application Design Examples)
- API protocol implementations for external MCU (Host API Library)
- Troubleshooting guides (Troubleshooting)
- Reference material (API Protocol Reference to Configuration Example References)

Acronyms and Abbreviations

Acronym/Abbreviation	Expanded Form
ADC	Analog to Digital Converter
API	Application Programming Interface
BLE	Bluetooth Low Energy
BR/EDR	Basic Rate/Enhanced Data Rate
BT	Bluetooth
CYSPP	Cypress Serial Port Profile
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GPIO	General Purpose Input/Output
HCI	Host Controller Interface
JSON	JavaScript Object Notation
L2CAP	Logical Link Control and Adaptation Protocol
MITM	Man-in-the-middle
OTA	Over-the-air
OTA	Over-the-Air
PIN	Personal Identification Number
PWM	Pulse Width Modulation
SIG	Special Interest Group
SMP	Security Manager Protocol
SPP	Serial Port Profile

Acronym/Abbreviation	Expanded Form
TX	Transmit
IoT	Internet of Things
PDS	Power Down Sleep
SDS	Shut Down Sleep
BR	Basic Rate
EDR	Enhanced Data Rate

Typographic Conventions

Convention	Usage

`Courier New`	Courier New, 9pt.
	Displays file locations, user entered text, and source code examples:
	`C:\ ...cd\icc\`
	`CyDelay`

`Consolas`	Consolas 9pt. API and function names (when mentioned within body text) If there is not response to the `DOWNLOAD_MINIDRIVER` command, the device may be in autobaud mode.
Italics	Arial,9pt in body text; 8pt in table text. Displays file names and reference documentation: Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*.
File > Open	Represents menu paths: **File** > **Open** > **New Project**
Bold	Displays commands, menu paths, and icon names in procedures: Click the **File** icon and then click **Open**.
Times New Roman	Displays an equation: $2 + 2 = 4$
Text in gray boxes	Describes Cautions or unique functionality of the product.

IoT Resources and Technical Support

The wealth of data available here will help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. You can access a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates. You can acquire technical documentation and software from the Support Community website.

1 Introduction

This guide explains EZ-Serial platform for the CYBT-4130XX-02 EZ-BT modules and covers the following:

- Cypress Serial Port Profile (CYSPP) UART-to-BLE bridge functionality
- GPIO status and control connections
- GAP central and peripheral operation
- GATT server and client data transfers
- L2CAP connections
- Customizable GATT structures
- Security features such as encryption, pairing, and bonding
- Remote configuration
- Beacon behavior with iBeacon and Eddystone
- API protocol allowing full control over these behaviors from an external host
- Serial Port Profile (SPP) using Bluetooth for RS232 serial cable emulation functionality

1.1 How to Use this Guide

The high-level concepts covered in this document are organized into the following categories:

- Cypress Serial Port Profile (CYSPP) UART-to-BLE bridge functionality
- GPIO status and control connections
- GAP central and peripheral operation
- GATT server and client data transfers
- L2CAP connections (requires a device with 256K flash memory)
- Customizable GATT structures
- Security features such as encryption, pairing, and bonding
- Remote configuration
- Beacon behavior with iBeacon and Eddystone
- API protocol allowing full control over these behaviors from an external host
- Serial Port Profile (SPP) using Bluetooth for RS232 serial cable emulation functionality

The following approach provides a good way to gain familiarity with EZ-Serial quickly:

Read through Introduction and Getting Started for a functional overview.

Find at least one example from Operational Examples that is interesting or relevant to your intended design. Follow along with the described configuration on a development kit for a true hands-on experience. These examples provide excellent out-of-the-box feature demonstration:

- Getting Started in CYSPP Mode with Zero Custom Configuration
- Defining Custom Local GATT Services and Characteristics
- Detecting and Processing Written Data from a Remote Client
- Bonding with or without MITM Protection
- Configuring iBeacon Transmissions
- Updating Firmware through WICED OTA GATT

Find at least one design example from Application Design Examples that is like the type of system you intend to use CYBT-4130XX-02 EZ-BT modules with, especially noting the functional capabilities provided by the configuration and GPIO connections.

If you are combining EZ-Serial with an external host microcontroller, read through Host API Library to understand how the external MCU will need to communicate with the module.

Spend a few minutes reading through Troubleshooting to avoid unnecessary frustration later if something does not behave the way you expect.

See the reference material available in this document to allow fast access to additional information and resources. When in doubt, always consult the API reference for helpful information and related content concerning any API command, response, or event.

Throughout the guide, you will find API methods referenced in the following format:

gpio_set_drive (SIOD, ID=9/5).

These links contain three important parts:

- Proper descriptive name (for example, gpio_set_direction), unique among all other methods.
- Text-mode name (for example, SIOD), applicable when using the API protocol in text mode (see Using the API Protocol in Text Mode).
- Group/method ID values (for example, 9/5), present in the 4-byte header when using the API protocol in binary mode (see Using the API Protocol in Binary Mode).

Click any linked API method for detailed reference material in API Protocol Reference.

1.2 Block Diagram

The EZ-Serial platform is built on top of CYBT-4130XX-02 EZ-BT modules from Cypress. Depending on the specific application, this platform may utilize an external host device such as a microcontroller (MCU) connected to the module via UART, GPIO pins, or both. CYBT-4130XX-02 EZ-BT modules communicate with a remote device using Bluetooth Low Energy (BLE) and Basic Rate/Enhanced Data Rate (BR/EDR) protocol. Throughout this document, BT is used instead of BR/EDR to indicate the support of BR/EDR protocol.

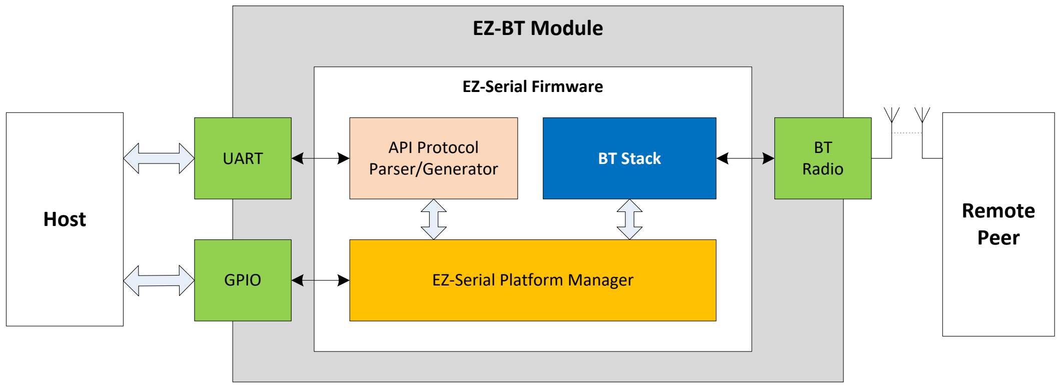


Figure 1-1. EZ-Serial System Block Diagram

1.3 Functional Overview

EZ-Serial provides an easy way to access the most commonly needed hardware and communication features in BT-based applications. To accomplish this, the firmware implements an intuitive API protocol over the UART interface and exposes several status and control signals through the module's GPIO pins.

1.3.1 BLE Communication Features

The EZ-Serial platform has the following BLE-related features:

- Bluetooth 4.2 support on compatible modules
- Master and slave connection roles
- Central, peripheral, broadcaster, and observer GAP roles
- Client and server GATT roles
- Customizable GATT database definition
- Direct L2CAP connectivity for maximum throughput
- Encryption, bonding, and protection from man-in-the-middle (MITM) threats
- CYSPP mode for bidirectional serial data transmission
- Over-the-air (OTA) method for firmware updates
- iBeacon and Eddystone beaconing
- Remote firmware configuration
- Efficient low-power operation

1.3.2 BT Communication Features

The EZ-Serial platform has the following BT-related features:

- BT 4.2 support on compatible modules
- SPP Profile

1.3.3 Hardware and Communication Features

The EZ-Serial platform also implements several features that rely on internal chipset features and local interfaces:

- Flexible text-mode and binary-mode API protocols
- GPIO reading, writing, and interrupt detection
- On-demand ADC conversion
- Configurable PWM output

1.3.4 Development Limitations

EZ-Serial is a ready-to-use platform intended to satisfy a wide variety of application design requirements with minimal effort. If you have use cases that cannot be handled easily with the EZ-Serial platform, use WICED® Studio SDK or ModusToolbox® IDE to build a custom firmware image. You can flash a custom firmware image onto any module via the debug interface and completely replace the existing EZ-Serial image at any time. To update EZ-Serial later, simply download the latest image from the Cypress website and flash it using the same mechanism.

For details on where to find these images, see Latest EZ-Serial Firmware Image.

1.4 EZ-BT WICED Module Support

Table 1-1 lists the modules that share the same hardware and firmware interfaces. The definitions of these modules' hardware pins are same, the behavior and firmware APIs are also the same as documented in this user guide.

Modules/Devices	EZ-Serial Hardware ID	Chip
CYBT-413034-02	E4	CYW20719B1
CYBT-413055-02	E5	CYW20719B2
CYBT-413061-02	E6	CYW20721B2

Table 1-1. Supported Devices

2 Getting Started

EZ-Serial allows for rapid integration of BLE and BT wireless communication into your designs. Its support for multiple API protocol formats enables easy testing of functions by typing commands into a serial terminal from your computer. Once the intended functionality is confirmed, the exact same behavior can be achieved with a compact binary protocol on a host microcontroller.

2.1 Prerequisites

For a streamlined experience, it is recommended that you have the following parts available:

- CYBT-413034-EVAL EZ-BT™ Module Arduino Evaluation Board with CYBT-413034-02 EZ-BT™ WICED Module.
- CYBT-413034-EVAL EZ-BT™ Module Arduino Evaluation Board with CYBT-413055-02 EZ-BT™ WICED Module.
- CYBT-413034-EVAL EZ-BT™ Module Arduino Evaluation Board with CYBT-413061-02 EZ-BT™ WICED Module.
- Computer with serial terminal software such as Tera Term, Realterm, or PuTTY
- *Optional:* CY5677 CySmart Bluetooth® Low Energy (BLE) 4.2 USB Dongle - BLE
- *Optional:* CYUSBS232 USB-UART LP Reference Design Kit for maximizing throughput with flow control
- *Optional:* Bluetooth 4.0 USB Dongle Adapter with CYW20702 Chipset – BT Classic/EDR
- *Optional:* BLE/BT-capable mobile device such as an iPad, iPhone, or Android phone or tablet

The CYBT-413034-EVAL EZ-BT™ Module Evaluation Board has two USB-to-UART bridge built in. The optional CySmart BLE dongle used with the matching CySmart software supports various client-side functions such as establishing connection and exploring GATT without a BLE-capable smartphone or tablet. The optional CYW20702 Bluetooth dongle supports various Bluetooth classic functions such as SPP connection without a BT-capable smartphone or tablet.

You can control EZ-Serial over a UART interface without additional GPIOs; see Application Design Examples for details. However, it is recommended that you use the CYBT-413034-EVAL evaluation board for the best experience learning and prototyping due to its comprehensive design and peripheral support.

2.2 Factory Default Behavior

The following is the default configuration of EZ-Serial firmware:

- UART interface configured for 115200 baud, 8 data bits, no parity, 1 stop bit
- UART flow control enabled
- Protocol parser/generator operating in **text mode** with local echo **enabled**
- CYSPP serial data transfer profile **enabled in auto-start mode**
- CYCommand remote configuration profile **enabled** with no special security
- All optional GPIO status/control pin functions **disabled** in both input and output directions

When the module is powered ON or reset, it will generate the system_boot (BOOT, ID=2/1) API event. This is only an example of one API method used by the platform; see API Protocol Reference for details on the structure and behavior of the API protocol.

The boot event will appear as shown below, if the protocol generator is in the default text mode:

```
@E,0040,BOOT,E=01000305,S=02041A2A,P=0100,H=E4,C=00,A=CBCBB705DD05,T=01
```

This text-mode string of data indicates:

- `@E` – An event has occurred.
- `0040` – There are 64 bytes (0x40) of content to follow.
- `BOOT` – The event which occurred is the **BOOT** event.
- `E=01000305` – The EZ-Serial application version is 1.0.3 build 5 (0x05).
- `S=02041A2A` – The WICED BTSDK stack version is 2.4.0 build 6698 (0x1A2A).
- `P=0100` – The protocol version is 1.0.
- `H=E4` – The hardware platform is CYBT-413034-02.
- `C=00` – The cause for this boot/reset is standard power-cycle or XRES hardware signal.
- `A=CBCBB705DD05` – The Bluetooth MAC address of this module is CB:CB:B7:05:DD:05.
- `T=01` – The Bluetooth MAC address type is random address.

Note: The version data and MAC address shown here are examples only. Actual values may differ.

Once the system boots, EZ-Serial will automatically start BT SPP, A2CP, AVRCP, and HFP profile service, and start the BLE CYSPP connection process by advertising as peripheral device, unless the CP_ROLE pin is asserted (LOW) in which case it will start the process by scanning as a central device. In the peripheral role, the gap_adv_state_changed (ASC, ID=4/2) API event will follow the boot event:

```
@E,000E,ASC,S=01,R=03
```

In the central role, the mgap_scan_state_changed (SSC, ID=4/3) API event will occur after the boot event, potentially followed by one or more scan result events:

```
@E,000E,SSC,S=01,R=03

@E,0062,S,R=00,A=00A050421650,T=00,S=CE,B=00,D=020106110700A1...
```

A central-mode scan will continue until it finds a compatible peer, and then EZ-Serial will automatically initiate a connection and set up the CYSPP data pipe and enter data mode upon completion. To change this behavior, you must either reconfigure the module using the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command or asset states of the CP_ROLE and CYSPP GPIO pins externally when it was powered ON or reset.

See Using CYSPP Mode and Cable Replacement Examples with CYSPP for details concerning CYSPP configuration and behavior. See GPIO Reference for the complete GPIO reference.

2.3 Connecting a Host Device

EZ-Serial communicates with an external host device such as a microcontroller using serial data (UART) and simple GPIO signals for status and control. Depending on your application, you may need to use one, both, or neither of these in your final design. Application Design Examples describe each of these use cases.

2.3.1 Connecting the Evaluation Board

When using the recommended EZ-BT WICED Evaluation Board for prototyping, simply connect the mini-USB cable between your PC and the main board and ensure that the evaluation module is securely plugged into the receptacle. This provides power to the module and a communication interface (UART) via the onboard USB-to-UART bridge. Once you have connected the cable and allowed any necessary drivers to install, two new virtual COM ports will become available, as shown in Figure 2-1. Usually the lower one (1 COM54) is HCI UART port, the higher one (2 COM86) is PUART port.

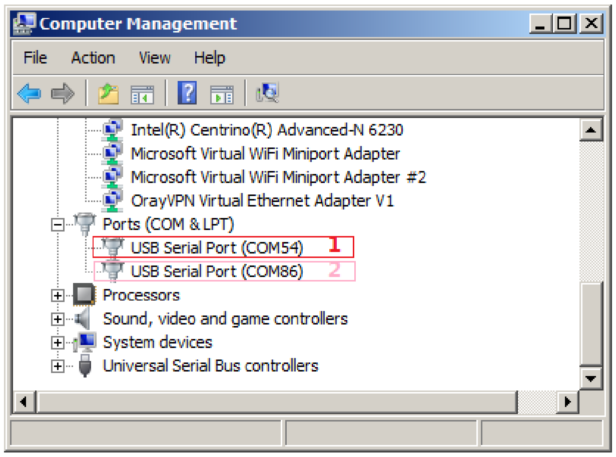

Computer Management
File
Action
View
Help
Intel(R) Centrino(R) Advanced-N 6230
Microsoft Virtual WiFi Miniport Adapter
Microsoft Virtual WiFi Miniport Adapter #2
OrayVPN Virtual Ethernet Adapter V1
Ports (COM & LPT)
USB Serial Port (COM54)
1
USB Serial Port (COM86)
2
Processors
Sound, video and game controllers
System devices
Universal Serial Bus controllers

Figure 2-1. Virtual Serial Port from Evaluation Board

Note: COM54 and COM86 are shown in Figure 2-1, but your port number may differ.

You can then use this serial port in any compatible application on your PC, such as Tera Term, Realterm, or PuTTY.

2.3.2 Connecting the Serial Interface

You can also connect your own host or USB adapter for UART communication. The module's UART interface uses standard true-type logic (TTL) signals, with logic LOW at the GND (0 V) level and logic HIGH at the VDD level (typically 3.3 V). This is necessary for high-throughput tests, which require flow control.

WARNING: Do not connect the module directly to RS-232 signals which have VDD level range between ±3 ~ ±15. To prevent damage to the device, you must add voltage convertors before connecting to RS-232 signals.

EZ-Serial's UART interface has two required signals for data and two optional signals for flow control, if enabled:

- Required: **RXD** – Receive data (input), connect to host TXD (output)
- Required: **TXD** – Transmit data (output), connect to host RXD (input)
- Optional: **RTS** – Module-side flow control (output), connect to host CTS (input), default enabled
- Optional: **CTS** – Host-side flow control (input), connect to host RTS (output), default enabled

See GPIO Pin Map for Supported Modules for pin-to-function correlations.

The default port settings are 115200 baud, 8 data bits, no parity, and one stop bit. Flow control is supported and enabled by default, and can be specifically disabled if desired.

You can change these settings using the system_set_uart_parameters (STU, ID=2/25) API command. UART transport settings are protected, which means the settings cannot be written to flash until they have first been applied to RAM. This prevents unintentional communication lockouts. See Protected Configuration Settings for details on protected settings.

If you experience any problems communicating over the serial interface, see Troubleshooting for solutions to common issues.

2.3.3 Connecting GPIO Pins

EZ-Serial also supports GPIO connections for status signals (output) and control signals (input). These allow more flexible hardware design choices and more efficient operation than what the serial interface alone provides.

The firmware provides eight single-function pins for status and control, aside from the four or two pins used for UART communication. These pin functions are enabled by default, but many of these functions can be disabled with the gpio_set_function (SIOF, ID=9/3) API command. Disabling the special functions on these pins allows you to use them for GPIO and manual interrupt detection.

Table 2-1 summarizes the functions provided by these pins. For additional information including module-specific pin assignments, operational side-effects, and default logic states, see GPIO Reference.

Pin Name	Direction	Optional[1]	Functional Description
LP_MODE	Input	No	Low-power mode control. Assert (LOW) to prevent sleep, de-assert (HIGH) to allow sleep. After the device has enter the Sleep mode, assert (LOW) will wake the device up.
ATEN_SHDN	Out	Yes	Module will assert (LOW) to indicate internal serial buffer overflow.
CP_ROLE	Input	Yes	CYSPP role control. Assert (LOW) for central mode, de-assert (HIGH) for peripheral mode. **Note**: When this pin is connected with a PULL-UP resister, this pin can only be pulled down to LOW by external. The read value of this pin may always be HIGH when not pulled down to LOW by external, so the firmware role control configuration is disabled, only can be controlled by external.
CYSPP	Input	No	CYSPP mode control. Assert (LOW) for CYSPP data mode, de-assert (HIGH) for command mode. **Note**: Asserting this pin will begin CYSPP operation in the configured role even if the CYSPP profile is disabled in the platform configuration. See Using CYSPP Mode for details.
DATA_READY	Output	Yes	Data ready indicator. Asserted (LOW) when serial data is read to be sent to the host, de-asserted (HIGH) after all data is fully transmitted.
CONNECTION	Output	Yes	Connection indicator. Asserted (LOW) when a BLE connection is established, de-asserted (HIGH) upon disconnection. **Note**: When the CYSPP data mode is active with the CYSPP pin in the asserted (LOW) state, the CONNECTION pin is asserted only when a remote device has connected and completed the CYSPP GATT data characteristic subscription, indicating that the bidirectional data pipe is ready. It is de-asserted when data can no longer flow, either due to disconnection or because the data characteristic subscription is ended.
LP_STATUS	Output	Yes	Low-power state indicator. Asserted (LOW) if the CPU is awake, de-asserted (HIGH) if asleep.
FACTORY_TR	Input	Yes	Factory test/reset control. Assert (LOW) at boot time to trigger factory test mode, indicated by the system_factory_test_entered (TFAC, ID=2/4) API event. If asserted (LOW) at boot time while the CYSPP pin is simultaneously asserted (LOW), this will trigger a factory reset of all user-defined settings on the module, returning the firmware to a known state upon the next boot. **Note**: If entered, manufacturing test mode will remain active until you de-assert the FACTORY_TR pin. Factory test mode is a special manufacturing step used by Cypress and is not intended for general use.

Table 2-1. GPIO Function Summary

By default, the pins noted as output are not strongly driven. To prevent unintentional damage, avoid the cases that directly short between VDD and GND. Since this can result in unexpected behavior with some external devices that have equal or stronger pulls in input mode, you can change the drive mode of special-function output pins to use strong drive instead with the gpio_set_function (SIOF, ID=9/3) API command. Only the **UART_TX** pin is strongly driven by default, because it cannot function properly with any other configuration.

For more details on GPIO functionality, see GPIO Reference.

[1] *Optional pin functions can be disabled to allow standard GPIO behavior.*

2.4 Communicating with a Host Device

Once you have connected a host to the module via the serial interface, you can send and receive data. EZ-Serial supports two different modes of communication: command mode (API protocol communication and control) and CYSPP mode (transparent wireless cable replacement to remote device). The following sections describe these modes.

The active communication mode depends on the state of the CYSPP pin, which can be one of the following options:

- CYSPP pin externally de-asserted (HIGH): Command mode (text or binary)
- CYSPP pin externally asserted (LOW): CYSPP mode

Ensure that the CYSPP pin is in the intended state at boot time to achieve the desired behavior. If you assert this pin, the API parser and generator become inactive, because all serial data is piped through the BLE connection (once established). You will experience lack of communication if you attempt to send API commands to the module while in CYSPP mode.

2.4.1 Using the API Protocol in Text Mode

EZ-Serial implements a text-mode API protocol which allows full control of the platform using human-readable commands, responses, and events. This mode is the default setting from the factory to provide the fastest possible path to rapid prototyping. Commands are typed using short codes, and responses and events return with predictable timing and formats.

2.4.1.1 Text Mode Protocol Characteristics

The text mode protocol has the following general behavior:

- Commands sent from the host must be terminated with a carriage return (0x0D) or line feed (0x0A) byte, or both, or with a semicolon (; or 0x3B) byte.

  Note: The semicolon byte is a visible character used to terminate a command where only the visible characters can be input in the text mode.
- Commands begin with ‘/’ (forward slash), ‘S’, ‘G’, or ‘.’ to indicate ACTION, SET, GET, or PROFILE commands, respectively.
- Commands are always immediately followed by a corresponding response, if they are parsed correctly.
- Commands with multiple arguments allow the arguments to be supplied in any order.
- Commands with multiple arguments do not require all arguments to be present in most cases; SET commands with some arguments omitted will leave non-set values unchanged, and ACTION commands with some arguments omitted will fall back to the default platform settings relevant for those arguments.

Commands with syntax errors are followed by the system_error (ERR, ID=2/2) API event with an error code indicating the nature of the problem, rather than a response packet (see Error Codes).

- All numeric data must be entered in hexadecimal notation, without prefixes (“0x”) or signs (“+” or “-”); negative numbers should be entered in two’s complement form (for example, -1 = FF, -16 = F0, -128 = 80).
- All multi-byte numeric data is entered and expressed in big-endian byte order (for example, 0x12345678 is “12345678”).
- Text command codes and hexadecimal data are not case sensitive.
- New command entry in text mode must start with a printable ASCII character (0x20 – 0x7E), or the byte will be ignored. When the module enters Sleep mode, assert the LP_MODE GPIO pin LOW to wake the module up from Sleep mode. The send command data will be ignored before the module wakes up from Sleep mode. For more details, see Managing Sleep States.
- Responses always begin with “@R,” followed by a 16-bit “length” value describing the number of bytes that come after the four length characters (including the comma), followed by the response text code.
- Responses always include a “result” value as the first parameter after the text code, indicating success or failure.
- Events always begin with “@E,” followed by a 16-bit “length” value similar to responses described above.
- Responses and events are terminated with carriage return (0x0D) and line feed (0x0A) bytes.
- Lines beginning with a “#” symbol are treated as comments and discarded by the parser.

2.4.1.2 *Text Mode API Command Categories*

There are four main categories of commands in text mode: ACTION, SET, GET, and PROFILE. All these categories use the same basic syntax, but execute different types of behavior.

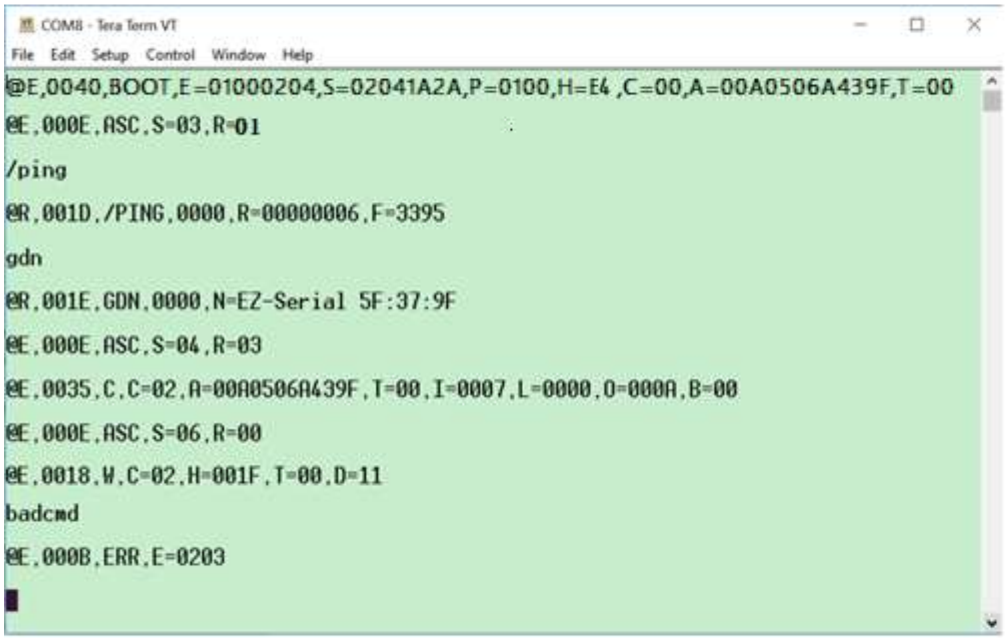
Category	Features
ACTION	ACTION commands trigger operations that cannot persist across resets or power-cycles, with very few exceptions. These commands establish connection, query GPIO logic states, enter advertisement mode, discover remote GATT, and transfer data. The following are the exceptions to the "current session only" rule: • system_store_config (/SCFG, ID=2/4)system_store_config (/SCFG, ID=2/4): Writes all modified settings to flash immediately • system_factory_reset (/RFAC, ID=2/5): Clears all modified settings and reset the module • system_write_user_data (/WUD, ID=2/11): Writes arbitrary user data to a dedicated section of flash • gatts_create_attr (/CAC, ID=5/1): Adds custom GATT database attributes • gatts_delete_attr (/CAD, ID=5/2): Removes custom GATT database attributes • smp_pair (/P, ID=7/3): Initiates pairing, resulting in new bonding data stored in flash • smp_delete_bond (/BD, ID=7/2): Deletes an existing bond, altering data stored in flash
SET	SET commands affect configuration settings that control many types of behavior, but do not typically trigger immediate changes to the operational state like ACTION commands do. Every argument in a SET command may be stored in nonvolatile (flash) memory so that it persists across power-cycles. Modified settings are stored in RAM only by default, and you must use the `/SCFG` command to write the modified settings to flash. In text mode, you can also invoke a SET command with a '`$`' after the text code (for example, "`SDN$,N=...`") to cause the change to be written to both RAM and flash immediately. A small number of SET commands also manage protected settings, which are the settings that can affect core chipset operation and communication. For these settings, you cannot write changed values directly to flash without performing a separate write to RAM only first. This prevents accidental changes that are difficult to undo. For more details on this behavior, see Protected Configuration Settings.
GET	GET commands provide the ability to read all settings that can be changed with SET commands. There is a corresponding GET command for every SET command found in the protocol with matching parameters returned in the response. Like SET commands, GET commands return data from the RAM-stored configuration structure by default. However, using the '`$`' after the text code will cause the flash-stored data to be returned instead. A few GET commands are similar in name to related ACTION commands such as `GIOL` (get GPIO logic settings) and `/QIOL` (query GPIO logic state). Keep in mind that GET/SET commands concern user-defined settings, while ACTION commands concern immediate behavior changes. Always see the API reference material when in doubt about the intended use and behavior of any API method.
PROFILE	PROFILE commands configure the behavior of special built-in behaviors, such as CYSPP data mode, CYCommand remote configuration mode, and iBeacon and Eddystone beaconing. Depending on the profile, these commands may perform actions or get or set configuration values as described for the previous three command types.

Table 2-2. Text Mode Command Categories

For more information on these command categories and behaviors, see the configuration hierarchy in Factory, Boot, Runtime, and Automatic Settings and the material in API Protocol Reference.

2.4.1.3 Text Mode API Example

The easiest way to use text command mode is with a serial terminal application. You can use any application of this kind, if it works with standard serial ports and can be configured to open the port with the proper baud rate, flow control, and other settings. Figure 2-2 shows an example session using factory default firmware and the PuTTY terminal application, starting with the system_boot (BOOT, ID=2/1) API event and demonstrating a few commands, responses, and other events.



```
@E,0040,BOOT,E=01000204,S=02041A2A,P=0100,H=E4,C=00,A=00A0506A439F,T=00
@E,000E,ASC,S=03,R=01
/ping
@R,001D,/PING,0000,R=00000006,F=3395
gdn
@R,001E,GDN,0000,N=EZ-Serial 5F:37:9F
@E,000E,ASC,S=04,R=03
@E,0035,C,C=02,A=00A0506A439F,T=00,I=0007,L=0000,O=000A,B=00
@E,000E,ASC,S=06,R=00
@E,0018,W,C=02,H=001F,T=00,D=11
badcmd
@E,000B,ERR,E=0203
```

Figure 2-2. Text Command Mode Session with Tera Term

Table 2-3 describes the various protocol methods shown in Figure 2-2.

Direction	Content	Detail
←RX	`@E,0040,BOOT,E=01000204,S=02041A2A,P=0100,H=E4,C=00,A=00A0506A439F,T=00`	system_boot (BOOT, ID=2/1) API event received: app = 1.0.2 build 4 stack = 2.4.0 build 6695 protocol = 1.0 hardware ID = E4, indicates CYBT-413034-02 module boot cause = code boot MAC address = 00:A0:50:6A:43:9F MAC address type = public address

←RX	`@E,000E,ASC,S=03,R=01`	gap_adv_state_changed (ASC, ID=4/2) API event received:
		state = 3 (Active Undirected High)
		reason = 1 (boot up)

TX→	`/ping`	system_ping (/PING, ID=2/1) API command sent to ping the local module to verify proper communication

←RX	`@R,001D,/PING,0000,R=00000006,F=3395`	system_ping (/PING, ID=2/1) API response received:
		result = 0 (success)
		runtime = 3 seconds
		fraction = 13025/32768 seconds

TX→	`gdn`	gap_get_device_name (GDN, ID=4/16) API command sent to get the configured device name

←RX	`@R,001E,GDN,0000,N=EZ-Serial 5F:37:9F`	gap_get_device_name (GDN, ID=4/16) API response received:
		result = 0 (success)
		name = “EZ-Serial 5F:37:9F”

←RX	`@E,000E,ASC,S=04,R=03`	gap_adv_state_changed (ASC, ID=4/2) API event received:
		state = 4 (Active Undirected Low)
		reason = 3 (CYSPP operation)

Direction	Content	Detail

←RX	`@E,0035,C,C=02,A=00A0506A439F, T=00,I=0007,L=0000,O=000A,B=00`	gap_connected (C, ID=4/5) API event received:
		conn_handle = 2
		peer = 00:A0:50:6A:43:9F
		addr_type = 0 (public)
		interval = 7 (8.75ms)
		slave_latency = 0
		supervision_timeout = 0xA (100 = 1 second)
		bond = 0 (not bonded)

←RX	`@E,000E,ASC,S=06,R=03`	gap_adv_state_changed (ASC, ID=4/2) API event received:
		state = 6 (Active Non-connectable Low)
		reason = 3 (CYSPP operation)

←RX	`@E,0018,W,C=02,H=001F,T=00, D=11`	gatts_data_written (W, ID=5/2) API event received:
		conn_handle = 2
		attr_handle = 0x1F (31)
		type = 0 (simple write)
		data = 1 bytes `[11]`

TX→	`badcmd`	Invalid API command sent to demonstrate text mode error event
←RX	`@E,000B,ERR,0203`	system_error (ERR, ID=2/2) API event received: reason = 0x0203 (Unrecognized Command)

Table 2-3. Text Mode Communication Example

See the reference material API Protocol Reference for details on each of these API methods and text-mode syntax rules.

2.4.2 Using the API Protocol in Binary Mode

EZ-Serial also implements a binary-format API protocol that allows the same control of the platform using compact binary commands, responses, and events. This mode is typically preferable when controlling the EZ-Serial-based module from an external microcontroller. The binary byte stream is easier to parse and generate from MCU application code than human-readable text strings.

The binary protocol uses a fixed packet structure for every transaction in either direction. This fixed structure comprises a 4-byte header followed by an optional payload, terminating with a checksum byte. The payload carries information related to the command, response, or event. If present, this payload always comes immediately after the header and before the checksum byte.

Header				Payload (optional)	Checksum
`[0]` Type	`[1]` Length	`[2]` Group	`[3]` ID	`[4...N-1]` Parameter(s)	`[N]` Summation

Table 2-4. Binary Packet Structure

The checksum byte is calculated by starting from `0x99` and adding the value of each header and payload byte, rolling over back to 0 (instead of 256) to stay within the 8-bit boundary. The checksum byte itself is not included in the summation process. For the example 4-byte binary packet for the system_ping (/PING, ID=2/1) API command:

```
C0 00 02 01
```

Calculate the checksum as follows:

```
0x99 + 0xC0 + 0x00 + 0x02 + 0x01 = 0x15C
```

Retain only the final lower 8 bits (0x5C) for the 1-byte checksum value. The final 5-byte packet (including checksum) is:

```
C0 00 02 01 5C
```

The above structure allows a packet parser implementation to know exactly how much data to expect in advance any time a new packet begins to arrive, and to calculate the checksum as new bytes arrive.

The “Type” byte in the header contains information not only about the packet type (highest two bits), but also the memory scope (where applicable), and the highest three bits of the 11-bit “Length” value. For details on the binary packet format and flow, see the API structural definition in Protocol Structure and Communication Flow.

2.4.2.1 Binary Mode Protocol Characteristics

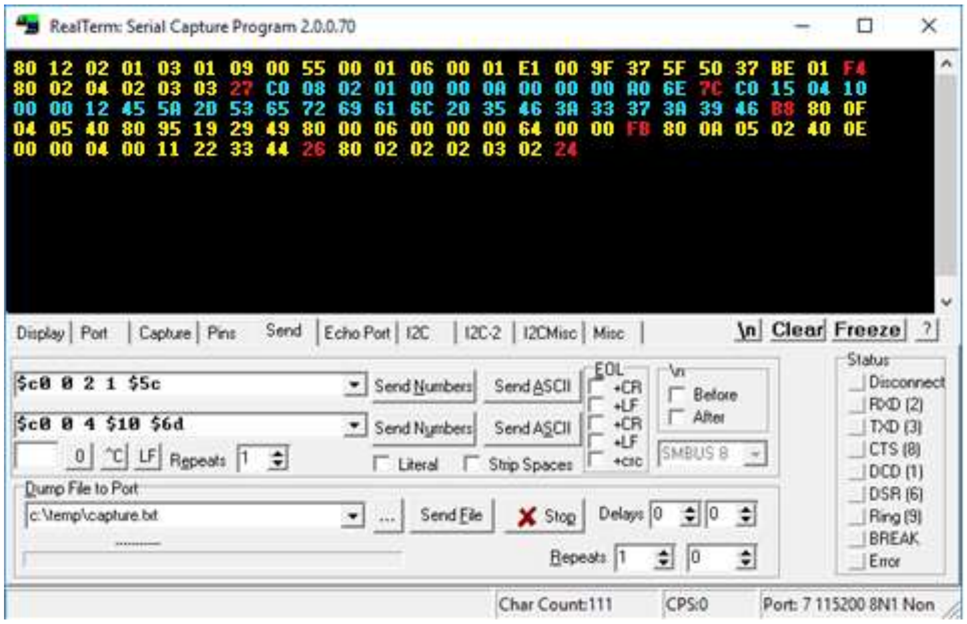
The binary mode protocol has the following general behavior:

- Commands sent from the host must begin with a properly formatted 4-byte header.
- Commands must contain the number of payload bytes specified in the **Length** field from the header.
- Commands must end with a valid checksum byte, but no additional termination such as NULL or carriage return.
- Commands are always immediately followed by a response, if they are parsed correctly.
- Commands require all arguments to be supplied in the binary payload according to the protocol structural definition, in the right order (no arguments are optional).
- Commands with syntax errors are followed by a system_error (ERR, ID=2/2) API event with an error code indicating the nature of the problem, rather than a response packet.
- Commands must be fully transmitted within one second of the first byte, or the parser will time out and return to an idle state after triggering the system_error (ERR, ID=2/2) API event with a timeout error code.
- All multi-byte integer data is entered and expressed in little-endian byte order (e.g. 0x12345678 is [78 56 34 12]). Note that this only applies to API method arguments and parameters with a fixed width - 1, 2, or 4-byte integers and 6-byte MAC addresses.
- All multi-byte data passed inside a variable-length byte array (uint8a or longuint8a) remains in the original order provided by the source. This includes UUID data found during GATT discovery. If unsure, consult the API reference manual to verify the argument data type.
- Response payloads always begin with a 16-bit "result" value as the first parameter, indicating success or failure of the command triggering the response.
- The binary command header includes a single bit in the first byte which performs the same duty as the '$' character in text mode, to cause changed settings to be written to flash immediately instead of just RAM.

2.4.2.2 Binary Mode API Example

The easiest way to use binary command mode is with a host MCU or other application that has a complete parser and generator implementation available, such as the host API library provided by Cypress and discussed Host API Library.

However, it is also possible to test individual commands manually with a serial terminal application capable of entering and displaying binary data. Figure 2-3 shows an example of testing individual commands manually using Realterm, including hexadecimal representation of data. There is no local echo when binary mode is used, so Figure 2-3 does not show the command packets sent to the module. To assist in identifying the packet types and boundaries, responses are colored **cyan,** events are **yellow**, and the final checksum byte of each packet is **red**.



```
80 12 02 01 03 01 09 00 55 00 01 06 00 01 E1 00 9F 37 5F 50 37 BE 01 F4
80 02 04 02 03 03 27 C0 08 02 01 00 00 0A 00 00 00 A0 6E 7C C0 15 04 10
00 00 12 45 5A 2D 53 65 72 69 61 6C 20 35 46 3A 33 37 3A 39 46 B8 80 0F
04 05 40 80 95 19 29 49 80 00 06 00 00 00 64 00 00 F8 80 0A 05 02 40 0E
00 00 04 00 11 22 33 44 26 80 02 02 02 03 02 24
```

$c0 0 2 1 $5c

$c0 0 4 $10 $6d

Repeats 1

c:\temp\capture.txt

Delays 0 0

Repeats 1 0

Char Count:111 CPS:0 Port: 7 115200 8N1 Non

Figure 2-3. Binary Command Mode Session with Realterm

Note: This is helpful for testing, but not an efficient way to communicate in binary mode.

Each binary packet (including the checksum byte) is described in Table 2-5. For better comparison between text mode and binary mode, the API transactions demonstrated here are the same as those used in the text mode example. Note that multi-byte integer data such as the 6-byte MAC address and the 16-bit advertisement interval are transmitted in little-endian byte order.

Direction	Content	Detail
←RX	`80 12 02 01 03 01 09 00 55 00 01 06 00 01 E1 00 9F 37 5F 50 37 BE 01 F4`	system_boot (BOOT, ID=2/1) API event received: **app** = 0.9.1 build 3 **stack** = 6.1.0 build 85 (85 = 0x0055) **protocol** = 1.0 **hardware** = E1 - CYBT-423028-02 module **boot cause = code boot** **MAC address =** BE:37:50:5F:37:9F MAC address type = Random address
←RX	`80 02 04 02 03 03 27`	gap_adv_state_changed (ASC, ID=4/2) API event received: **state** = 3 (Active Undirected High) **reason** = 3 (CYSPP operation)
TX→	`C0 00 02 01 5C` *(not visible)*	system_ping (/PING, ID=2/1) API command sent to ping the local module to verify proper communication
←RX	`C0 08 02 01 00 00 0A 00 00 00 A0 6E 7C`	system_ping (/PING, ID=2/1) API response received: **result** = 0 (success) **runtime** = 10 seconds **fraction** = 28320/32768
TX→	`C0 00 04 10 6D` *(not visible)*	gap_get_device_name (GDN, ID=4/16) API command sent to get the configured device name
←RX	`C0 15 04 10 00 00 12 45 5A 2D 53 65 72 69 61 6C 20 35 46 3A 33 37 3A 39 46 B8`	gap_get_device_name (GDN, ID=4/16) API response received: **result** = 0 (success) **name** = “EZ-Serial 5F:37:9F”

Direction	Content	Detail
←RX	`80 0F 04 05 40 80 95 19 29 49 80 00 06 00 00 00 64 00 00 FB`	gap_connected (C, ID=4/5) API event received: **handle** = 40 **peer** = 80:49:29:19:95:80 **addr_type** = 0 (public) **interval** = 6 (7.5 ms) **slave_latency** = 0 **supervision_timeout** = 0x64 (100 = 1 second) **bond** = 0 (not bonded)
←RX	`80 0A 05 02 40 0E 00 00 04 00 11 22 33 44 26`	gatts_data_written (W, ID=5/2) API event received: **conn_handle** = 4 **attr_handle** = 0x1F (31) **type** = 0 (simple write) **data** = 4 bytes `[11 22 33 44]`
TX→	`C0 00 EE EE 35` *(not visible)*	Invalid API command (group and ID bytes set to **0xEE**) sent to demonstrate binary mode error event
←RX	`80 02 02 02 03 02 24`	system_error (ERR, ID=2/2) API event received: **reason** = 0x0203 (Unrecognized Command)

Table 2-5. Binary Mode Communication Example

See the reference material in API Protocol Reference for details on each of these API methods and the binary packet format, including information on all header fields and supported data types.

2.4.3 Key Similarities and Differences Between Text and Binary Command Mode

The text-mode and binary-mode protocol formats provided by EZ-Serial have their own advantages. As a general guideline, text mode is better for initial development or one-time configuration, while binary mode is a better choice for production-stage control from an external host device due to the significantly less complex parser/generator implementation on an external host. The following are the key factors to consider when choosing the command mode:

Similarities:

- Both modes access the same internal API functionality. They are not different protocols, only different formats.
- Both follow the same command/response/event flow.
- EZ-Serial supports both simultaneously. There is no need to switch between firmware images.
- Your choice of protocol format only affects local communication with an external host over the wired serial interface. It does not have any impact on data sent over a wireless BLE connection, or on the type of host communication used on a remote device (for example, another Cypress module running EZ-Serial firmware).

Differences:

- Binary multi-byte integer data is transmitted in little-endian byte order for more efficient direct memory structure mapping on most common platforms, while text mode uses big-endian for easier left-to-right readability.
- Binary commands have a one-second timeout, while text mode commands have no timeout.
- Binary commands are semantically organized by functional group (system, protocol, GAP, GATT server, and so on) rather than the four categories used in text mode (ACTION, SET, GET, and PROFILE).
- Binary commands require all arguments in every case, while text mode commands often have optional arguments which fall back to default/preset values if omitted.
- Binary packets include basic checksum validation, while text mode packets do not.
- Binary is more efficient for MCU-based communication, while text mode is easier for manual entry in a terminal.
- Binary commands are never echoed back to the host, while text mode commands are echoed (by default).

2.4.4 API Protocol Format Auto-Detection

EZ-Serial uses text mode for API protocol communication by default, but you can change this setting with the protocol_set_parse_mode (SPPM, ID=1/1) API command. If "binary" mode is specified and written to flash, the module will use binary mode automatically on subsequent resets or power-cycles.

The parser also automatically detects whether the external host is using binary or text mode, and temporarily switches to the detected mode for the active session. The detection logic behaves in the following way:

- If the parser is in text mode, a byte received at any time with the two most significant bits set (0xC0-0xFF) will switch the parser to binary mode immediately. The “trigger” byte will not be discarded, but will be processed as the first byte in the command packet. This mechanism is considered safe because no valid text-mode command begins with a byte that has the highest two bits set.
- If the parser is in binary mode, a byte received when the parser is idle (not mid-command) that is one of the initial category characters for any of the four types of commands (‘/’, ‘S’, ‘G’, and ‘.’) will switch the parser to text mode immediately. The “trigger” byte will not be discarded, but will be processed as the first byte in the text command string. This mechanism is considered safe because no binary command begins with one of these characters. Note that this requires the parser to be idle, not in the middle of a packet, because a binary command packet could easily have one of these characters in its header or payload.

The automatically detected parse mode is not retained across power-cycles, nor is it stored in the same configuration setting area as a value explicitly set by the protocol_set_parse_mode (SPPM, ID=1/1) API command. For more details on this type of temporary configuration, see Factory, Boot, Runtime, and Automatic Settings.

2.4.5 Using CYSPP Mode

EZ-Serial implements a special CYSPP profile that provides a simple method to send and receive serial data over a BLE connection. This operational mode is separate from the normal command mode where the API protocol may be used. When CYSPP data mode is active, any data received from an external host will be transmitted to the remote peer, and any data received from the remote peer will be sent out through the hardware serial interface to the external host.

Note, when using CYSPP Mode, please make sure that the auto response is enabled. See gatts_set_parameters (SGSP, ID=5/14) for the details on the auto response flag bit.

2.4.5.1 Starting CYSPP Operation

You can start CYSPP mode using any of these three methods:

1. Assert (LOW) the CYSPP pin externally, ensuring that you have also set the CP_ROLE pin to the correct logic state for the desired GAP role. You may connect this pin to ground in hardware designs which only require CYSPP operation and never need API communication. You can also use this pin to enter CYSPP mode even if the CYSPP profile is disabled in the platform configuration.
2. Use the p_cyspp_start (.CYSPPSTART, ID=10/2) API command. You can use this command to enter CYSPP mode even if the CYSPP profile is disabled in the platform configuration.
3. Have a remote GATT client connect and subscribe to the CYSPP acknowledged data characteristic (enabling indications) or unacknowledged data characteristic (enabling notifications). This method will only enter CYSPP mode if the CYSPP profile is enabled in the platform configuration.

When starting CYSPP mode locally using either the CYSPP pin or the p_cyspp_start (.CYSPPSTART, ID=10/2) API command, the data pipe will not be immediately available because the remote device must still connect and set up the proper GATT data subscriptions. If 100% data delivery is required in this context, the host should monitor the CONNECTION pin to determine when it is safe to begin sending data from the host for BLE transmission. Once the CONNECTION pin is asserted while the CYSPP pin is also asserted, the host may send and receive data over CYSPP.

Note: Externally asserting (LOW) the CYSPP pin will always begin CYSPP operation, even if the profile has been disabled in the platform configuration via the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command. If you do not require CYSPP operation, make sure that this pin remains electrically floating or externally de-asserted (HIGH).

2.4.5.2 Sending and Receiving Data in CYSPP Data Mode

Once you have started CYSPP mode, the EZ-Serial platform will take care of the rest of the connection process and data pipe construction on the module side. If you are using modules running EZ-Serial firmware on both ends of the connection, then start CYSPP mode with complementary roles (peripheral on one end, central on the other), and the modules will automatically connect and prepare the data pipe using the processes described below.

A device such as a BLE-enabled smartphone will frequently be used for one end of the connection, and you must configure the device to follow the same procedure.

For configuration examples in each mode, see Cable Replacement Examples with CYSPP.

.

If you have configured CYSPP to operate in peripheral mode:

1. EZ-Serial will begin advertising with configured advertisement settings.
2. Upon connection, a remote peer must subscribe to one of the two “Data” characteristics:
    a. Acknowledged Data, enable indications (guaranteed reliability)
    b. Unacknowledged Data, enable notifications (faster potential throughput)
3. Remote peer may optionally subscribe to the “RX Flow Control” characteristic, to allow the server to communicate whether it is safe to write new data.
4. EZ-Serial will assert the CONNECTION pin (if enabled), indicating that CYSPP is ready to send and receive data.
5. Data pipe will remain open until the central device disconnects or unsubscribes from the data characteristic, or the CYSPP pin is de-asserted locally.

If you have configured CYSPP to operate in central mode:

1. EZ-Serial will begin scanning with configured scan settings, searching for a connectable remote peer that includes the CYSPP service UUID and matching connection key within its advertisement packet payload.
2. Upon identifying a suitable peer, it will initiate a connection to that peer with configured connection settings.
3. Upon connection, it will perform a remote GATT discovery to identify the relevant CYSPP service, characteristic, and descriptor attribute handles, if you have not manually set them already with the p_cyspp_set_client_handles (.CYSPPSH, ID=10/5) API command.
4. Upon successful completion of GATT discovery, it will subscribe to the configured data characteristic and the RX Flow Control characteristic (if enabled). Use the client flags setting of the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command to control acknowledged versus unacknowledged data and RX flow usage.
5. EZ-Serial will assert the CONNECTION pin (if enabled), indicating that CYSPP is ready to send and receive data.
6. The data pipe will remain open until the peripheral device disconnects, or the CYSPP pin is de-asserted locally.

2.4.5.3 *Exiting CYSPP Mode*

Once in CYSPP mode, the API parser is logically disconnected from incoming serial data, so you will not be able to send any commands to the module. However, you can still exit from CYSPP in two ways:

1. De-assert (HIGH) the CYSPP pin externally.
2. Have the remote GATT client unsubscribe from the relevant CYSPP data characteristic (only applies when CYSPP pin is not externally asserted).

When CYSPP operation ends, EZ-Serial will return to command mode.

> **WARNING:** It is not possible to use an API command to exit from CYSPP data mode, because the API parser is not available while in this mode. If your design needs to switch between modes on demand, include external access to the CYSPP pin so you can control the operational mode.

2.4.5.4 *Customizing CYSPP Behavior for Specific Needs*

While the default behavior is suitable in many cases, there are configuration settings that allow a great deal of control over this behavior. The following list describes the options that can be changed, and how to change the options:

- CYSPP mode uses the system’s configured UART host transport settings for sending and receiving serial data. To change these settings, use the system_set_uart_parameters (STU, ID=2/25) API command.
- CYSPP mode uses the system’s configured radio transmit power setting for all BLE communication. To change this setting, use the system_set_tx_power (STXP, ID=2/21) API command.
- CYSPP mode supports special incoming data packetization modes. This helps make radio transmissions and data delivery more efficient in a variety of use cases. To change these settings, use the p_cyspp_set_packetization (.CYSPPSK, ID=10/7) API command.

- When operating in peripheral mode, CYSPP uses the system's configured advertisement parameters, including the advertisement and scan response packet content (which may be based on the device name) and the system's whitelist. To change these settings, use one or more of the following API commands:
  - gap_set_adv_parameters (SAP, ID=4/23)
  - gap_set_adv_data (SAD, ID=4/19)
  - gap_set_sr_data (SSRD, ID=4/21)
  - gap_set_device_name (SDN, ID=4/15)
- When operating in central mode, CYSPP uses the system's configured scanning and connection parameters, including the system's whitelist. To change these settings, use one or more of the following API commands:
  - gap_set_scan_parameters (SSP, ID=4/25)
  - gap_set_conn_parameters (SCP, ID=4/27)

2.4.5.5 Understanding CYSPP Connection Keys

EZ-Serial also supports CYSPP connection keys, which improve usability in environments where multiple CYSPP-capable devices are operating in an automated configuration. This feature allows an advertising peripheral device to broadcast an arbitrary 4-byte value that a scanning device can filter against, searching either for a masked range of devices or a single specific device.

CYSPP connection keys are not set in the factory default configuration; CYSPP peripheral advertisements contain a "0" key, and CYSPP central scans do not attempt to match any bits. To change this, use the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command, and specifically the "local_key", "remote_key", and "remote_mask" arguments of this command as described in the following sections.

2.4.5.6 Using CYSPP Peripheral Connection Key

The CYSPP peripheral connection key affects only the content of the advertisement packet while the module is in an advertising state. The CYSPP peripheral role does not include any filtering behavior; filtering is left to the scanning device that is operating in the CYSPP central role.

When the CYSPP profile is enabled, the platform-managed advertising packet contains a special Manufacturer Data field to hold the local connection key value. It is not stored elsewhere, such as in a GATT characteristic. This advertisement packet field has the structure listed in Table 2-6.

Length	Type	Company ID	Connection Key
`07`	`FF`	*`b0 b1`*	*`b0 b1 b2 b3`*

Table 2-6. CYSPP Peripheral Connection Key Manufacturer Data Field Structure

The Company ID value is a 16-bit value that the Bluetooth SIG assigns to member companies that have requested them (see resources on www.bluetooth.com). The factory default value is the Cypress company identifier, 0x0131, but you can change this with the same command used to change other CYSPP parameters. Note that both the Company ID and the Connection Key values are broadcast in little-endian byte order.

Use the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command and enter the desired 32-bit value for the "local_key" argument to apply a new peripheral connection key. Changes will take effect immediately, even if the module is already advertising in the CYSPP peripheral role.

WARNING: EZ-Serial will only incorporate the CYSPP peripheral connection key into the advertising packet if you have not enabled user-defined advertisement content. If you have configured user-defined advertisement content instead as described in Customizing Advertisement and Scan Response Data, then changing this value will have no effect. You must ensure that your user-defined advertisement packet contains an equivalent field to allow scanning devices to filter properly.

Example 1: Update CYSPP peripheral key to 0x11223344

Direction	Content	Effect
TX→	`.CYSPPSP,L=11223344`	Apply new CYSPP configuration
←RX	`@R,000E,.CYSPPSP,0000`	Response indicates success

2.4.5.7 Using the CYSPP Central Connection Key and Mask

The CYSPP central connection key affects the scanning operation that occurs when CYSPP is active in the central role and has not yet connected to a remote peer. The central connection key has two parts:

1. remote_key – The value used to compare with the peripheral key from the advertisement packet.
2. remote_mask – The bitmask used to strip away any irrelevant bits from the peripheral key before comparison.

For EZ-Serial to initiate a connection to a CYSPP peripheral device, the “remote_key” value must match the advertised peripheral connection key after a logical AND operation with the “remote_mask” value. A mask with all bits set (“`FFFFFFFF`”) will require an exact match between the two keys, while a mask with no bits set (“`00000000`”) will match any device. The factory default configuration is the all-zero mask, so any CYSPP-capable peer will match. The mask values between these two extremes provide the option to connect only to devices within specific segments of the connection key space, much like an IP-based network. Table 2-7 provides examples of each case.

Remote Key	Remote Mask	Key and Mask	Result
`11223344`	**`FFFFFFFF`**	**`11223344`**	Connect to a device whose key is exactly “`11223344`”
`55667788`	**`FFFFFF`**`00`	**`556677`**`00`	Connect to any device whose key begins with “`556677`”
`12345789`	**`FFFF`**`0000`	**`1234`**`0000`	Connect to any device whose key begins with “`1234`”
`18F7A9CC`	**`FFFF`**`00`**`FF`**	**`18F7`**`00`**`CC`**	Connect to any device whose key begins with “`18F7`” and ends with “`CC`”
Any	`00000000`	`00000000`	Connect to any device

Table 2-7. Connection Key and Mask Examples

Use the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command and enter the desired 32-bit values for the “remote_key” and “remote_mask” arguments to apply a new central connection key and mask. Changes to these values will take effect immediately, even if the module is already scanning in the CYSPP central role.

Note: If an advertising peripheral device is broadcasting the CYSPP service UUID but does not have a **Manufacturer Data** field containing a connection key in the same advertisement packet, the value “0” will be substituted for an actual key to filter the scanning device.

Example 1: Update CYSPP central key to 0x11223344 and require exact matching

Direction	Content	Effect
TX→	`.CYSPPSP,R=11223344,M=FFFFFFFF`	Apply new CYSPP configuration
←RX	`@R,000E,.CYSPPSP,0000`	Response indicates success

2.4.5.8 CYSPP Configuration and Pin States

Table 2-8 describes the relationship between the state of the CYSPP pin and the CYSPP firmware configuration managed with the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command. Note the following two key behaviors concerning hardware control versus software control:

- Asserting the CYSPP pin externally will always trigger automatic CYSPP operation in the configured role (or the role dictated by externally driving the CP_ROLE pin). This will occur even if you have disabled the profile in software.
- CYSPP data mode (where the API is suppressed and all serial data is channeled to the remote peer) ultimately depends on the state of the CYSPP pin. EZ-Serial pulls this pin to the appropriate logic level based on internal CYSPP state changes when CYSPP is enabled, but you can override the pulled state with an external host or hardware design feature.

CYSPP Pin State	CYSPP “enable” Value in Configuration	CYSPP Operation
Floating **(assumed default)**	Disabled	**Inactive.** All advertising, scanning, connections, GATT subscriptions, GATT transfers, and so on occur via API commands and events. CYSPP GATT structure is not visible to a remote client.
	Enabled	**Idle until start.** When started via the p_cyspp_start (.CYSPPSTART, ID=10/2) API command, module will begin advertising or scanning depending on configured role and **CP_ROLE** pin. API events (boot, stage changes, connections, and so on) will be **visible** over UART until the CYSPP data connection is opened between the local device and remote peer. The **CYSPP** pin will be pulled LOW when this occurs, at which point the API will be suppressed and the serial interface may be used only for CYSPP data pipe. This mode will continue until the remote host disconnects or unsubscribes.
	Autostart **(factory default)**	**Automatic.** Same behavior as “Enabled” case above, except CYSPP operation begins automatically at boot time and restarts upon disconnection.
Externally driven HIGH (de-asserted)	Disabled	**Inactive.** All advertising, scanning, connections, GATT subscriptions, GATT transfers, etc. occur via API commands and events. CYSPP GATT structure is not visible to a remote client.
	Enabled	**Idle until start, command mode retained.** When started via the p_cyspp_start (.CYSPPSTART, ID=10/2) API command, module will begin advertising or scanning depending on configured role and **CP_ROLE** pin. API events (BOOT, stage changes, connections, etc.) will be **visible** over UART. API communication will continue throughout the process; CYSPP data from the remote host will never be raw/transparent unless the host asserts the **CYSPP** pin.
	Autostart	**Automatic.** Same behavior as “Enabled” case above, except CYSPP operation begins automatically at boot time and restarts upon disconnection. API events will continue to be visible while **CYSPP** pin is de-asserted (HIGH).
Externally driven LOW (asserted)	Doesn’t matter	**Active regardless of firmware configuration.** Automatic advertising or scanning will begin at boot time depending on the configured role and **CP_ROLE** pin state. API events (boot, state changes, connections, and so on) will not be visible over UART, because API communication is always suppressed when **CYSPP** pin is asserted.

Table 2-8. CYSPP Configuration and Pin Relationship

2.4.5.9 *CYSPP State Machine*

Figure 2-4 describes the way EZ-Serial manages CYSPP operation, depending on firmware configuration and the logic states of the CYSPP and CP_ROLE pins.

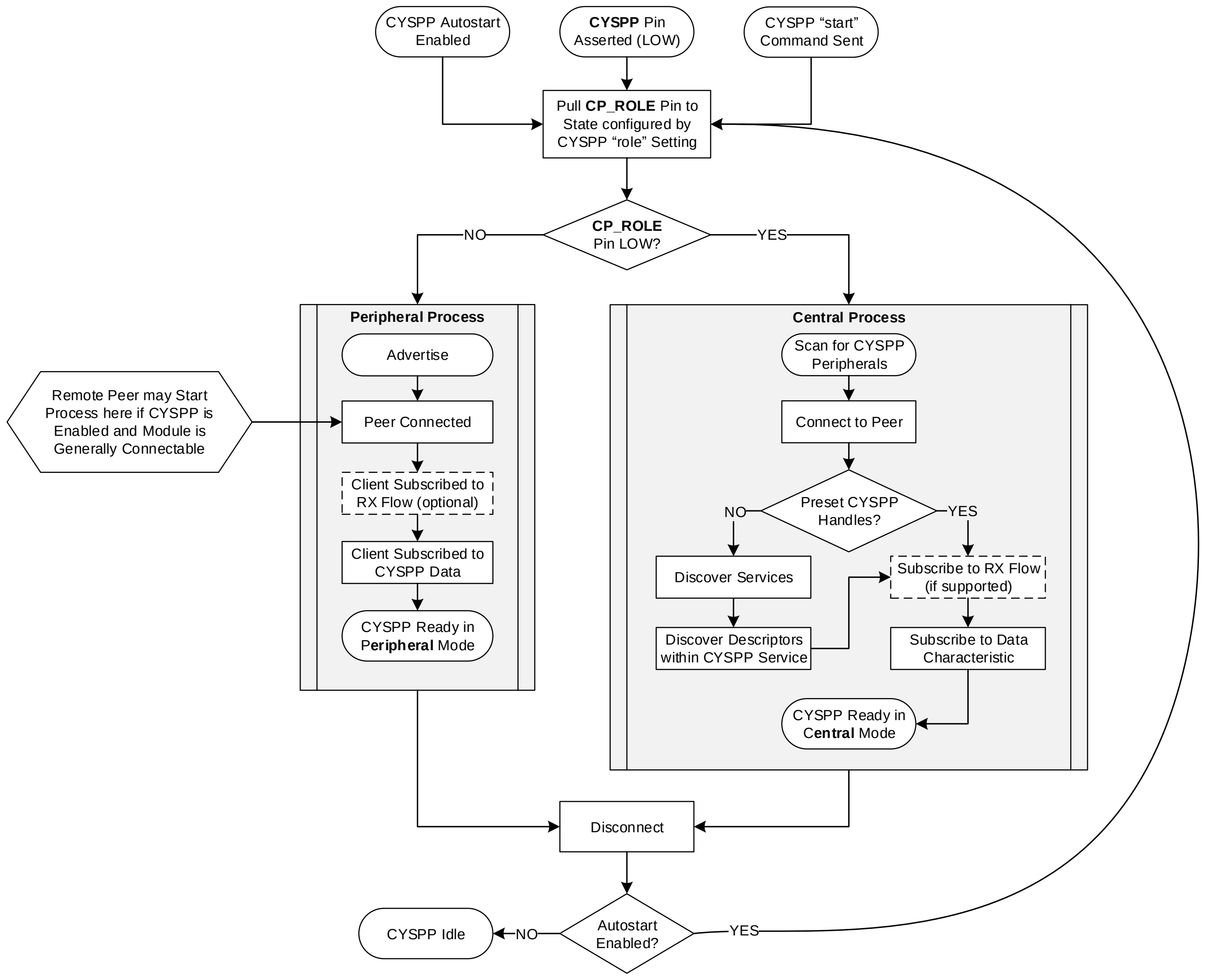

CYSPP Autostart Enabled
CYSPP Pin Asserted (LOW)
CYSPP "start" Command Sent
Pull CP_ROLE Pin to State configured by CYSPP "role" Setting
CP_ROLE Pin LOW?
NO
YES
Peripheral Process
Advertise
Remote Peer may Start Process here if CYSPP is Enabled and Module is Generally Connectable
Peer Connected
Client Subscribed to RX Flow (optional)
Client Subscribed to CYSPP Data
CYSPP Ready in Peripheral Mode
Central Process
Scan for CYSPP Peripherals
Connect to Peer
Preset CYSPP Handles?
NO
YES
Discover Services
Subscribe to RX Flow (if supported)
Discover Descriptors within CYSPP Service
Subscribe to Data Characteristic
CYSPP Ready in Central Mode
Disconnect
CYSPP Idle
NO
Autostart Enabled?
YES

Figure 2-4. CYSPP State Machine

Note that EZ-Serial pulls the CP_ROLE pin to the state configured by the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command, but if the host or hardware design drives it to a different state, CYSPP will operate in the pin-defined state and not the firmware-defined state.

2.4.6 Using BT SPP Profile

EZ-Serial supports BT SPP profile for providing a simple method to send and receive serial data over a BR/EDR connection.

2.4.6.1 *Adding Device to Windows*

On Windows, before establishing the connection to the BT SPP profile, the target device must be added to the Windows. For example, in Windows 10, go to **Settings → Devices → Bluetooth and Other Devices**. Click **Add Bluetooth and Other Devices** to scan all discoverable devices nearby. Then, in the scanned devices list, select and click the **Audio** icon item to add the BR/EDR device and wait for the added device to be completely configured. Note that during the scanning, both the BLE devices and BR/EDR devices can be scanned and displayed. The target EZ-Serial device that support BT SPP profile is always shown with an Audio icon (The compute icon item is scanned as BLE device).

After the BR/EDR device has been added and configured successfully, two Standard Serial over Bluetooth link devices can be found in the Device Manager under the COM Port device list.,

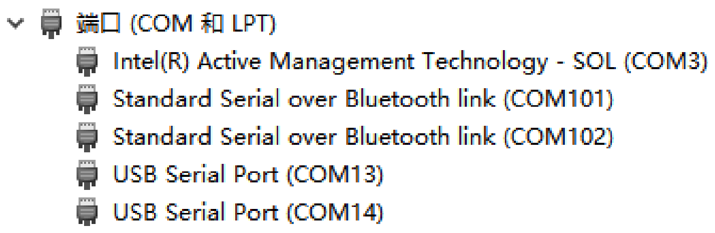

端口 (COM 和 LPT)
Intel(R) Active Management Technology - SOL (COM3)
Standard Serial over Bluetooth link (COM101)
Standard Serial over Bluetooth link (COM102)
USB Serial Port (COM13)
USB Serial Port (COM14)

Figure 2-5. Standard Serial over Bluetooth Link Devices in Device Manager

2.4.6.2 Connecting to BT SPP Profile

Open a COM terminal tool, such as Tera Term, Realterm, and so on. Select a Standard Serial over Bluetooth link port (for example COM101), set up the port, and connect to the port. If the connection is successful, the BTPCON event is displayed in the target EZ-Serial PUART port, and the data can be sent and received through the BT SPP profile. Table 2-9 lists the example events that may be received after the BT SPP Profile connection is successful.

Direction	Content	Effect
←RX	`@E,001F,BTCON,C=01,A=0F99092FE24E,B=00`	BR/EDR connection established.
←RX	`@E,002C,BTPCON,C=01,A=0F99092FE24E,T=00,B=00,H=0002`	BT SPP profile connection established.

Table 2-9. EZ-Serial Events When BT SPP Profile Connection is Established

Note that only one of the two Standard Serial over Bluetooth link devices can be used to connect to the BT SPP Profile. So, if the serial port selected first (for example, COM101) is incorrect, use the second serial port (for example, COM102).

2.4.6.3 Two EZ-Serial Devices Connected through BT SPP Profile

The BT SPP profile connection also can be established between two EZ-Serial modules. After two EZ-Serial modules boot, get the Bluetooth of one device. Then, in the other device, run the bt_connect (/BTC, ID=14/4) command with the parameter P set to 0 to connect to the target EZ-Serial device. After the two EZ-Serial modules establish connection over the BT SPP profile successfully, you can see bt_profile_connected (BTPCON, ID=14/8) events as shown in Connecting to BT SPP Profile.

Currently, if one EZ-Serial device is used to connect to another EZ-Serial device with BTT SPP profile using static random address, the connection might not be successful at the first instance. The connection will automatically disconnect and the pairing will fail. You need to rerun the same connection command, and the BT SPP profile connection will be successful.

Direction	Content	Effect
TX→	`GBA`	Get local device address
←RX	`@R,001D,GBA,0000,A=CEA2AD1A8F12,T=01`	Response with local device address
←RX	`@E,001F,BTCON,C=01,A=0F99092FE24E,B=00`	BR/EDR connection established
←RX	`@E,000F,PR,C=01,R=0905`	Pair failed with 0905
←RX	`@E,0012,BTDIS,C=01,R=1600`	BR/EDR disconnected
←RX	`@E,001F,BTCON,C=01,A=0F99092FE24E,B=00`	BR/EDR connection established
←RX	`@E,002C,BTPCON,C=01,A=0F99092FE24E,T=00,B=00,H=0002`	BT SPP profile connection established
←RX	`Data21xxxx`	Received data from device 2
TX→	`Data12xxxx`	Send data to device 1

Table 2-10. EZ-Serial Device 1 (BT SPP Server)

Direction	Content	Effect
TX→	`GBA`	Get local device address
←RX	`@R,001D,GBA,0000,A=CF99092FE24E,T=01`	Response with local device address
TX→	`/BTC,A=CEA2AD1A8F12,P=0`	Connect to device 1 with BT SPP profile
←RX	`@R,000A,/BTC,0000`	Connect started response
←RX	`@E,001F,BTCON,C=01,A=CEA2AD1A8F12,B=00`	BR/EDR connection established
←RX	`@E,000F,PR,C=01,R=0905`	Pair failed with 0905
←RX	`@E,0020,BTCF,C=00,A=CEA2AD1A8F12,R=EEEE`	BR/EDR connection failed event
←RX	`@E,001F,BTPDIS,C=00,T=00,H=0003,R=0000`	BT SPP profile disconnected
←RX	`@E,0012,BTDIS,C=01,R=1600`	BR/EDR disconnected.
TX→	`/BTC,A=CEA2AD1A8F12,P=0`	Retry to connect to 1 immediately after failed
←RX	`@R,000A,/BTC,0000`	Connect started response
←RX	`@E,001F,BTCON,C=01,A=CEA2AD1A8F12,B=00`	BR/EDR connection established
←RX	`@E,002C,BTPCON,C=01,A=CEA2AD1A8F12,T=00,B=00,H=0001`	BT SPP profile connection established
TX→	`Data21xxxx`	Send data to device 1
←RX	`Data12xxxx`	Received data from device 1

Table 2-11. EZ-Serial Device 2 (BT SPP Client)

2.4.6.4 Entering Command Mode

When the BT SPP profile connection is established, by default, the EZ-Serial device will enter the BT SPP data mode. The data input through PUART transport will be transferred to the BT SPP profile immediately. In some situations, EZ-Serial device might still require running some local commands, and this mode is called the BT SPP command mode. To support this required, the CYSPP pin is reused.

By default, the EZ-Serial device will assert the CYSPP to LOW and enter the BT SPP data mode automatically, after the BT SPP profile connection is established. To enter the BT SPP command mode, the host must pull the CYSPP pin to HIGH. When the EZ-Serial device detects that the CYSPP pin is HIGH, it will process input data from PUART as command data instead of sending the data to the BT SPP profile, so that local command mode can be supported.

When the CYSPP pin is asserted to HIGH, the host can send normal commands through PUART transport same as the BT SPP profile is not connected. When the BTT SPP command mode is not required, the host can de-assert the CYSPP pin, and the CYSPP pin will internal reset to LOW again automatically. Then, the EZ-Serial device will re-enter the BT SPP data mode automatically.

2.4.6.5 Disconnecting from BT SPP Profile

When the BT SPP profile connection is established successfully, the CYSPP pin will be asserted to LOW, and data can be sent and received through the SPP profile terminal. To stop the data input through the PUART terminal to be sent directly through the BT SPP profile, you should assert the CYSPP pin to HIGH. When the CYSPP pin is asserted to HIGH, the BT SPP connection will enter command mode instead of data mode. In this command mode, the data input through the PUART terminal will be processed as command. Then, you can use the bt_disconnect (/BTDIS, ID=14/6) command to disconnect the BT SPP profile connection. Also, when the EZ-Serial is connected to a Windows SPP terminal, close the SPP terminal, on Windows, to disconnect. Table 2-12 lists the commands and events to disconnect a EZ-Serial BT SPP profile connection.

Direction	Content	Effect
TX→	`/BTDIS,C=1`	Disconnect from the BT connection 1.
←RX	`@R,000C,/BTDIS,0000`	Response with local device address
←RX	`@E,001F,BTPDIS,C=01,T=00,H=0002,R=0000`	Connect started response
←RX	`@E,0012,BTDIS,C=01,R=1600`	BR/EDR disconnected.

Table 2-12. EZ-Serial BT SPP Profile Connection Disconnect Command and Events

2.5 Configuration Settings, Storage, and Protection

The EZ-Serial platform provides methods to customize its many built-in functions. It is important to understand how these settings are stored and changed in different contexts to avoid unexpected behavior.

2.5.1 Factory, Boot, Runtime, and Automatic Settings

EZ-Serial implements four different "layers" of configuration data, each of which serves a unique purpose. Table 2-13 describes each type of configuration storage in detail.

Layer	Details
Factory (FLASH)	**Description:** Factory-level settings are hard-coded into the firmware image and stored in flash, and cannot be changed independently by the user. They are used for runtime-level settings until/unless customized boot-level values exist. Using the system_factory_reset (/RFAC, ID=2/5) API command will revert to these values. **Content:** These values contain only platform configuration settings, but no custom GATT structure definitions or value data. **Data retention during chipset reset: YES** These values are retained upon power cycles and chipset reset conditions. **Data retention during DFU: VERSION-SPECIFIC** These values may change during the DFU process if updating to a new EZ-Serial image with different factory default values.
Boot (FLASH)	**Description:** Boot-level settings are set by the user and stored in flash, and applied to the runtime-level area for active use when the module boots. (If no customized boot-level settings have been set by the user, the factory-level settings are applied instead upon first boot.) These values can be modified using API commands, and they are erased when performing a factory reset. **Content:** These values contain both platform configuration settings and any custom GATT structure definitions. Actual GATT characteristic values such as those written by a remote client are not included in this data. **Data retention during chipset reset: YES** These values are retained during power cycles and chipset reset conditions. **Data retention during DFU: YES** These values are retained during the DFU process. Boot-level configuration data is kept in a special "user data" area of flash, which is excluded during updates to new EZ-Serial firmware images.
Runtime (RAM)	**Description:** Runtime-level settings are used as the active configuration set that controls EZ-Serial's behavior always, with a few exceptions as noted in the "Automatic" section below. API commands that set or get configuration values access this layer of configuration data unless explicitly noted otherwise. **Content:** These values contain platform configuration settings, custom GATT structure definitions, and GATT characteristic values written from a remote client. **Data retention during chipset reset: NO** These values are not retained during power cycles and chipset reset conditions. Any runtime settings or GATT database structure definitions should be written to flash with the relevant API command(s) before performing a reset. **Data retention during DFU: NO** These values are not retained during the DFU process, which involves a chipset reset prior to image transfer.

Layer	Details
Automatic (RAM)	**Description:** Automatic settings are set by the firmware based on detected external behavior, and EZ-Serial uses these values to augment the settings in the runtime configuration block. Currently, only one setting falls into this category: • API parse mode (binary or text mode depending on initial packet byte) **Content:** These values contain a very limited subset of auto-detected configuration settings, and do not include most configuration data or any GATT structure or value data. **Data retention during chipset reset: NO** These values are not retained during power cycles and chipset reset conditions. **Data retention during DFU: NO** These values are not retained during the DFU process, which involves a chipset reset prior to image transfer.

Table 2-13. Configuration Setting Storage Layers

2.5.2 Saving Runtime Settings in Flash

Storing settings in flash memory is critical to allow predictable, long-term customized behavior without needing to reconfigure each time. EZ-Serial provides two ways to accomplish this:

1. Use the system_store_config (/SCFG, ID=2/4) API command to write all current runtime-level settings to the boot-level configuration. This applies a snapshot of the current configuration to flash in one step. Use this method if you are unsure of the settings that have changed between boot-level and runtime-level values, or if you want to test out a new set of options before making them permanent.
2. Set the “flash” memory scope bit in the binary command packet header when writing new configuration values with relevant commands, or append the ‘$’ character to command names in text mode. Use this method if you know exactly which settings need to be changed, because it does not require the final use of the system_store_config (/SCFG, ID=2/4) API command afterward.

Note that while the flash memory scope bit (in binary mode) or ‘$’ character (in text mode) may be used with any command; doing so is only relevant for commands which either read or write configuration values directly. For other commands, these flags will be silently ignored. See the API reference material in API Protocol Reference for details.

To ensure the longest flash memory life, writes to flash should be as infrequent as possible in production-ready designs. Settings that must be changed frequently should be modified in RAM and only written to flash if required. Note that the internal chipsets used in the CYBT-4130XX-02 EZ-BT modules that run EZ-Serial have a minimum flash endurance rating of 100,000 cycles.

2.5.3 Protected Configuration Settings

A small number of configuration values have the potential to put the module into a state where it is no longer possible to communicate over the serial interface as intended. While it is always possible to completely revert to factory default values using the FACTORY_TR and CYSPP pins while booting the module, logical access to these pins for this purpose is not always readily available, and a complete factory reset may be too disruptive for your application.

To help avoid this potential problem, a few settings are classified as protected. This means that they must be changed at the runtime-level only (RAM) before they may be applied to the boot-level (flash) area. Currently, only one command affects protected settings: system_set_uart_parameters (STU, ID=2/25).

The changes that are most likely to cause an unintended communication lockout are serial transport reconfigurations, such as selecting a baud rate that is not supported by the host. To store new values in flash for protected configuration settings, you must either send the same command twice with the flash memory scope bit/character used only the second time, or else use the system_store_config (/SCFG, ID=2/4) API command to write all runtime-level settings to the boot level after first setting the new value in RAM only. This forces the flash write to occur using the new configuration, which can only occur if communication is still possible.

2.6 Where to Find Related Material

This guide refers to firmware images and example source code files that must be accessed separately from this document.

2.6.1 Latest EZ-Serial Firmware Image

You can find the latest available EZ-Serial firmware image files on the website.

These images are suitable for both HCI UART re-flashing through WICED Studio SDK tools and for over the air (OTA) upgrade over BLE through GATT WICED OTA service. See Device Firmware Update Examples for details on how to flash these firmware images onto target modules.

2.6.2 Latest Host API Protocol Library

You can find the latest host API protocol library source code on the website.

2.6.3 Comprehensive API Reference

While this guide contains many specific functional examples, these are not intended to provide a full reference to all possible functionality provided by the API. See API Protocol Reference for detailed material concerning the API structure and protocol.

3 Operational Examples

EZ-Serial provides a great platform on which you can build a wide variety of BLE/BT applications. The sections describe the common operations that you can experiment with or combine to create the behavior needed for your application.

3.1 System Setup Examples

These examples demonstrate basic platform behavior and configuration of the system.

Note: The first example shown below provides low-level detail and explanation of some API protocol formatting features, while all other examples assume a basic understanding of the mechanics of the protocol and will only show example snippets in text format. For details on the API methods used in each case and the binary equivalents of each command, response, and event, see API Protocol Reference.

3.1.1 Identifying Firmware and BLE Stack Version

You can obtain the EZ-Serial firmware, BLE/BT stack, and protocol version details can be obtained from the API event generated at boot time, or on demand using an API command.

3.1.1.1 Getting Version Details from Boot Event

Capture and process the system_boot (BOOT, ID=2/1) API event that occurs when the module is powered on or reset. This event includes the application version, stack version, protocol version, boot cause, and unique Bluetooth MAC address.

If the protocol parser/generator is in text mode (factory default), the system_boot (BOOT, ID=2/1) API event looks like:

```
@E,0040,BOOT,E=01000305,S=02041A2A,P=0100,H=E2,C=00,A=BE37505F379F,T=01
```

If the protocol parser is in binary mode, this event will be similar to that shown in Table 3-2, expressed in hexadecimal notation

Header	Payload	Checksum
`80 12 02 01`	`05 03 00 01 2A 1A 04 02 00 01 E2 00 9F 37 5F 50 37 BE 01`	`DF`

Table 3-1. Binary Event Data of EZ-Serial BOOT Event

To simplify manual interpretation in this guide, individual parameters within the payload are separately underlined.

Note: In text mode, multi-byte integer data is expressed in big-endian notation, while in binary mode, multi-byte integer data is transmitted in little-endian order.

The payload data in the event text/binary examples shown above is described in Table 3-2.

Text Code	Text Data	Binary Data	Details	Interpretation
E	“01000305”	`05 03 00 01`	EZ-Serial application version	Version 1.0.3 build 5
S	“02041A2A”	`2A 1A 04 02`	BLE stack version	WICED BTSDK version 2.4.0.6698
P	“0100”	`00 01`	API protocol version	Version 1.0
H	“E2”	`E2`	Hardware ID	CYBT-423054-02 module
C	“00”	`00`	Cause for boot event	Code boot
A	“BE37505F379F”	`9F 37 5F 50 37 BE`	MAC address	BE:37:50:5F:37:9F
T	“01”	`01`	MAC address type	Random/Private address type

Table 3-2. Payload Detail for Boot Event

3.1.1.2 Getting Version Details on Demand

Use the system_query_firmware_version (/QFV, ID=2/6) API command to request version details at any time. The response to this command contains the same initial information in the system_boot (BOOT, ID=2/1) API event, but it does not include the boot cause or the module's Bluetooth MAC address.

The text-mode response to this API command is as shown below:

```
@R,002C,/QFV,0000,E=01000305,S=02041A2A,P=0100,H=E2
```

Table 3-3 shows the binary-mode response packet.

Header	Payload	Checksum
C0 0D 02 06	00 00 05 03 00 01 2A 1A 04 02 00 01 E2	A4

Table 3-3. Binary Response Data of EZ-Serial Query Firmware Version Command

To simplify manual interpretation in this guide, individual parameters within the payload are separately underlined.

3.1.2 Changing Serial Communication Parameters

Use the system_set_uart_parameters (STU, ID=2/25) API command to reconfigure the serial interface used for host communication. This command affects protected settings, and therefore it must be applied in RAM first before it can be written to flash.

All data entered via text mode must be expressed in hexadecimal notation. Table 3-4 lists the common baud rates and their hexadecimal equivalents.

Baud Rate	Hex Equivalent
1,200	4B0
2,400	960
4,800	12C0
9,600	2580
14,400	3840
19,200	4B00
28,800	7080
38,400	9600
57,600	E100
115,200 (default)	1C200
230,400	38400
460,800	70800
921,600	E1000

Table 3-4. Common UART Baud Rates and Hex Equivalents

Note: EZ-Serial supports non-standard baud rates not listed in Table 3-4, and should remain below 3% clock error due to the use of an internal fractional clock divider. While this is within the tolerance level required by many UART interfaces, you should measure the actual bit timing with a scope or logic analyzer to verify that the baud rate is operating within required tolerance for your host device.

WARNING: Selecting a baud rate below 9600 and using API protocol communication can result in a situation where EZ-Serial generates API response and event packets faster than the UART interface can transmit them to the host. If this occurs, data will flow continuously out of the module, but it will not respond to incoming commands. The most likely trigger for this situation is a scan started with gap_start_scan (/S, ID=4/10), or auto-starting CYSPP client mode operation (which also begins a scan). Performing a scan in a busy environment will generate scan result events rapidly and continuously.

Possible workarounds include:

- If using CYSPP, keep the CYSPP pin externally asserted to suppress API output.

- If possible, select a faster baud rate.

- If possible, reduce the quantity of devices in the environment to decrease scan result frequency.

Example 1: Set UART to 38400 baud, even parity, flow control enabled, and store in flash

Direction	Content	Effect
TX→	`STU,B=9600,F=1,P=2`	Set new UART parameters (RAM only) – "38400" decimal is "9600" hex
←RX	`@R,0009,STU,0000`	Response indicates success
Change host UART parameters to match new settings here before sending additional data.		
TX→	`STU$`	Write UART settings to flash
←RX	`@R,000A,STU$,0000`	Response indicates success

Note, the use of the command "`STU$`" with no additional arguments. In text mode, most SET commands have no required arguments, allowing you to change only the desired settings. Optional arguments that are omitted will not be modified, because the EZ-Serial platform substitutes the current runtime values as if you had supplied all of them.

In the example above, the "baud," "flow," and "parity" settings are stored in RAM with the first command, and then the second command writes to flash whichever runtime values are affected by the system_set_uart_parameters (STU, ID=2/25) API command.

Example 2: Set UART to 115200 baud, no parity, flow control disabled, and store in RAM only

Direction	Content	Effect
TX→	`STU,B=1C200,F=0,P=0`	Apply new UART parameters
←RX	`@R,0009,STU,0000`	Response indicates success

Note: The UART flow control is enabled by default in this EZ-Serial firmware platform. When the UART flow control is enabled or disabled, the host UART terminal (such as Tera Term, Realterm, and so on) must also set the UART port flow control configuration same as the module UART flow control setting. Otherwise, the TX and RX data stream may be blocked by the incorrect RTS and CTS signals.

3.1.3 Changing Device Name and Appearance

Use the gap_set_device_name (SDN, ID=4/15) API command to set a new friendly device name at any time, and the gap_set_device_appearance (SDA, ID=4/17) API command to set a new appearance value.

EZ-Serial uses the device name and appearance to populate the GAP service's name and appearance characteristic values in the GATT database. If EZ-Serial is allowed to automatically manage the advertisement and scan response data content (default behavior), then it will also include up to 29 bytes of the device name in the scan response packet. (The limit of 29 bytes is due to a BLE specification limit on the maximum scan response payload, which is 31 bytes; the other two bytes are needed for the field length and field type values that are part of the device name field.)

Note: EZ-Serial limits the device name length to 64 bytes to minimize internal SRAM requirements.

Using EZ-Serial's special macro codes, described in Table 7-6. EZ-Serial GATT Validation Error Codes

Macro Definitions, you can enter a single text string which is expanded internally to include module-specific values—in this case, the Bluetooth MAC address. This is shown in Example 1.

The device appearance value is a 16-bit field made up of a 10-bit and 6-bit subfield. Allowed values are defined by the Bluetooth SIG and is available at developer.bluetooth.org.

Changes made to the device name and appearance values take effect immediately. They are written to the local GATT characteristics for these two values (always present), and the device name is updated in the scan response packet if user-defined advertisement content has not been enabled with the gap_set_adv_parameters (SAP, ID=4/23) API command.

Example 1: Set device name with partial MAC address incorporation

Direction	Content	Effect
TX→	`SDN$,N=EZ-Serial %M4:%M5:%M6`	Set new device name in flash using 4th, 5th, and 6th MAC bytes (module-specific)
←RX	`@R,000A,SDN$,0000`	Response indicates success

This configured name will result in an actual name of "EZ-Serial E3:83:5F" assuming the module in use has a MAC address of `00:A0:50:E3:83:5F` (as is used in other examples throughout this document).

Example 2: Set device appearance to "Generic Computer" (0x0080)

Direction	Content	Effect
TX→	`SDA$,A=0080`	Set new appearance value in flash
←RX	`@R,000A,SDA$,0000`	Response indicates success

3.1.4 Changing Output Power

Use the system_set_tx_power (STXP, ID=2/21) API command to set a new radio transmit power level. The argument to this command is not the dBm value directly, but rather a set of predefined values representing a fixed range from -28 dBm to +4 dBm. Table 3-5 lists the allowed values.

Argument	Power Level
0	-28 dBm
1	-24 dBm
2	-20 dBm
3	-16 dBm
4	-12 dBm
5	-8 dBm
6	-4 dBm
7 (default)	+0 dBm
8	+4 dBm

Table 3-5. Supported TX Power Output Options

Changes to the configured output power will take effect immediately.

Example 1: Set output power to -6 dBm

Direction	Content	Effect
TX→	`STXP,P=3`	Set new TX power (RAM only)
←RX	`@R,000A,STXP,0000`	Response indicates success

3.1.5 Managing Sleep States

EZ-Serial manages transitions between active CPU and sleep states automatically. It chooses the mode requiring the lowest safe power consumption according to the current operational state and configuration, including transitioning into Sleep mode between BLE radio events (advertising, scanning, or while connected). Table 3-6a provides a high-level summary of the four power states used by the platform.

Power Mode	Current Range (typical), V_{DD} = 3.3 V to 5.0 V	Wakeup Time	Description
Active	1.3 mA to 14 mA	NA	CPU and all peripherals are active. All functionalities are possible with no delay.
Power Down Sleep (PDS)	80 uA to 150 uA	0	CPU is in WFI and the HCLK is not running, most of the peripherals such as UART and SPI are turned OFF. The entire memory is retained, and on waking up, the execution resumes from where it was paused. Wake-up is possible by de-asserting the **LP_MODE** pin or BT/BLE connection events or read/write operations. Note that this mode can be entered only when PWM and CYSPP are not running, and the sleep level is set to 1 (PDS mode), also the timeout parameters are not set to 0.
Shut Down Sleep (SDS)	70 uA to 100 uA	0	Everything is turned OFF except LHL, RTC, and LPO. The device can come out of this mode either due to Bluetooth activity or a LHL interrupt. Wake-up is possible by de-asserting the **LP_MODE** pin or BT/BLE connection events or read/write operations. Note that this mode can be entered only when PWM and CYSSP are not running, and the sleep level is set to 2 (SDS mode), also the timeout parameters are not set to 0.

Table 3-6. EZ-Serial Power States

The EZ-Serial firmware platform for CYBT-4130XX-02 EZ-BT module does not support timed Deep Sleep and Hibernate sleep modes. When the PDS/SDS Sleep mode is enabled, the WICED stack will control and change the Sleep mode to active, pause, and PDS/SDS automatically based on the preconfigured wake up GPIO and BT/BLE connections activities and operations.

EZ-Serial uses the allowed sleep level and idle to sleep timeout value on combined data from the system-wide sleep setting, CYSPP data mode sleep setting (if CYSPP data mode is active), PWM output state, and LP_MODE pin state to control the sleep state. Figure 3-1 describes the sleep level determination logic.

After EZ-Serial enters the Sleep mode, the UART communication will also be disabled, so to reenable the UART communication, assert the LP_MODE pin to low and release it to trigger a wake-up interrupt to EZ-Serial; so EZ-Serial can wake up and switch back to Active mode to support UART communication again. Note, by default, if there is no operation with the EZ-Serial or communication through UART, the EZ-Serial will enter Sleep mode automatically based on the default configuration to save power consumption.

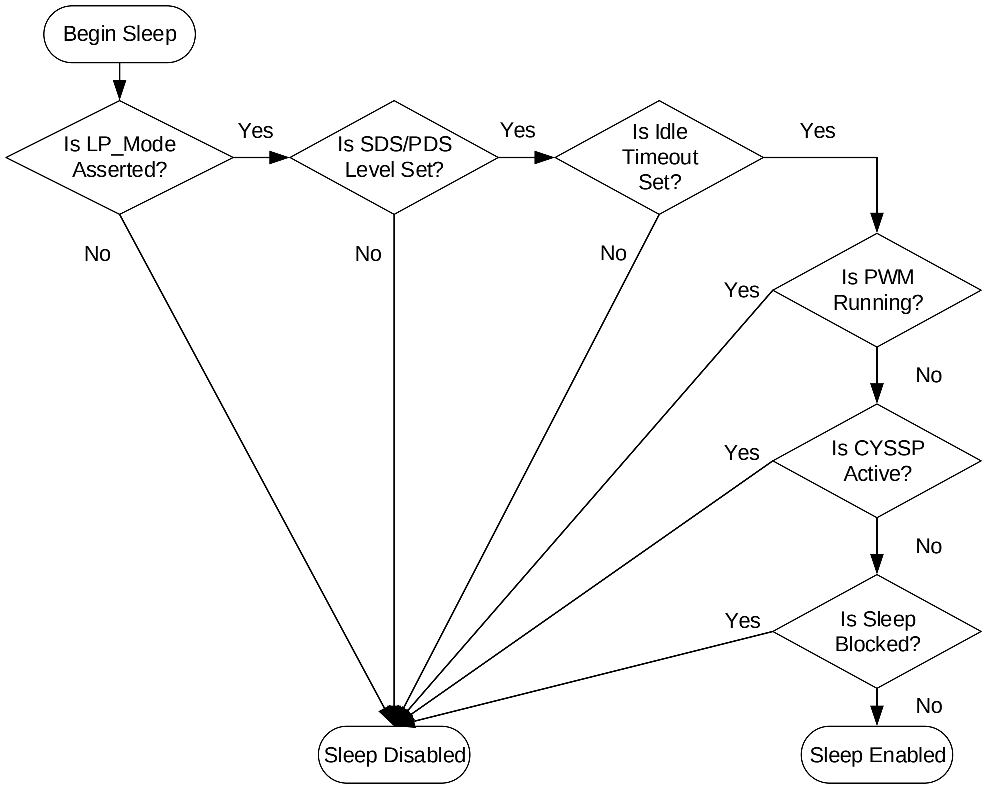


Figure 3-1. EZ-Serial Sleep State Behavior

In outline form, the sleep state logic follows this process:

1. If the LP_MODE pin is asserted, EZ-Serial can enter Sleep mode. Otherwise, it remains in Active mode.
2. If sleep level is set to PDS (1) or SDS (2) is not set, EZ-Serial can enter PDS or SDS Sleep mode. Otherwise, it remains in Active mode. The system sleep level can be configured with the system_set_sleep_parameters (SSLP, ID=2/19) API command.
3. If the idle timeout value is not set to 0, EZ-Serial can enter PDS or SDS Sleep mode after the idle time times out without any operation. Otherwise, it remains in in Active mode. Set idle timeout value to 0 to keep EZ-Serial running in active mode forever. The system sleep idle timeout value can be configured with the system_set_sleep_parameters (SSLP, ID=2/19) API command.
4. When no PWM channel is configured and enabled, EZ-Serial can enter Sleep mode. Otherwise, it remains in Active mode. The PWM output can be enabled with the gpio_set_pwm_mode (SPWM, ID=9/11) API command.
5. When CYSSP mode is not Active, EZ-Serial can enter Sleep mode. Otherwise, it remains in Active mode. The CYSPP-specific sleep level configured with the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command, if the CYSPP data pipe is open (connected and in CYSPP data mode).
6. If the Sleep mode is not blocked by other components internally, EZ-Serial can enter Sleep mode. Otherwise, it remains in Active mode. The internal sleep block events include interrupt processing, waiting for a scheduled task to process, and so on.
7. After EZ-Serial enters Sleep mode, the LP_MODE pin can be asserted LOW to wake the EZ-Serial back to Active mode.

3.1.5.1 Configuring the System-Wide Sleep Level

Configure the system-wide sleep level using the system_set_sleep_parameters (SSLP, ID=2/19) API command. When sleep is not prevented by asserting the LP_MODE pin, this value is the first “default” sleep level limit applied when calculating the Sleep mode to be used.

EZ-Serial allows only PDS (value = 1) and SDS (value = 2) sleep levels. The PDS Sleep mode is configured as the factory default system-wide sleep level, and the factory default system-wide idle sleep timeout value is set to 60 seconds (4 * 15s = 60s).

Example 1: Change system-wide Sleep level to Deep Sleep

Direction	Content	Effect
TX→	`SSLP,L=1,T=4`	Set new system Sleep level to "PDS", and set the idle timeout value to (4 * 15) 60 seconds.
←RX	`@R,000A,SSLP,0000`	Response indicates success
Transmissions to the module now require a preceding dummy byte for wake-on-RX, or proper use of the LP_MODE pin as described in Preventing Sleep with the LP_MODE Pin.		

3.1.5.2 Configuring the CYSPP Data Mode Sleep Level

Configure the CYSPP data mode Sleep level using the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command.

Setting the CYSPP data mode sleep level to enabled (value = 1) or disabled (value = 0) ensures that EZ-Serial does not use a sleep level beyond that setting whenever a CYSPP data pipe is open (connected and in CYSPP data mode). The factory default setting for this option is to allow sleep (value = 1), but factory defaults also set the system-wide sleep level limit to control the Sleep mode and enter idle time.

For using CYSPP mode in the peripheral role with legacy systems, which cannot use either the LP_MODE pin or preceding dummy bytes, one possible compromise for improved power consumption is to set the system-wide sleep level to SDS and the CYSPP data mode sleep level to sleep enabled.

Example 1: Limit CYSPP-specific sleep level to normal sleep

Direction	Content	Effect
TX→	`.CYSPPSP,S=1`	Enable system Sleep when CYSPP is Active.
←RX	`@R,000E,.CYSPPSP,0000`	Response indicates success

3.1.5.3 Preventing Sleep with the LP_MODE Pin

Assert the LP_MODE control pin LOW to prevent the module from sleeping. Properly asserting and de-asserting this pin surrounding host-to-module UART transmissions provides the most efficient power consumption while still allowing Deep Sleep at all other times.

After system enters PDS or SDS Sleep mode, the UART/SPI transmission is also disabled. To resume the use of UART/SPI transmission, assert (LOW) the LP_MODE pin to resume the system, then restart the idle count down time. The UART/SPI transmission can then be used again. This operation is required for the sending API command through the UART interface to the EZ-Serial firmware when system enters Sleep mode.

Note: The LP_STATUS output pin provides an externally accessible signal to determine whether the CPU is currently awake (LOW) or asleep (HIGH).

3.1.5.4 Managing Host and Module Sleep Simultaneously

In applications that include both an external host MCU and a BLE/BT module, usually both components need to sleep to save as much power as possible. The DATA_READY pin is asserted (LOW) whenever there is UART data in the output buffer and not yet fully clocked out of the module. Using this pin as the wake-up signal for the MCU is the recommended way to allow the module to alert the host whenever some interaction needs to occur.

Sometimes, the external MCU takes long enough to wake up that it loses the first few bits or bytes of the incoming UART data from the module. If the host needs extra time to wake up and RTS/CTS flow control is unavailable, then you should enable UART flow control in EZ-Serial with the system_set_uart_parameters (STU, ID=2/25) API command and then control the module's CTS pin from a host GPIO. When CTS is held in the de-asserted (HIGH) state, the module will wait to send any outgoing UART data. The host can complete its wake-up process and then assert (LOW) the module's CTS pin to allow serial data transmission when ready.

Real flow control support on the host MCU is not necessary in this case, and you can leave the module's RTS pin disconnected. However, you will still need to enable flow control within EZ-Serial. Flow control is not enabled by default.

To summarize the complete cycle:

1. Host sets the module CTS pin HIGH to prevent UART transmission.
2. Host enables the DATA_READY pin falling-edge interrupt.
3. Host puts the CPU to sleep.
4. Module asserts (LOW) its DATA_READY pin when relevant activity occurs.
5. Host CPU wakes up.
6. Host sets the module CTS pin LOW to allow UART transmission.
7. Module transmits data to the host for processing.

3.1.6 Performing a Factory Reset

You can perform a factory reset using either GPIO signals or an API command.

EZ-Serial will generate the system_factory_reset_complete (RFAC, ID=2/3) API event immediately after erasing all settings, and before performing the final module reset to boot to the factory default state. The platform generates this event using the previously configured parser and transport mode. While this event is typically not processed by an external host during a hardware-triggered factory reset, it helps to verify the intended flow when controlling the module via software.

After the reset completes, the system_boot (BOOT, ID=2/1) API event will occur with the "cause" parameter indicating a factory reset.

3.1.6.1 Factory Reset via Hardware GPIO Signal

To trigger a factory reset with hardware, perform the following steps:

1. Assert (LOW) the FACTORY_TR pin.
2. Assert (LOW) the CYSPP pin.
3. Power-cycle or reset the module.
4. De-assert (HIGH) the FACTORY_TR and CYSPP pins.

Note: The last step is necessary because the firmware will not perform the final chipset reset to apply new settings until at least one of the two triggering pins changes to a different state. This requirement prevents an endless loop of factory resets.

3.1.6.2 Factory Reset via API Command

To trigger a factory reset over the serial interface, use the system_factory_reset (/RFAC, ID=2/5) API command.

Example 1: Perform a factory reset

Direction	Content	Effect
TX→	`/RFAC`	Trigger factory reset
←RX	`@R,000B,/RFAC,0000`	Response indicates success
←RX	`@E,0005,RFAC`	Event indicates factory reset completed
Short delay while chipset reset and boot process occurs, ~150 ms		
←RX	`@E,0040,BOOT,E=01000305,S=02041A2A,P=0100, H=E2,C=00,A=CBCBB705DD05,T=01`	Event indicates system has rebooted, cause is set to 0x00

3.2 Cable Replacement Examples with CYSPP

EZ-Serial's CYSPP implementation provides a simple way to use a BLE connection to manage a bidirectional stream of serial data. Both ends of the connection must support CYSPP, including the ability to either provide or make use of the CYSPP GATT structure for data flow. The EZ-Serial firmware can operate as either a GAP peripheral and CYSPP server device (typical when communicating with a smartphone) or as a GAP central and CYSPP client device (typical when communicating with a second module running EZ-Serial firmware).

See Using CYSPP Mode for a description of how CYSPP mode behaves generally and how it affects API communication.

3.2.1 Getting Started in CYSPP Mode with Zero Custom Configuration

The factory default configuration enables the CYSPP profile in auto-start mode. With this configuration, the module begins advertising or scanning as soon as it has power, depending on the state of the CP_ROLE pin.

If you are using the CYBT-4130XX-02 EZ-BT modules for evaluation, perform the following steps:

1. After connecting the CYBT-4130XX-02 EZ-BT module evaluation board to the Windows system, find the correct COM port number in the Device Manager, then open the COM port with correct port settings. If you have not changed any settings previously using API commands, the defaults are 115200 baud, 8 data bits, no parity, 1 stop bit, and hardware flow control.
2. To use CYSPP in central/client mode, assert CP_ROLE to low, then press and release SW1 (Reset) button to reboot in central mode. You can release CP_ROLE pin after the module boots; CYSPP will continue to operate as a central/client device until it has established and subsequently close a connection.
3. Connect to the EZ-Serial module from a compatible remote peer as described in Using CYSPP Mode, or activate another CYSPP-capable peripheral if running the local test module in central mode as described in the previous step.
4. Wait for the p_cyspp_status (.CYSPP, ID=10/1) API event that indicates the data channel is ready. The final status event should appear as one of the following:

   `@E,000C,.CYSPP,S=21` (running in peripheral role)

   `@E,000C,.CYSPP,S=35` (running in central role)
5. Send and receive data as desired.

If you are using a custom design:

1. Connect the CP_ROLE pin to either logic LOW (central) or logic HIGH (peripheral) to define the role used. EZ-Serial uses the peripheral role with factory default settings.
2. Connect the module's UART_RX pin to the external host's UART_TX pin.
3. Connect the module's UART_TX pin to the external host's UART_RX pin.
4. *OPTIONAL:* Assert (LOW) the CYSPP pin to force CYSPP data mode in hardware, preventing API usage or output.
5. Apply power to the module, or reset it with the hardware reset pin.
6. If you have asserted (LOW) the CYSPP pin externally:
   a. Monitor the CONNECTION pin to detect when the remote peer has connected and GATT data subscription is complete.
   b. Once the CONNECTION pin goes LOW, you can send and receive data from the host to the remote peer over the module's serial connection.
7. Wait for the CYSPP status events.
   a. Wait for the p_cyspp_status (.CYSPP, ID=10/1) API event to appear with the LSB set indicating the data channel is ready. The final status event should appear as one of the following:

      `@E,000C,.CYSPP,S=21` (running in peripheral role)

      `@E,000C,.CYSPP,S=35` (running in central role)
   b. Send and receive data as desired.

Note: If you externally de-assert (HIGH) the CYSPP pin, then EZ-Serial will never enter CYSPP data mode even if a remote peer has connected and all CYSPP mode data pipe preparations have completed. The remote peer may use CYSPP on its end normally, but all data transfers and status updates will appear on the local EZ-Serial end as API events to be processed normally.

3.2.1.1 Starting CYSPP Out of the Box in Peripheral Mode

EZ-Serial's factory default configuration automatically starts CYSPP operation in the peripheral role after booting. To establish a CYSPP data pipe, simply scan and connect from a remote device, then subscribe to RX flow control (optional) and the desired acknowledged or unacknowledged data characteristic as described in

Sending and Receiving Data in CYSPP Data Mode.

A second EZ-Serial module running in CYSPP central/client mode will perform all required client-side steps automatically. As of version 1.1, EZ-Serial shows all GATT events relating to CYSPP setup until the CYSPP data pipe is fully opened.

Example 1: Complete boot and CYSPP connection process in peripheral mode

Direction	Content	Effect
←RX	`@E,0040,BOOT,E=01000305,S=02041A2A,P=0100,H=E2, C=00,A=CBCBB705DD05,T=01`	Boot event
←RX	`@E,000E,ASC,S=03,R=03`	CYSPP-triggered advertisement started
←RX	`@E,0035,C,C=04,A=00A050421650,T=00, I=0006,L=0000,O=0064,B=00`	Connection established with remote device
←RX	`@E,001A,W,C=04,H=0015,T=00,D=0200`	Remote client writes [02 00] to Client Characteristic Configuration Descriptor for RX flow control to enable indications from that characteristic.
←RX	`@E,000C,.CYSPP,S=04`	CYSPP status update (0x04): • 0x04: Subscribed to RX flow control
←RX	`@E,001A,W,C=04,H=0012,T=00,D=0100`	Remote client writes [01 00] to Client Characteristic Configuration Descriptor for unacknowledged data to enable notifications from that characteristic.
←RX	`@E,000C,.CYSPP,S=25`	CYSPP status update (0x05): • 0x04: Subscribed to RX flow control • 0x01: Subscribed to unacknowledged data • 0x20: CYSPP Data mode active
Host may now send data to the module for delivery to the remote peer; received data comes from peer.		

3.2.1.2 Starting CYSPP Out of the Box in Central Mode

Starting CYSPP client mode with factory default settings also requires no reconfiguration, since CYSPP mode will start automatically. However, you must assert (LOW) the CP_ROLE pin at boot time.

Example 1: Complete boot and CYSPP connection process in central mode

Direction	Content	Effect
←RX	`@E,0040,BOOT,E=01000305,S=02041A2A,P=0100,H=E2, C=00,A=CBCBB705DD05,T=01`	Boot event
←RX	`@E,000E,SSC,S=01,R=03`	CYSPP-triggered scan started
←RX	`@E,0062,S,R=00,A=00A050421650,T=00,S=D1,B=00,D= 020106 110700A10C2000089A9EE21115A13333336507 FF310100000000`	Scan result (advertisement fields separated for easier interpretation)
←RX	`@E,000E,SSC,S=00,R=03`	CYSPP-triggered scan stopped
←RX	`@E,0035,C,C=04,A=00A050421650,T=00, I=0006,L=0000,O=0064,B=00`	Connection established with remote device
←RX	`@E,0029,DR,C=04,H=0001,R=0007,T=2800,P=00,U=0018`	GATT discovery result (0x1800)
←RX	`@E,0029,DR,C=04,H=0008,R=000B,T=2800,P=00,U=0118`	GATT discovery result (0x1801)

Direction	Content	Effect
←**RX**	`@E,0045,DR,C=04,H=000C,R=0015,T=2800,P=00,U=00A10C2000089A9EE21115A133333365`	GATT discovery result (CYSPP service)
←**RX**	`@E,0045,DR,C=04,H=0016,R=001C,T=2800,P=00,U=00A20C2000089A9EE21115A133333365`	GATT discovery result (CYCommand service)
←**RX**	`@E,0010,RPC,C=04,R=060A`	Remote procedure complete
←**RX**	`@E,0029,DR,C=04,H=000C,R=0000,T=2800,P=00,U=0028`	GATT discovery result (service declaration)
←**RX**	`@E,0029,DR,C=04,H=000D,R=0000,T=2803,P=00,U=0328`	GATT discovery result (characteristic declaration)
←**RX**	`@E,0045,DR,C=04,H=000E,R=0000,T=0000,P=00,U=01A10C2000089A9EE21115A133333365`	GATT discovery result (CYSPP acknowledged data)
←**RX**	`@E,0029,DR,C=04,H=000F,R=0000,T=2902,P=00,U=0229`	GATT discovery result (configuration descriptor)
←**RX**	`@E,0029,DR,C=04,H=0010,R=0000,T=2803,P=00,U=0328`	GATT discovery result (characteristic declaration)
←**RX**	`@E,0045,DR,C=04,H=0011,R=0000,T=0000,P=00,U=02A10C2000089A9EE21115A133333365`	GATT discovery result (CYSPP unacknowledged data)
←**RX**	`@E,0029,DR,C=04,H=0012,R=0000,T=2902,P=00,U=0229`	GATT discovery result (configuration descriptor)
←**RX**	`@E,0029,DR,C=04,H=0013,R=0000,T=2803,P=00,U=0328`	GATT discovery result (characteristic declaration)
←**RX**	`@E,0045,DR,C=04,H=0014,R=0000,T=0000,P=00,U=03A10C2000089A9EE21115A133333365`	GATT discovery result (CYSPP RX flow control)
←**RX**	`@E,0029,DR,C=04,H=0015,R=0000,T=2902,P=00,U=0229`	GATT discovery result (configuration descriptor)
←**RX**	`@E,0010,RPC,C=04,R=0000`	Remote descriptor discovery complete
←**RX**	`@E,000C,.CYSPP,S=10`	CYSPP status update (0x10): • 0x10: CYSPP peer support verified
←**RX**	`@E,0017,WRR,C=04,H=0015,R=0000`	Remote server acknowledged the write operation that enabled indications on RX flow control characteristic.
←**RX**	`@E,000C,.CYSPP,S=14`	CYSPP status update (0x14): • 0x10: CYSPP peer support verified • 0x04: Subscribed to RX flow control
←**RX**	`@E,0018,D,C=04,H=0014,S=02,D=00`	Remote server pushes a “flow allowed” value via an indication from the RX flow control characteristic.
←**RX**	`@E,0017,WRR,C=04,H=0012,R=0000`	Remote server acknowledged write operation which enabled notifications on unacknowledged data characteristic
←**RX**	`@E,000C,.CYSPP,S=35`	CYSPP status update (0x15): • 0x10: CYSPP peer support verified • 0x04: Subscribed to RX flow control • 0x01: Subscribed to unacknowledged data • 0x20: CYSPP data mode active
Host may now send data to the module for delivery to the remote peer; received data comes from peer.		

3.3 Remote Control Examples with CYCommand

CYCommand provides a way to control EZ-Serial from a remote GATT client, using the same API protocol exposed over the wired serial interface. This allows use cases like remote provisioning during manufacturing and GPIO control. You can optionally require a password, a specific level of encryption and bonding, or both before a remote peer can control the module.

The CYCommand profile also provides an optional “safe mode” setting, which prohibits modifications to CYCommand settings over the remote-control interface. If enabled, this prevents locking yourself out by accidentally (or intentionally) disabling remote access. In this configuration, any reconfiguration using the p_cycommand_set_parameters (.CYCOMSP, ID=11/1) API command must occur over the wired serial interface.

Note: CYCommand is enabled in factory default settings to allow remote configuration simply by supplying power to the module and connecting from any remote peer. However, safe mode is disabled, so you can use the configuration API command remotely to disable CYCommand, if desired, either immediately or after performing initial provisioning steps.

When using CYCommand, make sure that the auto response is enabled. See gatts_set_parameters (SGSP, ID=5/14) for details on the auto response flag bit.

EZ-Serial implements the GATT server side of CYCommand behavior using the GATT structure detailed in CYCommand Profile.

Note: CYCommand access requires the module to be connectable for remote peers to use it. If you enable the CYCommand profile, but do not also enable connectable advertising via some other means, then remote configuration may still be or become inaccessible.

Methods to put the module into a connectable advertising state include using:

- gap_start_adv (/A, ID=4/8) sent from a host to advertise on demand.
- gap_set_adv_parameters (SAP, ID=4/23) to auto-start advertising on boot.
- p_cyspp_set_parameters (.CYSPPSP, ID=10/3) to auto-start peripheral role CYSSP operation.

3.3.1 Securing the CYCommand Profile

If you do not need to use CYCommand in your application, disable it with the p_cycommand_set_parameters (.CYCOMSP, ID=11/1) API command. This will hide the relevant GATT attributes from remote discovery and prevent any internal EZ-Serial application behavior that bridges CYCommand GATT operations to the API.

To retain CYCommand functionality requiring one or more levels of authentication before a client can send any API commands, use the `security`, `challenge`, and `secret` arguments of the p_cycommand_set_parameters (.CYCOMSP, ID=11/1) API command. You can select any combination of challenge types and security requirements; the API reference material for this command describes the options and behavior available with each configuration.

Example 1: Disable the CYCommand profile, store in flash

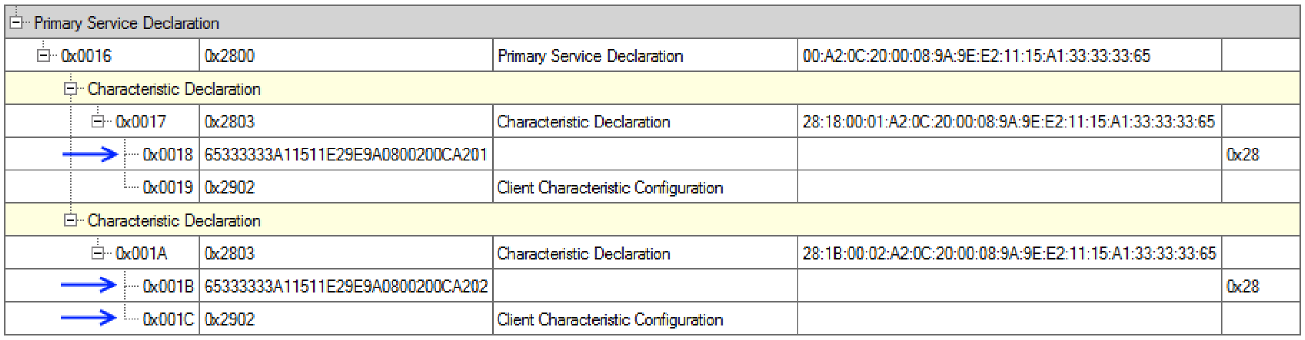
Direction	Content	Effect
TX→	`.CYCOMSP$,E=0`	Disable CYCommand, write to flash immediately
←RX	`@R,000F,.CYCOMSP$,0000`	Response indicates success

Example 2: Require CYCommand password “cypress”, store in flash

Direction	Content	Effect
TX→	`.CYCOMSP$,C=1,R=63797072657373`	Enable password challenge, set secret to “cypress” (hex)
←RX	`@R,000F,.CYCOMSP$,0000`	Response indicates success

3.3.2 Sending and Receiving API Commands over GATT

EZ-Serial implements the GATT server side of CYCommand behavior using the GATT structure detailed in CYCommand Profile. Figure 3-2 shows the CYCommand service structure as discovered and organized in the CySmart application, with the three most relevant attributes highlighted.



Primary Service Declaration				
0x0016	0x2800	Primary Service Declaration	00:A2:0C:20:00:08:9A:9E:E2:11:15:A1:33:33:33:65	
Characteristic Declaration				
0x0017	0x2803	Characteristic Declaration	28:18:00:01:A2:0C:20:00:08:9A:9E:E2:11:15:A1:33:33:33:65	
0x0018	65333333A11511E29E9A0800200CA201			0x28
0x0019	0x2902	Client Characteristic Configuration		
Characteristic Declaration				
0x001A	0x2803	Characteristic Declaration	28:1B:00:02:A2:0C:20:00:08:9A:9E:E2:11:15:A1:33:33:33:65	
0x001B	65333333A11511E29E9A0800200CA202			0x28
0x001C	0x2902	Client Characteristic Configuration		

Figure 3-2. CYCommand GATT Structure shown in CySmart Application

To use CYCommand from a client, perform the steps outlined below. These instructions assume that you have already enabled CYCommand and placed the module into a connectable advertising state. This is the factory default state after applying power to the module.

Note: While CYCommand data mode is active, you cannot send any API commands over the wired serial interface. EZ-Serial will buffer incoming API data (up to 136 bytes) and release it for parsing only after closing the CYCommand session. However, you can allow real-time transmission of outgoing response and event data that occurs during a CYCommand session, using the `hostout` argument of the p_cycommand_set_parameters (.CYCOMSP, ID=11/1) API command. This allows you to monitor remote activity from a local wired host device. The factory default configuration enables both response and event local host output during an active CYCommand session.

On the client side (Smartphone, CySmart Application, or another module):

1. Scan and connect to the EZ-Serial module from a client device.
2. Discover all GATT attributes or discovery services, and then discover all attributes within the CYCommand service.
3. Write value `[ 02 00 ]` (0x0002) to handle 0x001C (Client Characteristic Configuration for CYCommand Data). This subscribes to indications on CYCommand Data, allowing the module to send response and event data when it occurs.
4. If you have enabled a password challenge, write the password to handle 0x0018 (CYCommand Challenge).
5. Write API protocol commands as desired to handle 0x001B (CYCommand Data), and process response and event data indicated back via the same attribute. You can use either text mode or binary mode in the same way as you would over the wired serial interface.

Table 3-7 lists the example commands.

Commands	Binary Mode	Text Mode (Hexadecimal String)	Text Mode (Visible Characteristic)
system_ping (/PING, ID=2/1) Test communication	`C00002015C`	`2F50494E470A`	`/PING;`
gap_get_device_name (GDN, ID=4/16) Get the device name	`C00004106D`	`47444E0A`	`GDN;`
gpio_set_drive (SIOD, ID=9/5) Set P16 (bit 0 of the Port 1) pin drive modes to high-Z digital input	`C00B0905010100000000000000010075`	`2F53494F442C503D312C4D3D312C443D310A`	`SIOD,P=1,M=1,D=1;`
gpio_query_logic (/QIOL, ID=9/1) Query Port 2 logic state	`C00109010064`	`2F51494F4C2C503D300A`	`/QIOL,P=0;`

Table 3-7. Example Commands

On the server side (local EZ-Serial module):

The p_cycommand_status (.CYCOM, ID=11/1) API event will occur one or more times as the client performs the steps listed above. Once the CYCommand status value has the LSB set (0x01), the client can control the module remotely, and EZ-Serial will disconnect the local serial interface from the API parser.

This same API event will occur once when the client disconnects or unsubscribes from the CYCommand Data characteristic, indicating to the server that it can resume local control. At that moment, EZ-Serial will process any buffered API data previously sent from the host during the active CYCommand session.

3.4 GAP Peripheral Examples

GAP peripheral operation is one of the most common use cases for BLE designs, since it is usually the simplest way to communicate with a smartphone operating as a central device.

The Bluetooth specification defines different types of roles for the devices on each end of a BLE link:

- Link layer
  - Master – Device which initiates a connection (always GAP central).
  - Slave – Device which accepts a connection (always GAP peripheral).
- GAP layer
  - Central – Device which initiated a connection (always LL master).
  - Peripheral – Device which accepted a connection (always LL slave).
  - Broadcaster – Device which is advertising in a non-connectable state.
  - Observer – Device which is scanning without initiating a connection.
- GATT layer
  - Client – Device which accesses data from a remote GATT server.
  - Server – Device which provides attribute data to be accessed remotely.

Link layer roles are defined when a connection is initiated based on the side that initiates the connection.

The GAP layer provides four roles, two of which involve connections (central and peripheral) and two of which are connectionless (broadcaster and observer). The link layer and GAP layer roles are closely related, particularly when a connection is involved.

The GATT layer role is independent of other behavior. A single device may even perform GATT duties in both the client and server roles. A common example of this is an iOS device providing the Apple Notification Center Service as a GATT server, even though it is connected to a peripheral device and acting as a GATT client to that device.

3.4.1 Advertising as Peripheral Device

Advertising is the BLE activity, which allows scanning devices to observe and connect to peripherals. It is required for a connection to be initiated, but it may also be done in a non-connectable way (called “broadcasting”). EZ-Serial supports non-connectable broadcasting even while connected.

EZ-Serial gives you full control over when and how to advertise using the gap_start_adv (/A, ID=4/8) API command and the gap_set_adv_parameters (SAP, ID=4/23) API command.

When the advertising state changes, the gap_adv_state_changed (ASC, ID=4/2) API event occurs. This event includes the new state as well as a code showing the reason why the state changed.

Note: If you do not have any automatic advertisement timeout set, then advertisements will continue until you explicitly stop them or a remote device initiates a connection.

In text mode, all arguments to the gap_start_adv (/A, ID=4/8) API command are optional. Any supplied arguments will be used only for the immediate advertisement that begins because of the command, while any omitted arguments will fall back to the values configured by the gap_set_adv_parameters (SAP, ID=4/23) API command. You can see these values at any time by using the gap_get_adv_parameters (GAP, ID=4/24) API command.

Example 1: Start advertising with preconfigured default parameters

Direction	Content	Effect
TX→	`/A`	Begin advertising with preconfigured defaults
←RX	`@R,0008,/A,0000`	Response indicates success

Direction	Content	Effect
←RX	`@E,000E,ASC,S=03,R=00`	Event indicates advertising state changed to "Connectable Undirected High"

Example 2: Start advertising with custom parameters

Direction	Content	Effect
TX→	`/A,M=5,O=1E`	Begin advertising with custom arguments, such as non-connectable, undirected high duty advertising, and stop after 30 seconds.
←RX	`@R,0008,/A,0000`	Response indicates success
←RX	`@E,000E,ASC,S=05,R=00`	Event indicates advertising state changed to "Active Undirected Low"
←RX	`@E,000E,ASC,S=00,R=02`	Advertising stopped caused by 30 seconds timeout.

3.4.2 Stopping Advertisement as Peripheral Device

To explicitly stop advertising, use the gap_stop_adv (/AX, ID=4/9) API command, or open a connection to the module from a remote BLE central device.

Example 1: Stop advertising

Direction	Content	Effect
TX→	`/AX`	Stop advertising
←RX	`@R,0009,/AX,0000`	Response indicates success
←RX	`@E,000E,ASC,S=00,R=00`	Event indicates advertising state changed to "inactive" due to user request

3.4.3 Customizing Advertisement and Scan Response Data

You can customize the content of the main advertisement payload and scan response payload with the gap_set_adv_data (SAD, ID=4/19) and gap_set_sr_data (SSRD, ID=4/21) API commands, respectively.

Note: If you intend to use user-defined advertisement content, you must explicitly enable this in the advertisement parameters. Normally, the EZ-Serial platform manages the content in the advertisement and scan response packets automatically based on the platform configuration, including the device name and the profiles that are enabled. If you set custom content but do not configure EZ-Serial to use that content, advertisement and scan response payloads will remain automatically managed.

Following are the key features and requirements for customizing data:

- Each advertisement and scan response packet payloads may have a maximum of 31 bytes. This is a BLE specification limit.
- Advertisement data in both packets should follow the correct [Length, Type, Value...] format required by the Bluetooth specification. Malformed data within advertisements can prevent proper scanning by remote devices. The Length value does not include itself, but does include the Type byte and all bytes in the remaining Value data.
- Each packet may contain as many fields as will fit in 31 bytes. Place multiple fields one right after the other with no special separator. Since each field begins with a "length" value, a scanning device is always able to properly identify the end of each field.
- Advertisement packets include the Bluetooth connection address (public or random) outside of the payload data. This does not count towards the 31-byte limit.
- The main advertisement packet is always transmitted while advertising. It typically includes connectable flags, important supported service UUIDs, and a custom manufacturer data field. Place any data that is critical for the remote device to see inside the main advertisement packet.
- The scan response packet is only transmitted when a remote device is performing an active scan. During an active scan, the scanning device send a scan request to any discovered advertising device immediately after receiving the main advertisement packet. The scan response packet typically includes the friendly name of the advertising device, and occasionally also includes transmit power, more manufacturer data, or other useful but less critical data that a remote scanning device may not need to see.

Detailed information on approved field types and their intended contents can be found the Bluetooth specification. Table 3-8 lists the most commonly used fields.

Type	Description	Value
0x01	Flags field – 1 byte of data	1 byte (bitfield)
0x02	Partial list of 16-bit UUIDs for supported GATT services	2*N bytes (UUIDs)
0x03	Complete list of 16-bit UUIDs for supported GATT services	2*N bytes (UUIDs)
0x04	Partial list of 32-bit UUIDs for supported GATT services	4*N bytes (UUIDs)
0x05	Complete list of 32-bit UUIDs for supported GATT services	4*N bytes (UUIDs)
0x06	Partial list of 128-bit UUIDs for supported GATT services	16*N bytes (UUIDs)
0x07	Complete list of 128-bit UUIDs for supported GATT services	16*N bytes (UUIDs)
0x08	Shortened local name	0-29 bytes (Text string)
0x09	Complete local name	0-29 bytes (Text string)
0x0A	TX power level	1 byte (dBm as signed integer)
0xFF	Manufacturer data	3-29 bytes (company ID + data)

Table 3-8. Common Advertisement Field Types

EZ-Serial does not validate advertisement or scan response payload content, and the underlying BLE/BT stack has only limited validation on the **Flags** field. Make sure that any customized data within either of these packets is correctly formatted. While the module will transmit whatever payload data is configured, scanning devices may not correctly identify your device if the data is malformed or missing (especially the **Flags** field).

The stack requires that the **Flags** field, if present, must have the final two bits set so that they match the Discovery Mode setting used when starting advertisements. For BLE and classic BR/EDR support Bluetooth devices, this means that the flags byte will almost always be one of the following three values:

- 0x00: Non-discoverable/broadcast-only (common for beacon-only devices)
- 0x01: Limited discoverable
- 0x02: General discoverable (most common for connectable devices)

See gap_start_adv (/A, ID=4/8) API command for additional reference on discoverable modes.

Table 3-9 provides examples for reference.

Byte content	Field Description
`02 01 02`	**Length:** 2 bytes **Type:** Flags (0x01) **Value:** LE General Discoverable Mode, BR/EDR Supported
`05 02 09 18 0D 18`	**Length:** 5 bytes **Type:** Complete list of 16-bit UUIDs for supported GATT services (0x02) **Value:** 0x1809 (Health Thermometer), 0x180D (Heart Rate)
`07 08 57 69 64 67 65 74`	**Length:** 7 bytes **Type:** Shortened local name (0x08) **Value:** “Widget”
`09 FF 31 01 AA BB CC DD EE FF`	**Length**: 9 bytes **Type:** Manufacturer data (0xFF) **Value:** Company ID = 0x0131 (Cypress Semiconductor) Data = `[AA BB CC DD EE FF]`

Table 3-9. Examples of Well-Formed Advertisement Fields

These four example fields require 25 bytes when combined, including each of the four Length values. They can be placed in a single advertisement packet if desired:

```
02 01 02 05 02 09 18 0D 18 07 08 57 69 64 67 65 74 09 FF 31 01 AA BB CC DD EE FF
```

Here, the shortened name is included in the same packet as the more critical information. This is uncommon, but not prohibited. The name typically goes in the scan response packet because there it cannot fit into the advertisement packet, but any field may be in any location if the scanning device knows what to expect.

Example 1: Set custom advertisement and scan response data

Direction	Content	Effect
TX→	`SAP,F=2`	Enable user-defined advertisement and scan response content
←RX	`@R,0009,SAP,0000`	Response indicates success
TX→	`SAD,D=020102050209180D18`	Set new advertisement content (RAM only), Flags and 16-bit UUID fields
←RX	`@R,0009,SAD,0000`	Response indicates success
TX→	`SSRD,D=0708576964676574`	Set new scan response content (RAM only), Complete local name field
←RX	`@R,000A,SSRD,0000`	Response indicates success

Example 2: Set advertisement and scan response data to value similar to factory defaults

Direction	Content	Effect
TX→	`SAP,F=1`	Enable user-defined advertisement and scan response content
←RX	`@R,0009,SAP,0000`	Response indicates success
TX→	`SAD,D=020102110700a10c2000089a9ee21115a133333365`	Set new advertisement content (RAM only)
←RX	`@R,0009,SAD,0000`	Response indicates success
TX→	`SSRD,D=1309455a2d53657269616c2045333a38333a3546`	Set new scan response content (RAM only)
←RX	`@R,000A,SSRD,0000`	Response indicates success

3.5 GAP Central Examples

Running as a GAP central allows you to scan for and connect to remote peripheral devices. You can also operate as a GAP observer by scanning without any subsequent connection attempts. For further discussion of various link-layer, GAP, and GATT roles, see GAP Peripheral Examples.

3.5.1 Scanning for Peripheral Devices

Use the gap_start_scan (/S, ID=4/10) API command to begin scanning for devices. Scanning is not required before initiating a connection, but doing so helps to identify potential connection targets or to make sure that known or compatible peripherals are nearby and connectable.

Note: If you do not have any automatic scan timeout set, scanning will continue until you explicitly stop it. Scanning will not automatically resume when a connection is terminated unless CYSPP is enabled in the central role. Otherwise, you must implement this behavior in your application logic as needed. You must stop scanning before you can initiate an outgoing connection to a remote peer. Requesting a connection with gap_connect (/C, ID=4/1) while scanning will result in an error.

In text mode, all arguments to the gap_start_scan (/S, ID=4/10) API command are optional. Any supplied arguments will be used only for the immediate scan that is started because of the command, while any omitted arguments will fall back to the values configured by the gap_set_scan_parameters (SSP, ID=4/25) API command. You can see these values at any time by using the gap_get_scan_parameters (GSP, ID=4/26) API command.

After you start scanning, EZ-Serial will begin generating the gap_scan_result (S, ID=4/4) API events each time a new advertisement packet is seen from a remote device. The same advertising device will generate multiple scan results until duplicate filtering is enabled in the scan parameters.

Passive versus Active Scanning:

- During a passive scan, EZ-Serial will not send scan requests to devices to ask for the “follow-up” scan response packet. In this mode, each device generates only one event for each detected advertisement packet. Passive scans use less power on average, since the transmitter remains inactive and the receiver is not intentionally re-activated for a second time for the same device.

- During an active scan, EZ-Serial sends a scan request to obtain additional information from the remote peripheral. In this mode, the BLE stack may generate two events for each device detected during a scan. However, the remote device may not send the scan response packet, or the local device may not receive it due to adverse RF conditions, so a second scan result event is not guaranteed. Active scans use more power that passive scans, and result in brief transmission bursts in between receive operations.

WARNING: Due to the precise timing required by the BLE protocol and the way active scans behave, many actively scanning devices in the same vicinity can result in none of the scanning devices successfully obtaining a scan response from an advertising device. If two or more scanning devices transmit a scan request on the same channel within the same ~150 µs window immediately after the main advertisement packet, the advertising device will not be able to parse the request and will not send a response to either device. This unlikely but possible issue does not occur while performing a passive scan.

Example 1: Start passive scanning with preconfigured default parameters

Direction	Content	Effect
TX→	`/S`	Begin scanning with preconfigured defaults
←RX	`@R,0008,/S,0000`	Response indicates success
←RX	`@E,000E,SSC,S=01,R=00`	Event indicates scanning state has changed to “active” due to user request
←RX	`@E,0052,S,R=00,A=00A050E3835E,T=00,S=D1,B=00, D=0201061107CA366D7D5BCC0288B14DE541D9FF652F`	Event indicates scan result from 00:A0:50:E3:83:5E, normal ad packet, RSSI -47 dBm (0xB1), Flags field and 128-bit UUID

Example 2: Start 5-second active scan with duplicate filtering enabled

Direction	Content	Effect
TX→	`/S,M=2,A=1,D=1,O=5`	Begin “observation” scanning, active mode, 5-second timeout, duplicate filter enabled
←RX	`@R,0008,/S,0000`	Response indicates success
←RX	`@E,000E,SSC,S=01,R=00`	Event indicates scanning state has changed to “active” due to user request
←RX	`@E,0052,S,R=00,A=00A050E3835E,T=00,S=D1,B=00 D=0201061107CA366D7D5BCC0288B14DE541D9FF652F`	Event indicates scan result from 00:A0:50:E3:83:5E, ad packet, RSSI -47 dBm (0xB1), Flags field and 128-bit UUID
←RX	`@E,004E,S,R=04,A=00A050E3835E,T=00,S=D1,B=00 D=1209426C7565666C6F772037383A46353A4236`	Event indicates scan result from 00:A0:50:E3:83:5E, scan response packet, RSSI -47 dBm, Local name field
←RX	`@E,000E,SSC,S=00,R=02`	Event indicates scanning state has changed to “stopped” due to configured timeout (5 seconds)

3.5.2 Stopping Scanning for Peripheral Devices

To explicitly stop scanning, use the gap_stop_scan (/SX, ID=4/11) API command, or initiate a connection request to a remote device using the gap_connect (/C, ID=4/1) API command.

WARNING: It is possible for additional gap_scan_result (S, ID=4/4) API events to occur between a successful response to the gap_stop_scan command and the gap_scan_state_changed event (“SSC” in text mode), due to the brief amount of time that it takes the stack to process the request and change states. Make sure that your application logic will not fail in this case.

Example 1: Stop scanning

Direction	Content	Effect
TX→	`/SX`	Stop scanning
←RX	`@R,0009,/SX,0000`	Response indicates success
←RX	`@E,000E,SSC,S=00,R=00`	Event indicates scanning state has changed to “inactive” due to user request

3.5.3 Connecting to a Peripheral Device

Use the gap_connect (/C, ID=4/1) API command to initiate a connection to a remote device based on its Bluetooth connection address. The Bluetooth connection address (also commonly referred to as a MAC address) is made up of the 6-byte device address and a 1-byte value indicating the address type. To initiate a connection, the module must be in a disconnected state (not advertising, scanning, connecting, or connected).

Note: Now, the EZ-Serial firmware platform supports one active connection at a time. To transfer data to and from multiple devices quickly, you must establish and tear down connections in rapid succession. With a fast advertisement interval on peripheral devices and a fast connection interval while connected, it is possible to perform many connect-transfer-disconnect cycles per second.

Addresses may be either public or random. Public addresses do not change, while random addresses change on some period determined by the device employing privacy measures (typically at least every few minutes). The use of random addresses, also called private addresses, reduces the possibility of passive profiling by a remote device. For example, iOS devices always use random addressing for BLE operations. EZ-Serial supports both types, and uses public addressing by default. For more information on this topic and how to configure EZ-Serial to use random addressing, see Using Peripheral and Central Privacy.

When a BLE device initiates a connection request, it does not immediately transmit anything. Rather, it must first scan until it receives a connectable advertisement packet from the target device. Therefore, a peripheral device must be in an advertising state to accept a connection. The full connection process includes the following steps:

1. Target peripheral device is advertising in a connectable state.
2. Central device begins scanning for advertisements from target peripheral device.
3. Central device detects advertisement and responds with connection request.
4. Peripheral device receives connection request and responds with connection response.
5. Connection is fully established.
6. Configure the MTU size if required. By default, the new established MTU size is 23, which means the data size is 20 bytes.

The API command used to initiate a connection includes arguments for scan parameters, because scanning is the first operation that the stack must perform on the GAP central device during a connection process.

Example 1: Connect to a remote device using default connection parameters

Direction	Content	Effect
TX→	`/C,A=00A050E3835E,T=01`	Initiate connection
←RX	`@R,000D,/C,0000`	Response indicates success
←RX	`@E,0035,C,C=01,A=CBCBB705DD05,T=01,I=0027,L=0000,O=02BC,B=00`	Event indicates connection opened
TX→	`/GCMTU,C=01,M=0203`	Configure the MTU size to 515
←RX	`@E,0010,MTU,C=01,M=0203`	Negotiated MTU size is 515

3.5.4 Cancelling Pending Connection to a Peripheral Device

Use the gap_cancel_connection (/CX, ID=4/2) API command to cancel a pending outgoing connection request. This only applies when the connection is not yet open and you have not received the gap_connected (C, ID=4/5) API event. If you need to close an open connection, use the gap_disconnect (/DIS, ID=4/5) API command.

Example 1: Cancel a pending connection to a remote device

Direction	Content	Effect
TX→	`/CX`	Cancel pending connection
←RX	`@R,0009,/CX,0000`	Response indicates success
←RX	`@E,0010,DIS,H=00,R=091F`	Event indicates connection canceled

3.5.5 Disconnecting from a Peripheral Device

Use the gap_disconnect (/DIS, ID=4/5) API command to close an active connection to a remote device. This only applies when the connection is already fully established, and should not be used to cancel a pending outgoing connection. In that case, use the gap_cancel_connection (/CX, ID=4/2) API command.

Example 1: Disconnect from a remote device

Direction	Content	Effect
TX→	`/DIS,C=01`	Disconnect from peer. The value of parameter C is the value returned from the /C command response.
←RX	`@R,000A,/DIS,0000`	Response indicates success
←RX	`@E,0010,DIS,C=01,R=0916`	Event indicates connection closed, reason=0x0916 (intentional local closure)

3.6 GATT Server Examples

BLE data transfer operations between two connected devices most often occur through the GATT layer, with a server on one side and a client on the other side. The GATT server makes use of a pre-defined attribute structure, which the client may remotely discover and use as needed. The GATT server defines what data is available and how it may be accessed, and has limited ability to push data to the client if the client has subscribed to receive these types of updates.

3.6.1 Defining Custom Local GATT Services and Characteristics

EZ-Serial implements a dynamic GATT structure that can be modified at runtime and stored in flash. Note that the structure itself is the part that is stored in flash; values stored within data characteristics (other than default values defined when creating new entries) are stored in RAM only. The dynamic GATT structures can be stored to flash when explicitly calling the gatts_store_db (/SGDB, ID=5/4) or system_store_config (/SCFG, ID=2/4) command.

The EZ-Serial platform contains a few pre-defined GATT elements in the factory default configuration. EZ-Serial requires these elements for correct operation, and cannot be removed or modified. However, additional structural elements are entirely customizable.

A GATT structure is fundamentally made up of individual attributes, each having a unique numeric handle, a UUID that is 16 bits, 32 bits, or 128 bits wide, and a value container. Attribute handles start at 1 and may go up to 0xFFFF (65535). No two attributes may have the same handle. The gatts_create_attr (/CAC, ID=5/1) API command will automatically choose the next available attribute handle and report the value in the response after a successful command.

WICED EZ-Serial firmware internally use three structures to store individual attributes:

- `gatts_db[]` is an array of GATT entry structures containing the fixed-length portion of each entry (type, permissions, length, and the 16-bit length prefix value from the data array).
- `gatts_db_const_data[]` is an array of UINT8 bytes containing the variable-length portion of each entry (the payload from the data array).
- `gatts_db_user_create_data[] is an array of UINT8 bytes containing the variable-length portion of each attribute characteristic value (The default characteristic read values can be stored in this data array).`

EZ-Serial provides the gatts_create_attr (/CAC, ID=5/1) API command to create a new custom attribute, which in the WICED EZ-Serial firmware takes the following arguments:

```
uint8 type
uint8 permissions
```

```
uint16 length
longuint8a data
```

The first six bytes of this packet structure (through the 16-bit length prefix on data) is a match for the GATT entry structure. Any payload data in the data structure goes in the constant data pool instead.

To use this correctly, you must have some prior knowledge of correct GATT structures, especially in the case of a characteristic declaration which includes additional metadata beyond the value attribute's UUID. The following pseudocode demonstrates how you would use this command to add the Generic Access service with a writeable Device Name value (though in EZ-Serial you should not do this since it is part of the default structure and cannot be removed):

Note: Multiple data should set in little-endian order.

```
// syntax: gatts_create_attr(type, permissions, length, data[])
```

1. Create Generic Access Service. (Handle: 0x0001)

```
gatts_create_attr(0, 0x02, 0x04,  [0x00, 0x28, 0x00, 0x18])
```

Details of this Example:

Type	0	(Structure Entry)
Permissions	*0x02*	(Readable)
Length	0x04	(2 bytes Type UUID + 2-byte SIG Service UUID)
Data	*0x00, 0x28*, *0x00, 0x18*	
Attribute Type	*0x00, 0x28*	(Primary Service Declaration: 0x2800)
UUID	*0x00, 0x18*	(Generic Access Service UUID: 0x1800)

2. Create Device Name (permission: Read) Characteristic. (Handle: 0x0002)

```
gatts_create_attr(0, 0x02, 0x07,  [0x03, 0x28, 0x02, 0x03, 0x00, 0x00, 0x2A])
```

Details of this Example:

Type	0	(Structure Entry)
Permissions	*0x02*	(Readable)
Length	0x07	(2 bytes Type UUID + 1 byte Properties + 2-byte Attribute Handle + 2 bytes SIG Characteristic UUID)
Data	*0x03, 0x28*, *0x02*, *0x03, 0x00*, *0x00, 0x2A*	
Attribute Type	*0x03, 0x28*	(Characteristic Declaration)
Properties	*0x02*	(Characteristic Properties: Read)
Handle	*0x03, 0x00*	(Characteristic Handle: 0x0003)
UUID	*0x00, 0x2A*	(Device Name Characteristic UUID: 0x2A00)

3. Create the value of Device Name Characteristic, maximum 64 bytes, permission: Read, Write_REQ, Variable_len. (Handle: 0x0003)

```
gatts_create_attr(1, 0x0B, 0x40, [])
```

Details of this Example:

Type	1	(Characteristic Value Entry)
Permissions	*0x0B*	(Readable, Write request, Variable length)
Length	0x04	(FW allocates 64 bytes space for storing the data)
Data	None	

4. Create Appearance (permission: Read) Characteristic. (Handle: 0x0004)

```
gatts_create_attr(0, 0x02, 0x07,  [0x03, 0x28, 0x02, 0x05, 0x00, 0x01, 0x2A])
```

5. Create the value of Appearance Characteristic, fixed 2 bytes, permission: read. (Handle: 0x0005)

```
gatts_create_attr(1, 0x02, 0x02,  [])
```

6. Create a custom service. (Handle: 0x0006)

Service UUID: aa001020304050607080901020304020.

```
gatts_create_attr(0, 0x02, 0x12,  [0x00, 0x28, 0xaa, 0x00, 0x10, 0x20, 0x30, 0x40,
0x50, 0x60, 0x70, 0x80, 0x90, 0x10, 0x20, 0x30, 0x40, 0x20])
```

Details of this Example:

Type	0	(Structure Entry)
Permissions	*0x02*	(Readable)
Length	0x12	(2 bytes Type UUID + 16 bytes Service UUID)
Data	*0x00, 0x28, 0xaa, 0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0x10, 0x20, 0x30, 0x40, 0x20*	
Attribute Type	*0x00, 0x28*	(Primary Service Declaration)
UUID	*0xaa, 0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0x10, 0x20, 0x30, 0x40, 0x20*	(Custom Service UUID)

7. Create a custom Characteristic within the customer service. (Handle: 0x0007)

Characteristic UUID: aa011020304050607080901020304020. Permission: Read, Write Request, Notify, and Indicate.

```
gatts_create_attr(0, 0x02, 0x15,  [0x03, 0x28, 0x3A, 0x08, 0x00,

0xaa, 0x01, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0x10, 0x20, 0x30,
0x40, 0x20])
```

Details of this Example:

Type	0	(Structure Entry)
Permissions	*0x02*	(Readable)
Length	0x15	(21 bytes GATT constant data = 2 bytes Type UUID + 1 byte Properties + 2-byte Attribute Handle + 16 bytes SIG Characteristic UUID)
Data	*0x03, 0x28, 0x02, 0x03, 0x00, 0xaa, 0x01, 0x10, 0x20, 0x30, 0x40, 0x50,*	

	0x60, 0x70, 0x80, 0x90, 0x10, 0x20, 0x30, 0x40, 0x20	
Attribute Type	*0x03, 0x28*	(Characteristic Declaration)
Properties	*0x02*	(Characteristic Properties: Read)
Handle	*0x08, 0x00*	(Custom Characteristic Value Attribute Handle: 0x0008)
UUID	*0xaa, 0x01, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0x10, 0x20, 0x30, 0x40, 0x20*	(Custom characteristic UUID)

8. Create the value attribute of the custom Characteristic, 1 bytes, permission: Read, Write Request. (Handle: 0x0008)

```
gatts_create_attr(1, 0x8A, 0x01, [])
```

Details of this Example:

Type	1	(Characteristic Value Entry)
Permissions	*0x8A*	(Service_UUID_128, Readable, Write request)
Length	0x01	(The attribute value size is fixed and 1 byte)
Data	None	(FW allocates 1- byte space for storing the data)

9. Create Characteristic Client Configuration Descriptor for the custom Characteristic. (Handle: 0x0009)

   Permission: Readable, Write Request.

```
gatts_create_attr(0, 0x0A, 0x04,  [0x02, 0x29])
```

Details of this Example:

Type	1	(Structure Entry)
Permissions	*0x0A*	(Readable, Write request)
Length	0x04	(The attribute value has fixed 4 bytes = 2 bytes Type UUID + 2 bytes of the CCCD value)
Data	*0x02, 0x29*	The Structure Entry supplies 2 bytes for GATT data, which is used to store the CCCD attribute value data. So, FW will allocate 2 bytes space to store the 2 bytes of CCCD attribute value data.)

Each characteristic declaration (2nd, 4th, and 6th entries) contain the 0x2803 structural UUID, 0x02 properties byte, then the 16-bit attribute handle (0x0003, 0x0005, and 0x0008 respectively) corresponding to the value attribute. EZ-Serial requires the value attribute to come immediately after the declaration.

WARNING: Modifications to the custom GATT structure require flash write operations, which can potentially disrupt BLE connectivity. Therefore, you should only make changes to the GATT database while there is no active BLE connection to avoid the possibility of a connection loss.

3.6.1.1 Understanding Custom GATT Limitations

The dynamic GATT implementation in EZ-Serial contains some built-in entries to provide required EZ-Serial functionality, leaving the remaining space available for custom entries. Table 3-10 lists the space left for customer entries.

Category	Built-in	CYBT-4130XX-02	
		Total	Available
GATT DB entry structures	37	128	91
Space for GATT DB entry constant data	272 bytes	1024 bytes	752 bytes
Space for GATT DB entry attribute values	0 bytes	1204 bytes	840 ~ 1200 bytes

Category	Built-in	CYBT-4130XX-02	
		Total	Available
(Note, each DB entry attribute value will take four additional header bytes, so maximum available bytes are (1024 – (available entries * 4) bytes)			

Table 3-10. Dynamic GATT Structural Limitations

Attempting to create a new custom attribute which exceeds any of these bounds will generate an error result indicating the nature of the limitation. See Error Codes for details.

3.6.1.2 Building Custom Services and Characteristics

The GATT database is made up of one or more primary services. Each primary service has a service declaration (UUID 0x2800) and includes one or more characteristics. Each characteristic has a characteristic declaration (UUID 0x2803) and a value attribute (any UUID not in the above list), and often has additional characteristic-related descriptors in the 0x2900 range.

UUIDs indicate the purpose of each attribute, but may be (and often are) repeated through the complete database. For example, a database containing three services will contain three separate attributes that contain UUID 0x2800 (UUID 0x2800 indicating official “Primary Service Declaration” UUID defined by the Bluetooth SIG). Table 3-11 lists notable pre-defined structural definition UUIDs from the Bluetooth SIG.

UUID	Description
0x2800	Primary Service Declaration
0x2801	Secondary Service Declaration
0x2802	Include Declaration
0x2803	Characteristic Declaration
0x2900	Characteristic Extended Properties
0x2901	Characteristic User Description
0x2902	Client Characteristic Configuration
0x2903	Server Characteristic Configuration
0x2904	Characteristic Format
0x2905	Characteristic Aggregate Format

Table 3-11. Bluetooth SIG Structural UUIDs

For more details, see the Bluetooth SIG website.

When defining GATT elements at runtime, you must enter each attribute in the correct order based on the desired structure. Any entries that do not conform to the correct order requirement will be rejected with a validation error. The only case where a validation warning is allowed is when you define a new service or characteristic declaration and have not yet entered the subsequent attributes which must follow. You can use the gatts_validate_db (/VGDB, ID=5/3) API command at any time to perform an integrity check on the current GATT structure to see whether additional attributes are expected.

The required order for each complete characteristic definition (declaration, value, and optional descriptors) is dictated by the internal BLE stack as listed in Table 3-12.

Order	UUID	Description	Required
#1	0x2803	Characteristic Declaration	**Yes**
#2	<custom>	Characteristic Value	**Yes**
#3	0x2900	Characteristic Extended Properties	No
#4	0x2901	Characteristic User Description	No
#5	0x2902	Client Characteristic Configuration	No
#6	0x2903	Server Characteristic Configuration	No

Order	UUID	Description	Required
#7	0x2904	Characteristic Format	No
#8	0x2905	Characteristic Aggregate Format	No

Table 3-12. Required Characteristic Attribute Order

Any optional (Required mask as No) attributes may be omitted if all provided attributes are supplied in the order listed in Table 3-12.

For details on how to use custom GATT creation API commands to add support for Bluetooth SIG official services such as Device Information, Health Thermometer, and others, see Adopted Bluetooth SIG GATT Profile Structure Snippets and the API reference material for gatts_create_attr (/CAC, ID=5/1).

3.6.1.3 Choosing Correct GATT Permissions and Properties

It is critical to use correct permissions when defining any custom GATT structural elements. See Adopted Bluetooth SIG GATT Profile Structure Snippets for example definitions, and you may notice certain patterns. Here are the recommended guidelines for the most common entries:

- Service declarations (type = 0x2800)
  - Read permissions = 0x01, to allow structure discovery (no encryption/authentication)
  - Write permissions = 0x00, to prevent attempted changes
  - Characteristic properties = 0x00, because the properties do not apply
- Characteristic declarations (type = 0x2803)
  - Read permissions = 0x01, to allow structure discovery (no encryption/authentication)
  - Write permissions = 0x00, to prevent attempted changes
  - Characteristic properties = <actual properties>
- Characteristic value attributes (type = 0x0000)
  - Read permissions = <actual permissions>
  - Write permissions = <actual permissions>
  - Characteristic properties = <actual properties, matching 0x2803 declaration>
- Characteristic user description attributes (type = 0x2901)
  - Read permissions = 0x01, to allow reading description
  - Write permissions = 0x00, to prevent attempted changes
  - Characteristic properties = 0x02 (read)
- Client characteristic configuration attributes (type = 0x2902)
  - Read permissions = 0x01, to allow reading current client flags
  - Write permissions = 0x01, to allow configuring new client flags
  - Characteristic properties = 0x0A (read + write)

Table 3-13 and Table 3-14 list the detailed values of the Attribute Permissions and Characteristic Properties, respectively.

Permissions	Description
0x00	None
0x01	Variable length
0x02	Readable
0x04	Write command (unacknowledged)
0x08	Write request (acknowledged)

Permissions	Description
0x10	Authentication Readable
0x20	Reliable Writable
0x40	Authentication Writable
0x80	128-bit Service UUID

Table 3-13. GATT Database Attribute Permissions Definitions

Properties	Description
0x01	Broadcast
0x02	Read
0x04	Write without response (unacknowledged)
0x08	Write (acknowledged)
0x10	Notify
0x20	Indicate
0x40	Authorized write
0x80	Extended Properties

Table 3-14. GATT Database Characteristic Properties Definitions

In general, structural elements such as service and characteristic declarations should be read-only, but should have no particular security restrictions on them. This ensures that a connected client can discover the database structure correctly, even if additional security is required to execute read, write, or both operations on the characteristic value attributes. Some Android devices are known to have problems during discovery if the declaration descriptors themselves have extra security requirements.

Note: Any attribute that requires authentication (bonding) must also require encryption. If you enable the authentication bit, make sure that you also enable the encryption bit, or the command will be rejected with an error result. For more information on the GATT database attribute permissions and characteristic properties, see the Bluetooth Specification Version 5.0 | Vol 3, Part F, 3.2.5 Attribute Permissions.

3.6.2 Listing Local GATT Services, Characteristics, and Descriptors

Listing the local GATT structure can be helpful in certain cases, even though it is typically the remote GATT structure that requires discovery (see Discovering a Remote Server's GATT Structure). This is especially true since you can dynamically change the local GATT structure at runtime. EZ-Serial provides three commands for local discovery, each of which provides output equivalent to its "remote discovery" counterpart.

Local discovery differs from remote discovery in two key ways:

1. Local discovery is instant and deterministic, while remote discovery is not. Remote discovery generates an unknowable number of result events over a relatively slow BLE connection, with completion indicated via the gattc_discover_characteristics (/DRC, ID=6/2) API event. In contrast, local discovery returns the known result count as part of the response to the discover request, and then generates exactly that many discovery result events without a final "complete" event (which would be redundant).
2. When discovering local descriptors, the output includes some extra information in results which is not provided during an equivalent remote descriptor discovery process. Specifically:
    a. All descriptors include the "properties" value. In remote results, this will always be 0.
    b. Service declarations include the end handle. In remote results, this will always be 0.
    c. Characteristic declarations include the value attribute handle. In remote results, this will always be 0.

3.6.2.1 Discovering Local GATT Services

Use the gatts_discover_services (/DLS, ID=5/6) API command to obtain a list of services in the local GATT database.

Example 1: Local GATT service discovery with factory default structure (no custom attributes)

Direction	Content	Effect
TX→	`/DLS`	Request to discover all local services
←RX	`@R,0011,/DLS,0000,C=0005`	Response indicates success, five records to follow
←RX	`@E,0024,DL,H=0001,R=0007,T=2800,P=00,U=0018`	Service 0x1800, start=0x0001, end=0x0007
←RX	`@E,0024,DL,H=0008,R=000B,T=2800,P=00,U=0118`	Service 0x1801, start=0x0008, end=0x000B
←RX	`@E,0040,DL,H=000C,R=0015,T=2800,P=00, U=00A10C2000089A9EE21115A133333365`	Service 0x6533…A100, start=0x000C, end=0x0015
←RX	`@E,0040,DL,H=0016,R=001C,T=2800,P=00, U=00A20C2000089A9EE21115A133333365`	Service 0x6533…A200, start=0x0016, end=0x001C
←RX	`@E,0040,DL,H=FF00,R=FF07,T=2800,P=00, U=1F38A138AD823586A043135C471E5DAE`	WICED OTA Service 0xAE5D…381F, start=0xFF00, end=0xFF07

3.6.2.2 Discovering Local GATT Characteristics

Use the gatts_discover_characteristics (/DLC, ID=5/7) API command to obtain a list of characteristics in the local GATT database.

Example 1: Local GATT characteristic discovery with factory default structure (no custom attributes)

Direction	Content	Effect
TX→	`/DLC`	Request to discover all local characteristics
←RX	`@R,0011,/DLC,0000,C=000C`	Response indicates success, 12 records to follow
←RX	`@E,0024,DL,H=0002,R=0003,T=2803,P=02,U=002A`	Char 0x2A00, handle=0x0002, value handle=0x0003, perm=0x02
←RX	`@E,0024,DL,H=0004,R=0005,T=2803,P=02,U=012A`	Char 0x2A01, handle=0x0004, value handle=0x0005, perm=0x02
←RX	`@E,0024,DL,H=0006,R=0007,T=2803,P=02,U=042A`	Char 0x2A04, handle=0x0006, value handle=0x0007, perm=0x02
←RX	`@E,0024,DL,H=0009,R=000A,T=2803,P=22,U=052A`	Char 0x2A05, handle=0x0009, value handle=0x000A, perm=0x22
←RX	`@E,0040,DL,H=000D,R=000E,T=2803,P=28, U=01A10C2000089A9EE21115A133333365`	Char 0x6533…A101, handle=0x000D, value handle=0x000E, perm=0x28
←RX	`@E,0040,DL,H=0010,R=0011,T=2803,P=14, U=02A10C2000089A9EE21115A133333365`	Char 0x6533…A102, handle=0x0010, value handle=0x0011, perm=0x14
←RX	`@E,0040,DL,H=0013,R=0014,T=2803,P=20, U=03A10C2000089A9EE21115A133333365`	Char 0x6533…A103, handle=0x0013, value handle=0x0014, perm=0x20
←RX	`@E,0040,DL,H=0017,R=0018,T=2803,P=28, U=01A20C2000089A9EE21115A133333365`	Char 0x6533…A201, handle=0x0017, value handle=0x0018, perm=0x0A
←RX	`@E,0040,DL,H=001A,R=001B,T=2803,P=28, U=02A20C2000089A9EE21115A133333365`	Char 0x6533…A202, handle=0x001A, value handle=0x001B, perm=0x28
←RX	`@E,0040,DL,H=FF01,R=FF02,T=2803,P=38, U=1B666C080A578E83994EA7F7BF50DDA3`	char 0xA3DD…661B, handle=0xFF01, value handle=0xFF02
←RX	`@E,0040,DL,H=FF04,R=FF05,T=2803,P=08, U=26FE2EE709244FB7914061D97A6CE8A2`	char 0xA2E8…FE26, handle=0xFF04, value handle=0xFF05
←RX	`@E,0040,DL,H=FF06,R=FF07,T=2803,P=02, U=4BDEC4EDD4753B91EB472D2E08767FA4`	char 0xA47F…DE4B, handle=0xFF06, value handle=0xFF07

3.6.2.3 *Discovering Local GATT Descriptors*

Use the gatts_discover_descriptors (/DLD, ID=5/8) API command to obtain a list of descriptors in the local GATT database.

Example 1: Local GATT descriptor discovery with factory default structure (no custom attributes)

Direction	Content	Effect
TX→	`/DLD`	Request to discover all local descriptors
←RX	`@R,0011,/DLD,0000,C=0007`	Response indicates success, 28 records to follow
←RX	`@E,0024,DL,H=000B,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x000B, perm=0x0A
←RX	`@E,0024,DL,H=000F,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x000F, perm=0x0A
←RX	`@E,0024,DL,H=0012,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x0012, perm=0x0A
←RX	`@E,0024,DL,H=0015,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x0015, perm=0x0A
←RX	`@E,0024,DL,H=0019,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x0019, perm=0x0A
←RX	`@E,0024,DL,H=001C,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x001C, perm=0x0A
←RX	`@E,0024,DL,H=FF03,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0xFF03, perm=0x0A

3.6.3 Reading and Writing Local GATT Attribute Values

Read and write local GATT values using the gatts_read_handle (/RLH, ID=5/9) and gatts_write_handle (/WLH, ID=5/10) API commands, respectively.

These commands work like their remote client-side counterparts, except that client-level permissions and access restrictions do not apply. It is always possible to locally read any attribute and to locally write any attribute that supports the write operation. Some attributes, such as service and characteristic declarations, contain only constant data (stored in flash) that is not meant to be modified with a typical GATT write command. If you intend to change the structure of the GATT database itself, use the gatts_create_attr (/CAC, ID=5/1) and gatts_delete_attr (/CAD, ID=5/2) API commands.

3.6.3.1 *Reading Local GATT Data*

You can read the value of a local attribute using the gatts_read_handle (/RLH, ID=5/9) API command. EZ-Serial will return the current value in the response.

Note: User-managed attributes have no RAM-backed data storage, so there is never any data to read. Attempting to read this type of characteristic will generate an error result in the response.

Example 1: Read local Device Name characteristic

Direction	Content	Effect
TX→	`/RLH,H=3`	Read attribute with handle = 3
←RX	`@R,0031,/RLH,0000,` `D=455A2D53657269616C2034323A31413A3633`	Response indicates success, hex data is “EZ-Serial 42:1A:63”

For a custom GATT value characteristic, EZ-Serial will allocate and reserve the memory and flash space to store the read and write data. When a remote client writes data to the custom GATT value characteristic, the written data will be stored, and the gatts_data_written (W, ID=5/2) event data will be sent to the local host. When a remote client reads the data from the custom GATT value characteristic, the data stored in the memory will be used, so no event data will be reported to local host, and no requirements for the host to implement this operation. The host can set or update the value of the local custom GATT value characteristic through the gatts_write_handle (/WLH, ID=5/10) API command before any remote client reads from it.

3.6.3.2 *Writing Local GATT Data*

You can write the value of a local RAM-backed attribute using the gatts_write_handle (/WLH, ID=5/10) API command. This command replaces any existing data in the attribute and is limited by the maximum length of the attribute in the GATT structure.

Writing data does not automatically push a notification or indication packet to a remote client, even if the client has subscribed to either of these types of pushed updates. See Notifying and Indicating Data to a Remote Client for details on how to push data.

Example 1: Write "ABCD" at beginning of local Device Name characteristic

Direction	Content	Effect
TX→	`/WLH,H=3,D=41424344`	Write "ABCD" (hex) into attribute with handle = 3
←RX	`@R,000A,/WLH,0000`	Response indicates success
TX→	`/RLH,H=3`	Read attribute with handle = 3 to verify
←RX	`@R,0031,/RLH,0000,D=41424344`	Response indicates success, data shows expected value

3.6.4 Notifying and Indicating Data to a Remote Client

Notifying and indicating allow a server to push updates to a client without the client specifically requesting the latest values. These transfer mechanisms provide an efficient way to send real-time updates without constant polling from the client side, saving power for use cases such as remote sensors or any interrupt-driven activities.

Notifications and indications transmit data from the server to the client, but notifications are unacknowledged, while indications are acknowledged. You can transmit multiple notifications during a single connection interval, but you can only transmit one indication every two connection intervals (one interval for the transmission and one for the acknowledgement).

Although the server decides when to push data to the client using these methods, the client retains ultimate control over whether the server may transmit at all, via the use of "subscription" bits for each type of transfer. All GATT characteristics which support either the "notify" or "indicate" operation must have a "Client Characteristic Configuration Descriptor" (CCCD) within the set of attributes making up the complete characteristic structure. For example, the "Service Changed" characteristic (UUID 0x2A05) within the "Generic Attribute" service (UUID 0x1801) is made up of three separate attributes as listed in Table 3-15.

Handle	UUID	Description
0x0009	0x2803	Characteristic Declaration
0x000A	0x2A05	Service Change Value Attribute
0x000B	0x2902	Client Characteristic Configuration Descriptor (CCCD)

Table 3-15. Service Changed GATT Characteristic Structure

This characteristic supports the "indicate" operation. For a client to subscribe to indications, it must set Bit 1 (0x02) of the value in the CCCD. This descriptor holds a 16-bit value, so the correct operation on the client side is to write `[ 02 00 ]` to handle 0x000B.

For characteristics that support the "notify" operation, the correct subscription flag is Bit 0 (0x01).

Notification and indication subscriptions do not persist across multiple connections.

3.6.4.1 *Notifying Data to a Remote Client*

Use the gatts_notify_handle (/NH, ID=5/11) API command to notify data to a remote client. You must use a handle corresponding to a value attribute for a characteristic for which the remote client has already subscribed to notifications by writing 0x0001 to the relevant CCCD.

Note: Notifying data to a client requires an active connection.

Example 1: Notify a four-byte value to a client manually using the CYSPP Unacknowledged Data characteristic

Direction	Content	Effect
TX→	`/NH,H=11,D=41424344`	Notify "ABCD" (hex) via attribute with handle = 17 (0x11)
←RX	`@R,0009,/NH,0000`	Response indicates success

3.6.4.2 *Indicating Data to a Remote Client*

Use the gatts_indicate_handle (/IH, ID=5/12) API command to indicate data to a remote client. You must use a handle corresponding to a value attribute for a characteristic for which the remote client has already subscribed to indications by writing 0x0002 to the relevant CCCD.

Note: Indicating data to a client requires an active connection.

Example 1: Indicate a start/end handle range to a client through the Service Changed characteristic

Direction	Content	Effect
TX→	`/IH,H=A,D=1D002500`	Write 1D002500 via attribute with handle = 10 (0x0A)
←RX	`@R,0009,/IH,0000`	Response indicates success
←RX	`@E,000F,IC,C=04,H=0009`	Event indicates client has confirmed receipt of data

3.6.5 Detecting and Processing Written Data from a Remote Client

Write operations from a remote GATT client will generate the gatts_data_written (W, ID=5/2) API event, containing the handle and value data as well as the remote connection handle from the device that initiated the request. This event will only occur if the write succeeds and was not blocked due to incorrect permissions, insufficient encryption or authentication levels, or invalid length or offset.

If the `type` parameter of this event has the high bit (0x80) set, you must manually respond to the write operation with the gatts_send_writereq_response (/WRR, ID=5/13) API command. This occurs for user-managed characteristics, or if you have globally disabled automatic write responses using the gatts_get_parameters (GGSP, ID=5/15) API command.

Note: EZ-Serial does not currently implement an API event for read requests.

3.7 GATT Client Examples

EZ-Serial provides GATT client operational support through a variety of API methods. All methods described in this section require an active connection to a remote peer device, and will generate an error result if attempted without a connection.

3.7.1 Discovering a Remote Server's GATT Structure

EZ-Serial's remote GATT discovery methods function the same as the local discovery methods, with the addition of a connection handle in the discovery result output. For an overview of some of the behavioral differences between local and remote GATT discovery, see Listing Local GATT Services, Characteristics, and Descriptors.

Note: Any attribute that requires authentication (bonding) must also require encryption. If you enable the authentication bit, make sure that you also enable the encryption bit, or the command will be rejected with a resultant error.

Note: Remote discovery procedures often complete with a final result code of 0x060A rather than 0x0000. This does not indicate a problem, but only means that the final internal request to find more data in the specified start/end range yielded no further results. This is a logical indicator to the client that it should terminate the discovery process. You can avoid this result code by specifying start and end range values in the discovery request command, which do not result in a final search in an empty range on the server. However, these start and end values are typically not available before performing the discovery in the first place.

3.7.1.1 Discovering Remote GATT Services

Use the gattc_discover_services (/DRS, ID=6/1) API command to obtain a list of services in the remote GATT database on a connected peer device.

Example 1: Remote GATT service discovery on an EZ-Serial peer device with factory default configuration

Direction	Content	Effect
TX→	`/DRS,C=2`	Request to discover all remote services
←RX	`@R,000A,/DRS,0000`	Response indicates success
←RX	`@E,0029,DR,C=02,H=0001,R=0007,T=2800,P=00,U=0018`	Service 0x1800, start=0x0001, end=0x0007
←RX	`@E,0029,DR,C=02,H=0008,R=000B,T=2800,P=00,U=0118`	Service 0x1801, start=0x0008, end=0x000B)
←RX	`@E,0045,DR,C=02,H=000C,R=0015,T=2800,P=00,U=00A10C2000089A9EE21115A133333365`	Service 0x6533…A100, start=0x000C, end=0x0015
←RX	`@E,0045,DR,C=02,H=0016,R=001C,T=2800,P=00,U=00A20C2000089A9EE21115A133333365`	Service 0x6533…A200, start=0x0016, end=0x001C
←RX	`@E,0043,DR,C=02,H=FF00,R=FF07,T=01,P=00,U=1F38A138AD823586A043135C471E5DAE`	Service 0xAE5D…381F, start=0xFF00, end=0xFF07
←RX	`@E,0010,RPC,C=02,R=0000,T=01`	Remote procedure complete

3.7.1.2 Discovering Remote GATT Characteristics

Use the gattc_discover_characteristics (/DRC, ID=6/2) API command to obtain a list of characteristics in the remote GATT database on a connected peer device.

Example 1: Remote GATT characteristic discovery on an EZ-Serial peer device with factory default configuration

Direction	Content	Effect
TX→	`/DRC,C=2`	Request to discover all remote characteristics
←RX	`@R,000A,/DRC,0000`	Response indicates success
←RX	`@E,0029,DR,C=02,H=0002,R=0003,T=2803,P=02,U=002A`	Char 0x2A00, handle=0x0002, value handle=0x0003, perm=0x02
←RX	`@E,0029,DR,C=04,H=0004,R=0005,T=2803,P=02,U=012A`	Char 0x2A01, handle=0x0004, value handle=0x0005, perm=0x02
←RX	`@E,0029,DR,C=04,H=0006,R=0007,T=2803,P=02,U=042A`	Char 0x2A04, handle=0x0006, value handle=0x0007, perm=0x02
←RX	`@E,0029,DR,C=04,H=0009,R=000A,T=2803,P=22,U=052A`	Char 0x2A05, handle=0x0009, value handle=0x000A, perm=0x22
←RX	`@E,0045,DR,C=04,H=000D,R=000E,T=2803,P=28,U=01A10C2000089A9EE21115A133333365`	Char 0x6533…A101, handle=0x000D, value handle=0x00E, perm=0x28
←RX	`@E,0045,DR,C=04,H=0010,R=0011,T=2803,P=14,U=02A10C2000089A9EE21115A133333365`	Char 0x6533…A102, handle=0x0010, value handle=0x0011, perm=0x14
←RX	`@E,0045,DR,C=04,H=0013,R=0014,T=2803,P=20,U=03A10C2000089A9EE21115A133333365`	Char 0x6533…A103, handle=0x0013, value handle=0x0014, perm=0x20
←RX	`@E,0045,DR,C=04,H=0017,R=0018,T=2803,P=28,U=01A20C2000089A9EE21115A133333365`	Char 0x6533…A201, handle=0x0017, value handle=0x0018, perm=0x28
←RX	`@E,0045,DR,C=04,H=001A,R=001B,T=2803,P=28,U=02A20C2000089A9EE21115A133333365`	Char 0x6533…A202, handle=0x001A, value handle=0x001B, perm=0x28
←RX	`@E,0043,DR,C=02,H=FF01,R=FF02,T=04,P=38,U=1B666C080A578E83994EA7F7BF50DDA3`	Char 0xA3DD…661B, handle=0xFF01, value handle=0xFF02, perm=0x28
←RX	`@E,0043,DR,C=02,H=FF04,R=FF05,T=04,P=08,U=26FE2EE709244FB7914061D97A6CE8A2`	Char 0xA2E8…FE26, handle=0xFF04, value handle=0xFF05, perm=0x28
←RX	`@E,0043,DR,C=02,H=FF06,R=FF07,T=04,P=02,U=4BDEC4EDD4753B91EB472D2E08767FA4`	Char 0xA47F…DE4B, handle=0x00FF06, value handle=0xFF07, perm=0x28

Direction	Content	Effect
←RX	`@E,0015,RPC,C=02,R=0000,T=04`	Remote procedure of discovering characteristic completed

3.7.1.3 Discovering Remote GATT Descriptors

Use the gattc_discover_descriptors (/DRD, ID=6/3) API command to obtain a list of descriptors in the remote GATT database on a connected peer device.

Example 1: Remote GATT descriptor discovery on an EZ-Serial peer device with factory default configuration

Direction	Content	Effect
TX→	`/DRD,C=2`	Request to discover all remote descriptors
←RX	`@R,000A,/DRD,0000`	Response indicates success
←RX	`@E,0024,DL,H=000B,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x000B, perm=0x0A
←RX	`@E,0024,DL,H=000F,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x000F, perm=0x0A
←RX	`@E,0024,DL,H=0012,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x0012, perm=0x0A
←RX	`@E,0024,DL,H=0015,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x0015, perm=0x0A
←RX	`@E,0024,DL,H=0019,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x0019, perm=0x0A
←RX	`@E,0024,DL,H=001C,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0x001C, perm=0x0A
←RX	`@E,0024,DL,H=FF03,R=0000,T=2902,P=0A,U=0229`	UUID 0x2902 (CCCD), handle=0xFF03, perm=0x0A
←RX	`@E,0015,RPC,C=02,R=0000,T=05`	Remote procedure of discovering descriptors completed

3.7.2 Reading and Writing Remote GATT Attribute Values

Reading and writing local GATT values may be accomplished with the gattc_read_handle (/RRH, ID=6/4) and gattc_write_handle (/WRH, ID=6/5) API commands, respectively.

3.7.3 Detecting Notified or Indicated Values from a Remote GATT Server

A remote GATT server may push data updates to a client at unpredictable times, if the client has subscribed to notifications or indication on a supported remote GATT server characteristic. When this occurs, EZ-Serial generates the gattc_data_received (D, ID=6/3) API event with the connection handle, attribute handle, and value data.

To receive notifications or indications from a remote server, you must first subscribe to the relevant type of data updates by writing a special value to the attribute called the CCCD. This attribute always has a UUID of 0x2902, and is a separate attribute relative to the characteristic declaration (UUID 0x2803) or characteristic value (custom UUID).

Usually, the CCCD attribute has a handle value that is +1 or +2 from the characteristic value attribute. You can use the gattc_discover_descriptors (/DRD, ID=6/3) API command to obtain a list of descriptors and identify which attributes you need to use. For example, a remote server structure might contain the following:

- Handle 0x0017, UUID 0x2803: Characteristic Declaration Descriptor
- Handle 0x0018, UUID 0x2A46: Characteristic Value Descriptor (“New Alert” characteristic)
- Handle 0x0019, UUID 0x2902: Client Characteristic Configuration Descriptor

With this structure, you can subscribe to notifications for this characteristic by writing the 16-bit value 0x0001 to the attribute with handle 0x0019. Remember, you must write this value as a little-endian integer `[ 01 00 ]`. To unsubscribe from receiving notifications, simply write the value 0x0000 to the same CCCD attribute.

Subscribing to indications requires the same procedure, but you must use the value 0x0002 instead of 0x0001.

The CCCD attribute with UUID 0x2902 will only be present for characteristic which support either notifications or indications. Whether you should enable notifications or indications depends on which of those two GATT methods is implemented on the server side. For official, adopted characteristics, you can find this information on the Bluetooth SIG developer website. For proprietary or custom characteristics, see the documentation or reference material available from the product developer.

3.8 Security and Encryption Examples

EZ-Serial supports built-in Bluetooth security technologies for safeguarding sensitive data transmitted wirelessly, including privacy and encryption.

3.8.1 Using Peripheral and Central Privacy

GAP privacy randomizes the Bluetooth connection address visible to remote devices while in certain operating modes. Use the smp_set_privacy_mode (SPRV, ID=7/9) API command to enable or disable peripheral or central privacy. Enabling privacy in each mode causes the Bluetooth connection address used in related states to be random (private) instead of fixed (public). This can make passive profiling by a remote observer more difficult.

Peripheral privacy affects the Bluetooth connection address broadcast during advertisements, which the remote central device may log or use for a scan request or connection request. Central privacy affects the Bluetooth connection address used for scan requests or connection requests when scanning for or communicating with a remote device.

Once enabled, EZ-Serial will randomize the private address on the interval configured by the smp_set_privacy_mode (SPRV, ID=7/9) API command.

Example 1: Enable peripheral and central privacy

Direction	Content	Effect
TX→	`SPRV$,M=3`	Enable central and peripheral privacy, store in flash
←RX	`@R,000B,SPRV$,0000`	Response indicates success

3.8.2 Bonding with or without MITM Protection

Bonding between two devices requires generating and exchanging encryption keys, and then permanently storing encryption data along with information required to identify the bonded device and reuse the same keys. The mechanics of pairing depend on the side (master or slave) that initiates the pairing request, and the I/O capabilities of each side.

Note: While the Bluetooth specification allows pairing (generation and exchange of encryption keys) without bonding (permanent storage of encryption data), most common smartphones, tablets, and computer operating systems require performing both at the same time if you need encryption. The encryption-only arrangement (no bonding) is supported only between modules that support pairing without bonding.

The Bluetooth specification provides a random passkey generation/display/comparison mechanism for preventing man-in-the-middle (MITM) attacks during the pairing process. EZ-Serial supports pairing with or without MITM protection enabled. The factory default settings apply the "just works" method, with no passkey entry and no MITM protection. You can set local I/O capabilities with the `io` argument of the smp_get_br_edr_security_parameters (GSBSP, ID=7/19) and smp_get_le_security_parameters (GSLSP, ID=7/21) API commands.

3.8.2.1 Understanding I/O Capabilities

The I/O capabilities of each peer involved in a pairing process affects the resulting security type (authenticated versus unauthenticated) and the exact nature of which events and commands must be used on each side. Table 3-16 describes all possible I/O arrangements and the resulting behavior and authentication level.

	INITIATOR				
RESPONDER	**DisplayOnly**	**Display+YesNo**	**KeyboardOnly**	**NoInput+NoOutput**	**Keyboard+Display**
DisplayOnly	Just Works (Unauthenticated)	Just Works (Unauthenticated)	Passkey Entry: Responder displays Initiator inputs (Authenticated)	Just Works (Unauthenticated)	Passkey Entry: Responder displays Initiator inputs (Authenticated)
Display+YesNo	Just Works (Unauthenticated)	Just Works (Unauthenticated)	Passkey Entry: Responder displays Initiator inputs (Authenticated)	Just Works (Unauthenticated)	Passkey Entry: Responder displays Initiator inputs (Authenticated)
KeyboardOnly	Passkey Entry: Initiator displays Responder inputs (Authenticated)	Passkey Entry: Initiator displays Responder inputs (Authenticated)	Passkey Entry: Initiator inputs Responder inputs (Authenticated)	Just Works (Unauthenticated)	Passkey Entry: Initiator displays Responder inputs (Authenticated)
NoInput+NoOutput	Just Works (Unauthenticated)	Just Works (Unauthenticated)	Just Works (Unauthenticated)	Just Works (Unauthenticated)	Just Works (Unauthenticated)
Keyboard+Display	Passkey Entry: Initiator displays Responder inputs (Authenticated)	Passkey Entry: Initiator displays Responder inputs (Authenticated)	Passkey Entry: Responder displays Initiator inputs (Authenticated)	Just Works (Unauthenticated)	Passkey Entry: Initiator displays Responder inputs (Authenticated)

Table 3-16. I/O Capabilities and Pairing Behavior

The information in Table 3-16 comes from the Bluetooth Core Specification. Combinations reporting “unauthenticated” do not support MITM protection mechanisms.

Note: Smartphones, tablets, and computers all support full Keyboard+Display I/O capabilities. Also, when a smartphone has connected as a central device (the one opening the connection), typically the smartphone OS will not allow the peripheral to act as the pairing initiator. The peripheral can request pairing using the smp_pair (/P, ID=7/3) API command, but the smartphone will reject the request and immediately initiate its own request instead, after first confirming with an on-screen prompt whether to proceed with pairing. When this happens, you will see the smp_pairing_requested (P, ID=7/2) API event immediately following your local pairing request command. The EZ-Serial peripheral device will then be operating in the “responder” role described in Table 3-16.

Table 3-17 describes the local API command and event flow you should expect when using EZ-Serial with some common configurations and remote peer devices. All API sequences shown here assume that the “auto-accept incoming pairing” flag bit is set. If it is not set, you must manually accept incoming requests with the smp_send_pairreq_response (/PR, ID=7/5) API command anytime the smp_pairing_requested (P, ID=7/2) event occurs. For more information, see Controlling Automatic Pairing Request Acceptance.

.

		Remote Peer	
Local I/O	**Role**	**Smartphone**	**EZ-Serial with Keyboard + Display (I=4)**
DisplayOnly (I=0)	Initiator	N/A, smartphone takes initiator role	TX: smp_pair (/P, ID=7/3) RX: smp_passkey_display_requested (PKD, ID=7/5) *[enter passkey in remote EZ-Serial to finish]*
	Responder	RX: smp_pairing_requested (P, ID=7/2) RX: smp_passkey_display_requested (PKD, ID=7/5) *[enter passkey on smartphone to finish]*	RX: smp_pairing_requested (P, ID=7/2) RX: smp_passkey_display_requested (PKD, ID=7/5) *[enter passkey in remote EZ-Serial to finish]*
Display+YesNo (I=1)	Initiator	*N/A, smartphone takes initiator role*	TX: smp_pair (/P, ID=7/3) RX: smp_passkey_display_requested (PKD, ID=7/5) *[enter passkey in remote EZ-Serial to finish]*
	Responder	RX: smp_pairing_requested (P, ID=7/2) RX: smp_passkey_display_requested (PKD, ID=7/5) *[enter passkey on smartphone to finish]*	RX: smp_pairing_requested (P, ID=7/2) RX: smp_passkey_display_requested (PKD, ID=7/5) *[enter passkey in remote EZ-Serial to finish]*
KeyboardOnly (I=2)	Initiator	*N/A, smartphone takes initiator role*	TX: smp_pair (/P, ID=7/3) *[passkey displayed in remote EZ-Serial]* RX: smp_passkey_entry_requested (PKE, ID=7/6)
	Responder	RX: smp_pairing_requested (P, ID=7/2) *[passkey displayed on smartphone]* RX: smp_passkey_entry_requested (PKE, ID=7/6) TX: smp_send_passkeyreq_response (/PE, ID=7/6)	RX: smp_pairing_requested (P, ID=7/2) *[passkey displayed in remote EZ-Serial]* RX: smp_passkey_entry_requested (PKE, ID=7/6) TX: smp_send_passkeyreq_response (/PE, ID=7/6)
NoInput+NoOutput (I=3)	Initiator	*N/A, smartphone takes initiator role*	TX: smp_pair (/P, ID=7/3) *[process completes without interaction]*
	Responder	RX: smp_pairing_requested (P, ID=7/2) *[process completes without interaction]*	RX: smp_pairing_requested (P, ID=7/2) *[process completes without interaction]*
Keyboard+Display (I=4)	Initiator	*N/A, smartphone takes initiator role*	TX: smp_pair (/P, ID=7/3) RX: smp_passkey_display_requested (PKD, ID=7/5) *[enter passkey in remote EZ-Serial to finish]*
	Responder	RX: smp_pairing_requested (P, ID=7/2) *[passkey displayed on smartphone]* RX: smp_passkey_entry_requested (PKE, ID=7/6) TX: smp_send_passkeyreq_response (/PE, ID=7/6)	RX: smp_pairing_requested (P, ID=7/2) *[passkey displayed in remote EZ-Serial]* RX: smp_passkey_entry_requested (PKE, ID=7/6) TX: smp_send_passkeyreq_response (/PE, ID=7/6)

Table 3-17. EZ-Serial API Flow in Common I/O Capability Configurations

3.8.2.2 Controlling Automatic Pairing Request Acceptance

EZ-Serial's default behavior is to accept all compatible pairing requests that come in from other devices. However, your application may benefit from having more control over the pairing process. To change this, clear Bit 0 (0x01) of the **`flags`** value in the smp_set_parameters (SSMPP, ID=7/32) API command. Subsequent pairing requests will generate the smp_pairing_requested (P, ID=7/2) API event, and you must respond with the smp_send_pairreq_response (/PR, ID=7/5) API command to accept or reject the request.

Example 1 assumes that you have already connected to a remote peer device. An active connection is required for any type of pairing operation to succeed. However, configuration of security settings may be done either before or after connecting.

Example 1: Disable automatic acceptance of incoming pairing requests, store in flash, then pair from remote peer

Direction	Content	Effect
TX→	`/P`	Clear Bit 0 (auto-accept)
←RX	`@R,000B,SSMPP$,0000`	Response indicates success, stored in flash
←RX	`@E,001B,P,C=04,M=01,B=01,K=10,P=00`	Event indicates incoming pairing request
TX→	`/PR,C=04,R=0`	Send pairing request response with "0" result (accept)
←RX	`@R,0009,/PR,0000`	Response indicates success
←RX	`@E,000E,ENC,C=04,S=01`	Event indicates encryption status has changed
←RX	`@E,001B,B,B=04,A=00A050E3835F,T=00`	Event indicates new bond entry has been created

Direction	Content	Effect
←RX	`@E,000F,PR,C=04,R=0000`	Event indicates pairing process has completed successfully

3.8.2.3 Pairing and Bonding in "Just Works" Mode Without MITM Protection

The simplest way to bond requires no special passkey entry or display. If your device has no input or output capabilities, you must use this mode for pairing since MITM protection requires numeric display or entry (or both) to function correctly.

Example 1 assumes that you have already connected to a remote peer device. An active connection is required for any type of pairing operation to succeed. However, configuration of security settings may be done either before or after connecting.

Example 1: Configure simple pairing without MITM protection, then initiate pairing

Direction	Content	Effect
TX→	`SSBP,M=0,I=3`	Set "No Input / No Output" I/O, no MITM protection
←RX	`@R,000A,SSPB,0000`	Response indicates success
TX→	`/P`	Initiate pairing request to remote peer
←RX	`@R,0008,/P,0000`	Response indicates success
←RX	`@E,000E,ENC,C=04,S=01`	Event indicates encryption status has changed (peer accepted)
←RX	`@E,001B,B,B=04,A=00A050421C63,T=00`	Event indicates new bond entry has been created
←RX	`@E,000F,PR,C=04,R=0000`	Event indicates pairing process has completed successfully

3.8.2.4 Pairing and Bonding with Full I/O Capabilities and MITM Protection

If your design includes a numeric display or keypad (or both), you can enable MITM protection for improved security during pairing. In this configuration, you must either display a passkey to the user or allow the user to enter a passkey, depending on the exact I/O capabilities and the side which initiates pairing and side which responds. See Understanding I/O Capabilities for details.

Note: All API events relating to passkey entry or display use hexadecimal formatting. However, user entry and display must use decimal format, including any necessary leading zeros for a full 6-digit value. Ensure that your application uses decimal format for any user interactions involving the passkey.

Example 1 assumes that you have already connected to a remote peer device. An active connection is required for any type of pairing operation to succeed. However, configuration of security settings may be done either before or after connecting.

Example 1: Configure keyboard+display I/O capabilities and MITM protection, then initiate pairing

Direction	Content	Effect
TX→	`SSBP,M=12,I=4`	Set "Keyboard+Display" I/O, enable MITM protection
←RX	`@R,000A,SSPB,0000`	Response indicates success
TX→	`/P`	Initiate pairing request to remote peer
←RX	`@R,0008,/P,0000`	Response indicates success
←RX	`@E,001B,P,C=04,M=02,B=01,K=10,P=00`	Event indicates incoming pairing request
←RX	`@E,0014,PKD,C=04,P=00017266`	Event indicates passkey display (17266 hex = **094822** dec)
←RX	`@E,000E,ENC,C=04,S=01`	Event indicates encryption status has changed (peer entered key)
←RX	`@E,001B,B,B=04,A=00A050421C63,T=00`	Event indicates new bond entry has been created
←RX	`@E,000F,PR,C=04,R=0000`	Event indicates pairing process has completed successfully

3.8.2.5 *Pairing and Bonding with a Fixed Passkey*

If your application requires it, EZ-Serial supports the configuration of a fixed passkey to be used during the pairing process instead of either no passkey or a random one. You can choose a fixed 6-digit value between 000000 and 999999 using the smp_set_fixed_passkey (SFPK, ID=7/13) API command and configuring the local I/O capabilities to the “Display Only” value with the smp_set_br_edr_security_parameters (SSBSP, ID=7/18) and smp_set_le_security_parameters (SSLSP, ID=7/20) API commands. During pairing, EZ-Serial will generate the smp_passkey_display_requested (PKD, ID=7/5) API event containing the value configured here. The remote peer must then enter this key to pair successfully.

Note: The fixed passkey will take effect only if you enable fixed passkey use by setting Bit 1 (0x02) of the security flags parameter and set the “Display Only” I/O capabilities value (0x00) using the smp_set_br_edr_security_parameters (SSBSP, ID=7/18) and smp_set_le_security_parameters (SSLSP, ID=7/20) API commands. If both conditions are not met, then the stack will revert to the default behavior of using a random passkey.

Example 1 assumes that you have already connected to a remote peer device. An active connection is required for any type of pairing operation to succeed. However, configuration of security settings may be done either before or after connecting.

Example 1: Configure “123456” fixed passkey value and required I/O capabilities, then pair from remote peer

Direction	Content	Effect
TX→	`SSBP,M=12,I=0,F=3`	Set “Display Only” I/O, enable fixed passkey use flag bit (0x02)
←RX	`@R,000A,SSPB,0000`	Response indicates success
TX→	`SFPK,P=1E240`	Set fixed passkey value (1E240 hex = **123456** dec)
←RX	`@R,000A,SFPK,0000`	Response indicates success
←RX	`@E,001B,P,C=04,M=13,B=01,K=10,P=01`	Event indicates incoming pairing request
←RX	`@E,0014,PKD,C=04,P=0001E240`	Event indicates passkey display (1E240 hex = **123456** dec)
←RX	`@E,000E,ENC,C=04,S=01`	Event indicates encryption status has changed (peer entered key)
←RX	`@E,001B,B,B=04,A=76C880C3F154,T=01`	Event indicates new bond entry has been created
←RX	`@E,000F,PR,C=04,R=0000`	Event indicates pairing process has completed successfully

3.9 Beacon Examples

EZ-Serial provides simple configuration commands for beacon broadcast management. Most BLE-based beaconing technologies require only a specially formed advertisement packet, but implementing this manually requires additional tracking and modifying of advertising behavior and does not allow scheduled interleaving with other types of behavior simultaneously.

3.9.1 Configuring iBeacon Transmissions

Use the p_ibeacon_set_parameters (.IBSP, ID=12/1) API command to configure automated iBeacon broadcast packets based on a supplied UUID and major/minor ID set.

Note: The UUID supplied in the configuration command will be added to the advertisement packet exactly as entered, with the same byte order. In contrast, the major and minor values are interpreted as fixed-length 16-bit integers and subject to the typical rules for text and binary mode byte ordering.

Official iBeacon specifications are available from the iBeacon page on Apple’s developer website.

Example 1: Enable auto-start iBeacon broadcasting with sample IDs at 100 ms interval, store in flash

Direction	Content	Effect
TX→	`.IBSP$,E=02,I=00A0,U=00112233445566778899AABBCCDDEEFF,A=1111,N=2222`	Set iBeacon configuration
←RX	`@R,000C,.IBSP$,0000`	Response indicates success

3.9.2 Configuring Eddystone Transmissions

Use the p_eddystone_set_parameters (.EDDYSP, ID=13/1) API command to configure automated Eddystone broadcast packets based on a supplied configuration set. EZ-Serial currently supports Eddystone-UID and Eddystone-URL frames, but does not support Eddystone-TLM frames (beacon telemetry data).

Official Eddystone beacon specifications are available from Google's "Eddystone" repository on GitHub.

Example 1: Enable auto-start Eddystone broadcasting of "http://www.cypress.com/" URL at 100 ms interval

Direction	Content	Effect
TX→	`.EDDYSP,E=02,I=00A0,T=10,D=006379707265737307`	Set Eddystone configuration with scheme and encoding
←RX	`@R,000D,.EDDYSP,0000`	Response indicates success

3.10 Performance Testing Examples

This section covers the techniques to achieve optimal performance in specific contexts.

3.10.1 Maximizing Throughput to a Remote Peer

Throughput concerns the amount of data you can move across a link within a specific period, usually expressed in bytes per second or bits per second (8 bits per byte). In case of BLE, the following guidelines will help improve average throughput:

- **Minimize the connection interval.** The BLE specification allows a minimum of 7.5 ms connection interval. Data transfers are specifically timed during BLE connections, and more frequent transfers mean higher potential throughput.
  - When operating in the GAP central role, you can determine the connection interval when initiating the connection with the gap_connect (/C, ID=4/1) API command, or afterwards with a connection update request using the gap_update_conn_parameters (/UCP, ID=4/3) API command.
  - When operating in the GAP peripheral role, the remote central determines the initial interval, and you must request an update with the gap_update_conn_parameters (/UCP, ID=4/3) API command after connecting. The remote peer (master/central device) may either accept or reject this request. Note that if the remote peer rejects the request, it will not notify the requesting device; the only evidence of the rejection will be the lack of a subsequent gap_connection_updated (CU, ID=4/8) API event.
- **Maximize the payload size for GATT transfers.** It takes longer to send 20 one-byte packets than one 20-byte packet, due to the low transmission duty cycle required by the BLE protocol. If your application has five 16-bit sensor measurement values that are used by the remote peer at the same interval, use a single characteristic to provide all 10 bytes at once rather than using five separate characteristics.
- **Use unacknowledged transfers.** You can push more unacknowledged data through a single connection interval than you can with acknowledged transfers. A typical acknowledged data transfer requires two full connection intervals to complete (one for the transfer and one for the acknowledgement), but multiple unacknowledged transfers can be used in a sequence within the same interval—up to one packet every 1.25 ms, if supported by the remote client. Typically, standalone full-stack modules cannot buffer and process data quite this fast, but it is often possible to achieve something near this level of throughput. Note that making this change may require additional application logic to provide a packet delivery/retry request mechanism.
  - For client-to-server transfers, use the "write-no-response" operation instead of "write."
  - For server-to-client transfers, use the "notify" operation instead of "indicate."

These actions will help in increasing the observed throughput, but will simultaneously increase power consumption. Keep this trade-off in mind to choose the right balance between power consumption and throughput.

Example 1: Request a connection parameter update to 7.5 ms interval, no latency, 1 sec timeout

Direction	Content	Effect
TX→	`/UCP,C=2,I=6,L=0,O=64`	Request connection update to 7.5 ms (6 * 1.25 ms), no slave latency, 1-second supervision timeout
←RX	`@R,000A,/UCP,0000`	Response indicates success, request sent to remote peer
←RX	`@E,001D,CU,C=02,H=04,I=0006,L=0000,O=0064`	Event indicates new connection parameters were accepted

3.10.1.1 Maximizing Throughput to an iOS Device

Apple devices began supporting BLE technology with the iPhone 4S and iOS 5. iOS devices have additional limitations on top of those mandated in the Bluetooth specification.

The following additional guidelines apply for maximizing iOS throughput:

- When operating in the GAP central role, the latest iOS devices limit the minimum connection interval of 30 ms (or 11.25 ms when connecting to HID devices). If the peripheral requests a shorter connection interval than this, the iOS device will reject the request.
- iOS devices limit unacknowledged GATT data transfers (write-no-response or notify) to a maximum of four per connection interval, according to widespread observations.
- iOS 5 added support for GAP peripheral role operation, which includes support for 7.5 ms intervals as required by the Bluetooth specification. However, switching GAP roles may not be suitable depending on other application requirements, and requires a notably different mobile app development approach with its own side effects.

See the Core Bluetooth Programming Guide on the Apple Developer website for official guidelines.

Example 1: Request a connection parameter update to 30 ms interval, no latency, 1 sec timeout

Direction	Content	Effect
TX→	`/UCP,C=2,I=18,L=0,O=64`	Request connection update to 30 ms (24 * 1.25 ms), no slave latency, 1-second supervision timeout
←RX	`@R,000A,/UCP,0000`	Response indicates success, request sent to remote peer
←RX	`@E,001D,CU,C=02,H=04,I=0010,L=0000,O=0064`	Event indicates new connection parameters accepted

3.10.1.2 Maximizing Throughput to an Android Device

Android devices officially began supporting BLE technology with the 4.3 release, though 4.4 and onward greatly improved stability and supported functionality.

The following additional guidelines apply for maximizing Android throughput:

- Through 4.4.2, Android supported only a single connection interval of 48.75 ms.
- Version 4.4.3 and later support intervals down to 7.5 ms when requested by the remote device, though the default interval is still 48.75 ms when first establishing the connection.
- Newer android handsets allow up to six unacknowledged GATT transfers in a single connection interval.

3.10.2 Minimizing Power Consumption

You can reduce power consumption by making the BLE radio active as infrequently as your application allows. The specific actions described in this section will not only help in decreasing average consumption, but will also decrease potential throughput. Keep this trade-off in mind to choose the right balance between power consumption and throughput.

If you have not already done so, ensure that the best possible CPU Sleep mode for your application is configured as described in Managing Sleep States. This will ensure that the CPU is not taking more power than necessary. If the CPU is fully or partially awake more often than necessary, the relative improvements possible using the methods described in this section may not make a notable difference.

3.10.2.1 Minimizing Power Consumption While Broadcasting

To reduce power consumption in an advertising state:

- **Maximize the advertisement interval while broadcasting.** The BLE specification allows advertising at any interval between 20 ms and 10240 ms. Increasing the interval means fewer transmissions within a given period. For example, a device advertising at 500 ms will use roughly 20% of the power required by that same device advertising at 100 ms. Use the gap_set_adv_parameters (SAP, ID=4/23) API command to change the default advertisement interval, or the gap_start_adv (/A, ID=4/8) API command to use a non-default interval at the moment you enter an advertising state.

  Side effects:

  - ☐ Scanning devices are less likely to detect each advertisement packet, due to the reduced probability of the scanning device actively receiving on the same channel at the same time the advertisement transmission occurs.
  - ☐ Connections may take longer to establish, since this process begins with the same scanning process and requires detection of a connectable advertisement packet from the target device.

- **Do not use all three advertisement channels.** The BLE spectrum dedicates three channels to advertisement packets, spread across the 2.4 GHz Bluetooth RF spectrum to help ensure reception in busy RF environments. Most BLE devices advertise on all three channels, but you can selectively advertise on only one or two of these channels using the gap_set_adv_parameters (SAP, ID=4/23) or gap_start_adv (/A, ID=4/8) API commands. Advertising on only one channel requires roughly 33% of the power needed when using all three.

  Side effects:

  - ☐ Scanning devices are less likely to detect advertisement packets for the same reason as above—there are fewer advertisement packets being transmitted, which reduces the probability of actively receiving on the correct channel at the correct time.
  - ☐ The advertising device cannot combat RF interference as effectively. If you enable only one advertisement channel, but that portion of the RF spectrum is extremely congested, then a scanning device may not be able to detect advertisement packets at all even if the timing lines up correctly.

- **If connections are not required, use a non-connectable/non-scannable mode.** When a peripheral device is connectable (accepting new connections) or scannable (accepting scan request packets while advertising), the BLE radio switches to a receiving state for approximately 150 us after every advertisement packet to listen for a connection request or scan request packet. When using all three advertising channels, three complete TX-RX cycles occur repeatedly at the configured advertisement interval. If a peripheral device only needs to broadcast (for example, in a beaconing state for iBeacon or Eddystone applications), you can configure a broadcast-only advertising mode with the gap_set_adv_parameters (SAP, ID=4/23) or gap_start_adv (/A, ID=4/8) API command. This prevents the radio from switching into a receiving state after each transmission, saving both time and power.

  Side effects:

  - ☐ Any data configured in the scan response packet payload will never be transmitted. Most often, this is the friendly device name.

- **Minimize the advertisement, scan response data payload length, or both**. Regardless of the configured advertisement interval, the advertisement payload also has a significant effect on the amount of time spent on transmissions. The advertisement payload may be between 0 and 31 bytes, and the BLE RF protocol uses a symbol rate of 1 Mbit/sec, which translates to 8 us per byte. The fixed encapsulation and overhead data in every advertisement or scan response packet takes roughly 140 us to transmit, but the payload can add up to 248 us to this duration. In other words, a 31-byte payload (~390 us) requires twice as much transmission time as a 7-byte payload (~195 us).

  In most cases, the application design requires very specific content in the advertisement payload. However, you should optimize this as much as possible if low power consumption is critical for performance. You can configure custom advertisement data content with the gap_set_adv_data (SAD, ID=4/19) and gap_set_adv_parameters (SAP, ID=4/23) API commands, as described in Customizing Advertisement and Scan Response Data.

3.10.2.2 Minimizing Power Consumption While Connected

To reduce power consumption in a connected state:

- **Maximize the connection interval.** The BLE specification allows a connection interval from 7.5 ms to 4000 ms.
  - When operating in the GAP central role, you can determine the connection interval when initiating the connection, or afterwards with a connection update request.
  - When operating in the GAP peripheral role, the remote central determines the initial interval, and you must request an update after connecting if you need to change it. The remote peer may either accept or reject this request.
- **Use non-zero slave latency.** While this only affects power consumption on the slave/peripheral device during a connection, the slave latency setting can drastically improve power efficiency in many applications. This setting controls how many connection intervals the slave may skip if it has no data to send to the connected master device. Once the allowed number of intervals have occurred, the slave must respond regardless of whether it has any new data to send. The slave may respond at any interval.

  With the default “0” slave latency setting, the slave must acknowledge the master’s connection maintenance packets at every interval. In applications requiring infrequent data transfers, this wastes a great deal of power. Increasing the slave latency value to “3” allows the slave to respond every four intervals instead of every interval, for an average power reduction of 75% while connected. Applications such as environmental sensors and human input devices can benefit greatly from non-zero slave latency.

  The slave latency value may not be higher than the maximum number that allows the calculated value for **`[conn_interval * slave_latency]`** to remain below the **`supervision_timeout`** value, since otherwise the connection would time out regularly.

  Side effects:
  - If the slave has no data to send, the master must wait until the slave latency period passes before it can send or request data to or from the slave. The slave will not be aware of any requests from the master until it enables its radio again. This can result in noticeable delays especially when using long connection intervals. For example, a 500 ms connection interval and slave latency setting of “3” could create a master-to-slave response delay of up to two full seconds. To mitigate this, select a balanced combination of connection interval and slave latency values that provides acceptable master-side delay and slave-side power consumption.
  - Non-zero slave latency interval increases the possibility of a connection timeout in non-optimal RF environments. The master will trigger a supervision timeout condition if it does not receive an acknowledgement from the slave before the timeout period elapses. The master will resend any connection maintenance packet that is not acknowledged, but if the slave has already switched back to a low-power state between required response intervals, the master’s attempted retries may be ignored for too long. To mitigate this, select a longer supervision timeout, shorter connection interval, and/or lower slave latency value to achieve required connection stability in the target environment.
- **Use unacknowledged transfers.** Acknowledged transfers involve more data sent OTA to handle the acknowledgement. This results in higher average consumption. If you do not need application-level data transfer confirmations, use unacknowledged methods instead.
  - For client-to-server transfers, use the “write-no-response” operation instead of “write.”
  - For server-to-client transfers, use the “notify” operation instead of “indicate.”

3.10.3 Communicating Using an L2CAP Channel

Using L2CAP eliminates the overhead and optional upper-layer acknowledgement involved with GATT-based communication. Instead of using structured attributes, L2CAP provides a single data stream for raw transfers.

Note: Most consumer smartphones and tablets available at the time of this publication do not support direct L2CAP connectivity. You must use standard GATT-based APIs to communicate with these devices. The example shown here works between two CYBT-4130XX-02 EZ-BT modules with its dedicated EZ-Serial firmware.

Example 1 assumes that you have already connected both devices together. An active connection is required for any type of L2CAP operations. Registering a PSM only needs to be done once per session; it will persist even after link closure until the module is reset.

Example 1: Open L2CAP connection between two devices and send data

Device	Direction	Content	Effect
#1	TX→	`/LLERPSM,P=88,R=0`	Register PSM on channel 0x0088, role=0
#1	←RX	`@R,0015,/LLERPSM,0000,P=0088`	Response indicates success
#2	TX→	`/LLERPSM,P=88,R=0`	Register PSM on channel 0x0088, role=0
#2	←RX	`@R,0015,/LLERPSM,0000,P=0088`	Response indicates success
#1	TX→	`/LLEC,P=88,A=D36D58D1DA98,T=01,` `M=2,U=200,S=0,K=10`	Open L2CAP connection to #2 peer device, MAC address is D36D58D1DA98, address type is 01. **Note:** Replace the #2 peer device's real MAC address and address type when running it.
#1	←RX	`@R,0012,/LLEC,0000,C=0040`	Response indicates success, and the opened Channel Identifier (CID) is 0x0040
#1	←RX	`@E,0035,C,C=02,A=D36D58D1DA98,T=01,` `I=0027,L=0000,O=02BC,B=00`	Event indicates that a connection has been established
#1	←RX	`@E,000E,ASC,S=00,R=00`	Event indicates that advertisement stopped
#1	←RX	`@E,0022,LLEC,C=0040,A=D36D58D1DA98,M=0080`	Event indicates that LE L2CAP channel has been connected successfully, CID=0x0040
#2	←RX	`@E,0035,C,C=02,A=D9F4FED8A65D,T=01,` `I=0027,L=0000,O=02BC,B=00`	Event indicates that a connection is established
#2	←RX	`@E,000E,ASC,S=06,R=00`	Event indicates that advertisement changes to non-connectable, because a connection has been established
#2	←RX	`@E,0022,LLEC,C=0040,A=D9F4FED8A65D,M=0200`	Event indicates that the LE L2CAP channel is connected, CID=0x0040
#1	TX→	`/LLESD,C=40,D=1122334455667788`	Send 8-byte data packet to peer
#1	←RX	`@R,000C,/LLESD,0000`	Response indicates that the command was sent successfully
#1	←RX	`@E,0015,LLETXC,C=0040,N=0001`	Event indicates that one data packet has completed successfully
#2	←RX	`@E,0020,LLERX,C=0040,D=1122334455667788`	Event indicates that 8-byte data packet was received
#2	TX→	`/LLESD,C=40,D=00998877665544`	Send 7-byte data packet to peer
#2	←RX	`@R,000C,/LLESD,0000`	Response indicates that the command was sent successfully
#2	←RX	`@E,0015,LLETXC,C=0040,N=0001`	Event indicates that one data packet has completed successfully
#1	←RX	`@E,001E,LLERX,C=0040,D=00998877665544`	Event indicates that 7-byte data packet was received
#1	TX→	`/LLEDISC,C=40`	Command to disconnect the L2CAP connection which CID=0x0040
#1	←RX	`@R,000E,/LLEDISC,0000`	Response indicates that the disconnect command was sent successfully
#1	←RX	`@E,0016,LLEDISC,C=0040,R=0000`	Event indicates that the LE L2CAP connection (CID=0x0040) has been disconnected
#2	←RX	`@E,0016,LLEDISC,C=0040,R=0000`	Event indicates that the LE L2CAP channel has been disconnected, CID=0x0040
#2	←RX	`@E,0010,DIS,C=02,R=0913`	Event indicates that a LE connection has been disconnected
#2	←RX	`@E,000E,ASC,S=04,R=06`	Event indicates that advertisement restarted and is advertising in low frequency.

3.11 Device Firmware Update Examples

EZ-Serial provides multiple methods for updating or replacing firmware on the module, and the ability to perform a remote update on a compatible target device using the Cypress WICED OTA GATT profile. See Latest EZ-Serial Firmware Image for information on where to find the latest EZ-Serial firmware images.

3.11.1 Updating Firmware Locally through HCI UART

If you have access to the HCI UART interface, you can use standard WICED Studio SDK tools to flash a new firmware image onto the module. For more details, see Programming an EZ-BT WICED Module – KBA223428.

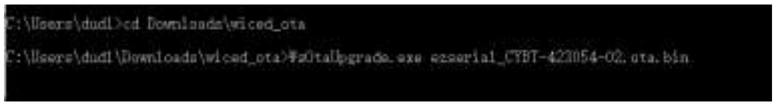
Updating firmware via this method will always return to factory default settings and the remove any bonding data and custom GATT structure.

3.11.2 Updating Firmware through WICED OTA GATT Profile

Power the device ON and set it up. It is assumed that you are using a Windows 10 PC and BLE Controller. Follow these steps to upgrade the firmware image OTA onto the module:

1. Connect the BLE controller to the Windows 10 computer, and make sure all drivers are installed and ready.
2. Find the *WsOtaUpgrade.exe* tool in the installed WICED Studio SDK. Copy it to the directory where your new firmware image exists. Then, run following command line to start upgrading.

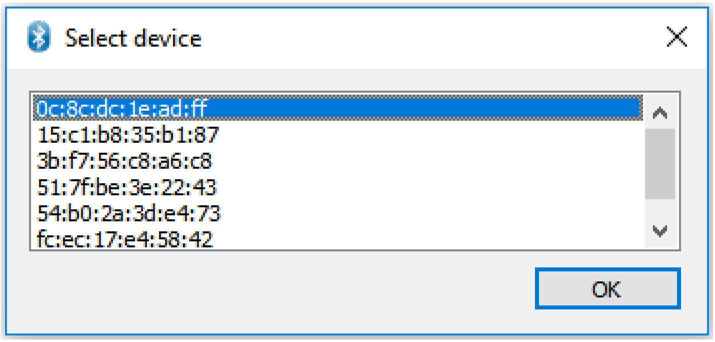
```
WsOtaUpgrade.exe ezserial_CYBT-423054-02.ota.bin
```



```
C:\Users\dudl>cd Downloads\wiced_ota

C:\Users\dudl\Downloads\wiced_ota>WsOtaUpgrade.exe ezserial_CYBT-423054-02.ota.bin
```

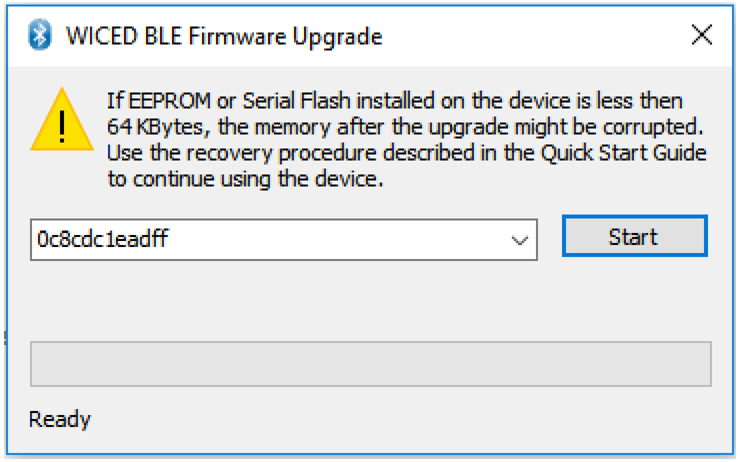
3. In the Select device dialog, select the target module device, and then click **OK**.



Select device

```
0c:8c:dc:1e:ad:ff
15:c1:b8:35:b1:87
3b:f7:56:c8:a6:c8
51:7f:be:3e:22:43
54:b0:2a:3d:e4:73
fc:ec:17:e4:58:42
```

4. The WICED BLE Firmware Upgrade dialog shows the selected target device. Click **Start** to upgrade with the new firmware image.



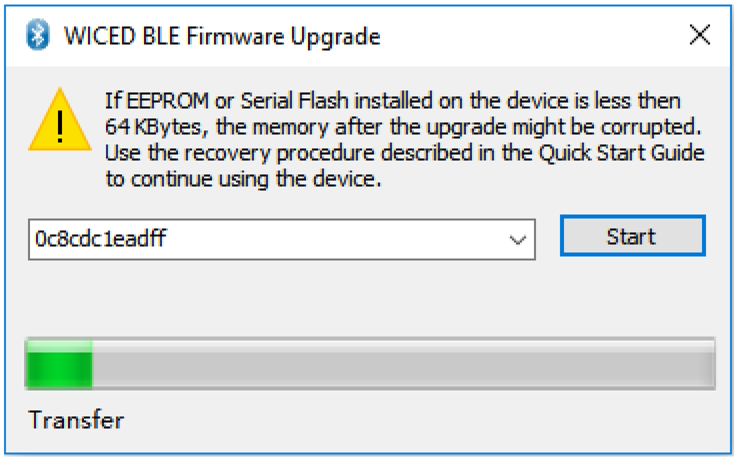
WICED BLE Firmware Upgrade

If EEPROM or Serial Flash installed on the device is less then 64 KBytes, the memory after the upgrade might be corrupted. Use the recovery procedure described in the Quick Start Guide to continue using the device.

0c8cdc1eadff

Ready

5. Wait the firmware upgrade process to finish. While the firmware upgrade is in progress, do not perform any operation or power OFF the module device. Also, do not operate or close the WICED BLE Firmware Upgrade application, which might cause the upgrade process to fail.



WICED BLE Firmware Upgrade

If EEPROM or Serial Flash installed on the device is less then 64 KBytes, the memory after the upgrade might be corrupted. Use the recovery procedure described in the Quick Start Guide to continue using the device.

0c8cdc1eadff

After the upgrade process is complete, the module will reboot automatically and execute the new image.

4 Application Design Examples

The examples in this section describe the hardware design and platform configuration necessary for some common types of applications. You can use any of these exactly as described for your design, or modify as needed.

4.1 Smart MCU Host with 4-Wire UART and Full GPIO Connections

This design takes allows maximum functionality with an external host microcontroller, including efficient sleep state control and optional CYSPP communication.

4.1.1 Hardware Design

Include the following design elements in your hardware:

- Module UART_TX pin to host UART RX pin
- Module UART_RX pin to host UART TX pin
- Module UART_CTS pin to host UART RTS pin
- Module UART_RTS pin to host UART CTS pin
- Module FACTORY_TR, CYSPP, CP_ROLE, LP_MODE, and ATEN_SHDN pins to digital output host GPIOs
- Module LP_STATUS, DATA_READY, and CONNECTION pins to high-impedance digital input host GPIOs

4.1.2 Module Configuration

Most configuration settings will depend on your communication requirements. However, you can make one or more of the following changes:

- Change device name with gap_set_device_name (SDN, ID=4/15)
- Change CYSPP connection key, security requirements, or both with p_cyspp_set_parameters (.CYSPPSP, ID=10/3)
- Change CYCommand security or disable entirely with p_cycommand_set_parameters (.CYCOMSP, ID=11/1)
- Enable system-wide Deep Sleep with system_set_sleep_parameters (SSLP, ID=2/19)
- Enable flow control and optionally change UART parameters with system_set_uart_parameters (STU, ID=2/25)

4.1.3 Host Configuration

The external host must match EZ-Serial's configured UART communication. With factory default settings, this will be 115200,8/N/1 with no flow control. However, you should enable and use flow control if the host supports it.

Use the host API library described in Host API Library to facilitate easy API communication between the host and the module, making sure to properly assert and de-assert the module's LP_MODE pin if you have enabled system-wide Deep Sleep.

Enable a falling-edge interrupt on the DATA_READY signal to allow the host to know when it needs to parse incoming serial API or CYSPP data. This pin will remain asserted (LOW) until no more data exists in the module's serial transmit buffer.

Monitor the CONNECTION signal for a simple indicator of BLE connectivity without needing to parse all possible API events from the module. This can be especially helpful when using CYSPP mode.

4.2 Dumb Terminal Host with CYSPP and Simple GPIO State Indication

This design takes advantage of the factory default EZ-Serial configuration and support for automatic CYSPP connectivity. It is best suited for applications where the external host cannot or does not need to impose any control over the EZ-Serial platform via API commands or events.

4.2.1 Hardware Design

Include the following design elements in your hardware:

- Module CYSPP pin to GND (force CYSPP data mode at all times, no API communication)
- Module UART_TX pin to host UART RX pin
- Module UART_RX pin to host UART TX pin
- Optional for flow control:
  - Module UART_CTS pin to host UART RTS pin
  - Module UART_RTS pin to host UART CTS pin
- Optional for connectivity status:
  - Module CONNECTION pin to LED (Active low)

4.2.2 Module Configuration

The factory default configuration provides most of the behavior required. However, you may wish to make one or more of the following changes:

- Change device name with gap_set_device_name (SDN, ID=4/15)
- Change CYSPP connection key, security requirements, or both with p_cyspp_set_parameters (.CYSPPSP, ID=10/3)
- Change CYCommand security or disable entirely with p_cycommand_set_parameters (.CYCOMSP, ID=11/1)
- Change system Sleep settings with system_set_sleep_parameters (SSLP, ID=2/19)
- Change UART baud or other parameters with system_set_uart_parameters (STU, ID=2/25)

With the CYSPP pin asserted in hardware and the API inaccessible, you can make these changes over CYCommand mode, as described in Remote Control Examples with CYCommand.

4.2.3 Host Configuration

The external host must match EZ-Serial's configured UART communication. With factory default settings, this will be 115200,8/N/1 with no flow control. However, you should enable and use flow control if the host supports it.

If the host supports a simple "enable" control line for whether it is safe to send data, use the module's CONNECTION pin. This signal will be asserted (LOW) only when the CYSPP data pipe is fully established.

4.3 Module-Only Application with Beacon Functionality

This design requires no special external hardware and only minimal initial configuration to define the type of beaconing desired.

4.3.1 Hardware Design

For correct operation, the module only requires power to the supply pins. You may also wish to include test pad or header access to the UART interface and status pins such as LP_STATUS or CONNECTION during prototyping, as this can greatly simplify debugging if necessary.

4.3.2 Module Configuration

Make the following changes from the factory default configuration:

- Disable CYSPP mode with p_cyspp_set_parameters (.CYSPPSP, ID=10/3)
- Change CYCommand security or disable it with p_cycommand_set_parameters (.CYCOMSP, ID=11/1)
- Enable system-wide Deep Sleep mode with system_set_sleep_parameters (SSLP, ID=2/19)
- Configure non-connectable (broadcast-only) with gap_set_adv_parameters (SAP, ID=4/23)
- Configure desired beaconing with p_ibeacon_set_parameters (.IBSP, ID=12/1) or p_eddystone_set_parameters (.EDDYSP, ID=13/1)

If the hardware design does not expose the UART interface, you can apply this initial configuration over CYCommand mode, as described in Remote Control Examples with CYCommand.

4.3.3 Host Configuration

The simple automatic beacon design does not require any host hardware, and therefore needs no host configuration.

5 Host API Library

The host library implements a protocol parser/generator that communicates with the EZ-Serial firmware using the API protocol. The provided library is written in standard C and wraps all API methods into easy-to-use command functions or response/event callbacks. This section describes how to use the library as designed, how to port it to other platforms, or how to create your own library if the provided code is not suited for direct use or porting for any reason.

5.1 Host API Library Overview

5.1.1 High-Level Architecture

The host library communicates with the EZ-Serial firmware platform, providing the host side of the command, response, or event communication mechanism that the module implements. The host must perform the following over the UART interface:

- Read and parse incoming data (may be either response or event packets)
- Validate packets using checksum
- Trigger application-defined callbacks when incoming packets arrive
- Generate and send outgoing data (command packets)

The protocol parser and generator on the module side must strictly follow these rules:

- Events may be generated by the module at any time.
- Every command received from the host will immediately generate a response.
- An event generated (for example, by a GPIO interrupt) while a command is being processed will not interrupt the command-response packet flow, but will be sent out after the response packet is sent.

The parser and generator on the host side must operate under these assumptions.

5.1.2 Host Library Design

Host communication with an EZ-Serial-based module requires only that the incoming module-to-host byte stream is processed correctly, and that the outgoing host-to-module byte stream is properly formatted. To simplify this and provide a convenient layer of abstraction, the host API library provides a simple “parse” function for incoming bytes, and “wrapper” command functions which convert named parameter lists into binary packets ready for transmission.

Other than expecting standard C compiler functionality and little-endian byte order, the library is intentionally platform-agnostic. The source of incoming data does not matter; the internal methods only process the data after it arrives. The destination of outgoing data also does not matter; the internal methods only perform packetization and buffering of data so that it is ready to transmit. This improves portability, since UART peripherals are accessed differently on different platforms, and a single library cannot provide support across all (or even very many) platforms if the UART peripheral implementation is built into the library itself.

5.2 Implementing a Project Using the Host API Library

5.2.1 Basic Application Architecture

Any host application which uses the EZ-Serial API library must follow the same basic behavior:

1. Set up UART peripheral for incoming and outgoing data.
2. Assign hardware-specific input/output callback methods.
3. Monitor UART for incoming data, and send to parser.
4. Handle event/response packets sent to callback handler.
5. Call command wrapper functions as needed for application.

Figure 5-1 shows the process.

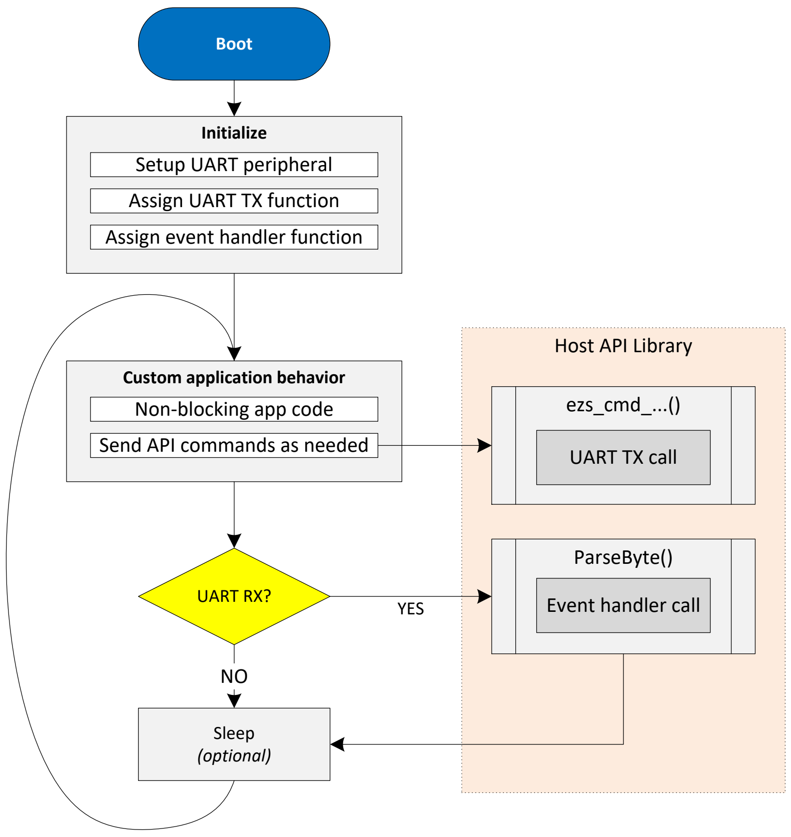


Figure 5-1. EZ-Serial Host API Library Application Flow

The host API library contains the core parsing and generating functions necessary to translate incoming data into callbacks and command function calls into binary packets.

5.2.2 Exposed API Functions

The generic host API implementation written in C provides the methods listed in Table 5-1.

Function	Description
`EZSerial_Init`	Initializes parser and callback functions used for event handling, serial output, and serial input
`EZSerial_Parse`	Processes incoming bytes and triggers event callback function when response or event packet is successfully processed
`EZSerial_FillPacketMetaFromBinary`	Fills binary packet metadata in ezs_packet_t structure based on 4-byte binary packet header content (used internally within EZSerial_Parse)
`EZSerial_SendPacket`	Sends binary packet and checksum byte using host-specific output callback function
`EZSerial_WaitForPacket`	Reads data using host-specific input callback function in a blocking or non-blocking way depending on timeout argument (calls EZSerial_Parse as part of its functionality)

Table 5-1. EZ-Serial Host APIs

The application is responsible for providing implementation functions for three methods, assigned to the function pointers listed in Table 5-2.

Function	Description
`EZSerial_AppHandler`	Called whenever a valid incoming packet is observed. This is strictly required in all cases. It is a core element of abstracting incoming packets into callback functions.
`EZSerial_HardwareOutput`	Called whenever the API generator needs to send data to the module over UART. This is required if you intend to use the EZSerial_SendPacket method, or the ezs_cmd_... macros which also use that method. If you are manually sending well-formed binary command packet data directly from your own application, this may be assigned as NULL.
`EZSerial_HardwareInput`	Called whenever the API parser needs to read data from the module over UART. This is required if you intend to use the EZSerial_WaitForPacket method, or the EZS_WAIT_... or EZS_CHECK_... macros which also use that method. If you are manually calling the EZSerial_Parse method after reading bytes in over UART, this may be assigned as NULL.

Table 5-2. EZ-Serial Host Response Handlers

5.2.3 Command Macros

To simplify binary packet creation, the library implements packet builder macros which match the protocol definitions for each command method. For example:

- `ezs_cmd_system_ping()`
- `ezs_cmd_system_reboot()`
- `ezs_cmd_gap_start_adv(mode, type, interval, channels, filter, timeout)`

Commands which fall into the SET/GET categories and may access flash memory for retrieving or storing setting data have two separate command functions for each:

- RAM: `ezs_cmd_gatts_set_parameters(flags)`
- Flash: `ezs_fcmd_gatts_set_parameters(flags)`

To substantially reduce flash usage, these are defined as macros, which make use of a single function that accepts variable arguments:

```
        ezs_output_result_t ezs_cmd_va(uint16 index, uint8 memory, ...)
```

This single method uses the supplied command table index (defined in the library header file as an enumerated list) and the packed binary protocol structure definition to determine the number of arguments needed for any given command and their data types.

In a macro-based approach, it is not possible to perform type checking at compilation; also, the entire command generator implementation uses a tiny quantity of flash memory (well under 1 KB as measured on one 8-bit MCU).

5.2.4 Convenience Macros

If the hardware-specific input and output functions are correctly defined, the library also provides macros to further abstract common behavior into simpler code.

Function	Description
`EZS_SEND_AND_WAIT(CMD, TIMEOUT)`	Sends a command and then calls EZS_WAIT_FOR_RESPONSE
`EZS_WAIT_FOR_PACKET(TIMEOUT)`	Calls EZSerial_WaitForPacket with type set to any
`EZS_WAIT_FOR_RESPONSE(TIMEOUT)`	Calls EZSerial_WaitForPacket with type set to response
`EZS_WAIT_FOR_EVENT(TIMEOUT)`	Calls EZSerial_WaitForPacket with type set to event
`EZS_CHECK_FOR_PACKET()`	Wrapper for EZS_WAIT_FOR_PACKET(0), a non-blocking attempt to read data

Table 5-3. EZ-Serial Host Convenience Macros

The assignable "return value" (evaluated expression result) for these macros is a pointer to an `ezs_packet_t` object. If the process fails at any point for any reason, timeout, command transmission failure, incoming packet in progress, and so on, the pointer value will be 0 (NULL).

5.3 Porting Host API Library to Different Platforms

Since the API protocol uses a packet byte stream, the API host library expects matching byte ordering and packet structure mapping to avoid any extra processing overhead. The module (and low-level Bluetooth spec) uses little-endian byte ordering, so the host must as well for all multi-byte integer data.

The example application code provided with the library to demonstrate EZ-Serial API usage includes a block of code, which can verify proper support and configuration of byte ordering and structure packing. While it is not possible to provide a single, comprehensive cross-platform implementation of a structure packing macro due to variations between compilers, it is possible to definitively test whether the existing code will work properly. This can quickly identify and avoid potential problems that are otherwise very difficult to troubleshoot.

No special C extensions are used; tested compilers are GCC or GCC-compliant and follow the default C89 ruleset since no additional extensions are enabled.

5.4 Using the API Definition JSON File to Create a Custom Library

The JSON schema used for the API definition has the following structure:

- `info` (single dictionary)
  - `date` – Definition revision date
  - `version` – API protocol definition version
- `groups` (list of dictionaries) […
  - `id` – Numeric ID assigned to group
  - `name` – Alpha name assigned to group (for example, “gap”)
  - `commands` (list of dictionaries) […
    - `id` – Numeric ID assigned to command
    - `name` – Alpha name assigned to command (for example, “start_adv”)
    - `flashopt` – Boolean flag indicating flash storage for settings
    - `parameters` (list of dictionaries) […
      - `type` – Data type (for example, “uint16”)
      - `name` – Alpha name assigned to parameter (for example, “mode”)
      - `textname` – text-mode equivalent (for example, “M”)
      - `required` – Boolean flag indicating optional or required parameter
      - `format` – Intended data presentation format (for example, “string” or “hex”)
      - `default` – Fixed default value if optional parameter
    - `returns` (list of dictionaries) […see `parameters`…]
    - `references` (single dictionary)
      - `commands` (dictionary)
      - `events` (dictionary)
  - `events` (list of dictionaries) […see `commands`…

6 Troubleshooting

EZ-Serial is designed to be as robust and intuitive as possible, but it is always possible for something to go wrong. The section can help narrow down the cause of failure and identify solutions in some cases.

6.1 UART Communication Issues

If you are unable to send or receive data as expected over the UART interface, perform the following steps:

1. Ensure that VDD, VDDR, and GND pins are properly connected (VDDR also requires power).
2. Ensure that VDD and VDDR have a stable supply within the supported range (typically 3 V – 5 V).
3. Ensure that the UART data pins are properly connected:
   a. Module UART_RX to host TX
   b. Module UART_TX to host RX
4. If flow control is enabled or expected, ensure that the UART flow control pins are properly connected:
   a. Module UART_RTS to host CTS
   b. Module UART_CTS to host RTS
5. Ensure that the ATEN_SHDN pin is floating or HIGH to avoid forced hibernation. If this pin is LOW at any time during or after boot, the CPU, radio, and peripherals will remain completely disabled and no UART communication will be possible.
6. Ensure that the CYSPP pin is floating or HIGH to avoid entry into CYSPP mode. When CYSPP is active, API communication is disabled, and this can appear as a non-communicative state until a connection is established.
7. Drive or strongly pull the LP_MODE pin LOW to disable Sleep mode. This is not necessary in most cases, but it can help eliminate potential uncertainty during testing.
8. Reset the module and monitor the UART_TX pin during the boot process. If the module boots normally (CYSPP pin de-asserted), the system_boot (BOOT, ID=2/1) API event should occur at the configured baud rate and in the configured protocol mode. With factory default settings, these values are 115200 baud and text mode. If possible, verify activity using an oscilloscope or a logic analyzer.
9. If attempting to communicate using the API protocol, ensure that your command packet structures are correct per the definitions in Protocol Structure and Communication Flow.
10. If you are sending commands in binary mode and the commands in use have any variable-length arguments (data type of uint8a or longuint8a), ensure that the argument has the correct `<length> [data`$_0$`, data`$_1$`, ..., data`$_N$`]` format. Omitting the length byte will cause the API parser to interpret the packet incorrectly.
11. If you are experiencing data corruption or loss on module-to-host transfers and using the BLE Pioneer kit with the "KitProg" firmware on the PSoC® 5LP MCU acting as the USB-to-UART bridge, ensure that you have the latest version of PSoC Programmer and have updated the KitProg firmware on the BLE Pioneer kit according to the PSoC Programmer user guide. KitProg firmware v2.17 and older has a known stability issue when used with certain host PC communication libraries such as PySerial for Python.

6.2 BLE Connection Issues

If you are unable to connect to or from a remote device, perform the following steps:

1. If attempting to initiate a connection to a remote peripheral/slave device:
   a. Ensure that the local device is in an idle state, not advertising or scanning, or connected to another device. You can stop these various operations with the gap_stop_adv (/AX, ID=4/9) API command, gap_stop_scan (/SX, ID=4/11) API command, and gap_disconnect (/DIS, ID=4/5) API command, respectively. Note that the factory default configuration will automatically boot into an advertising state due to CYSPP settings.
   b. Ensure that the remote device is advertising in a connectable state. Try scanning with the gap_start_scan (/S, ID=4/10) API command in "observation" mode to monitor for all advertising devices.
   c. Ensure that the remote device is not too far away or in any other situation resulting in very low signal strength. Scanning as described in 2.a will also reveal this with observation of scan result remote signal strength indication (RSSI) values.
   d. Ensure that you have specified the correct Bluetooth connection (MAC) address and address type (public or private). A connection attempt with the right Bluetooth address with the wrong address type will fail.
   e. Ensure that you are in the correct state to initiate a connection (idle, not advertising, scanning, connecting, or connected already).
   f. Try connecting to a different peripheral/slave device to see whether the problem persists.
2. If attempting to initiate a connection from a remote central/master device:
   a. Ensure that the module is advertising in a connectable state. Start advertising specifically in the "connectable, undirected" mode using the gap_start_adv (/A, ID=4/8) API command, and watch for the expected gap_adv_state_changed (ASC, ID=4/2) API event indicating that the state actually changed to "active."
   b. Ensure that you have set properly formed custom advertising data with gap_set_adv_data (SAD, ID=4/19) if you have disabled automatic advertising packet management with gap_set_adv_parameters (SAP, ID=4/23). Advertisement packets without a standard **Flags** field (usually `[ 02 01 06 ]`) will not appear in a generic scan. See Customizing Advertisement and Scan Response Data) for details.

6.3 GPIO Signal Issues

If you are not observing the expected behavior for GPIO input, output, or both signals, perform the following steps:

1. Ensure that the pins you have connected are correct based on your chosen module. See GPIO Pin Map for Supported Modules for per-device pin map details.
2. If a special-function pin is not generating or responding to an external signal as expected, ensure that the function is enabled using the gpio_set_function (SIOF, ID=9/3) API command. Note that all functions are enabled in the factory default configuration and there should not be a need to be re-enabled to work out of the box.
3. If a special-function output pin is not sufficiently driving a connected external device's input logic, ensure that the "strong drive" mode is enabled for that functional pin by using the gpio_set_function (SIOF, ID=9/3) API command.

7 API Protocol Reference

This section describes the API protocol that EZ-Serial uses. This protocol allows an external host to control the module, in addition to any GPIO signals involved in the design. The protocol follows a strict set of rules to make deterministic host-side behavior possible.

This section describes version 1.0 of the API protocol.

7.1 Protocol Structure and Communication Flow

7.1.1 API Protocol Formats

EZ-Serial implements a unified set of functionalities that can be accessed using either text or binary API communication. These two formats cover the same feature set, and do not offer control in any way (except for optional argument support in text mode, see Text Format Overview).

7.1.1.1 Text Format Overview

The text protocol definition is comprised entirely of printable ASCII characters for ease of use in terminal software. Response and Event packets sent from the module will end with "\r\n" characters (0x0D, 0x0A). Commands sent to the module may end with either or both. Unlike the binary mode (see Binary Format Overview), the text protocol does not contain any checksum data or have a command entry timeout.

7.1.1.2 Binary Format Overview

The binary protocol uses a fixed packet structure for every transaction in either direction. This fixed structure comprises a 4-byte header, followed by an optional payload of up to 2047 bytes (length specifier field is 11 bits wide).

No currently defined binary packet contains more than 520 payload bytes at this time, and very few contain more than 48. The API reference material lists every fixed or minimum/maximum length value for all commands, responses, and events within the protocol.

The payload carries information related to the command, response, or event. If present, this payload always comes immediately after the header. All data in the payload will be contained within one or more of the datatypes specified in API Protocol Data Types.

To simplify the implementation of parsers and generators both inside the firmware and on external host microcontrollers, any packet may have a maximum of one variable-length data member (byte array or string), and if present, it must be the last element in the payload.

7.1.2 API Protocol Data Types

The data types implemented for individual parameters/arguments in the API protocol are described in Table 7-1, including representative text and binary examples.

In both text and binary modes, all negative numbers are represented in two's complement form. In this form, the most significant bit is the sign bit, which indicates a negative number if set. The remaining bits count upward from the bottom of the selected (positive or negative) range. For example, the value 0x80 is the bottom of the "int8" range, -128.

Type	Bytes	Description	Example
`uint8`	1	Unsigned 8-bit integer. Range is 0 to 255.	Text Mode: - “10” = 0x10, decimal 16 - “9A” = 0x9A, decimal 154 Binary Mode: - [10] = 0x10, decimal 16 - [9A] = 0x9A, decimal 154
`int8`	1	Signed 8-bit integer. Range is -128 to 127.	Text Mode: - “10” = 0x10, decimal 16 - “9A” = 0x9A, decimal -102 Binary Mode: - [10] = 0x10, decimal 16 - [9A] = 0x9A, decimal -102
`uint16`	2	Unsigned 16-bit integer. Range is 0 to 65,535.	Text Mode: - “1234” = 0x1234, decimal 4,660 - “9ABC” = 0x9ABC, decimal 39,612 Binary Mode: *(little-endian)* - [34 12] = 0x1234, decimal 4,660 - [BC 9A] = 0x9ABC, decimal 39,612
`int16`	2	Signed 16-bit integer. Range is -32,768 to 32,767.	Text Mode: - “1234” = 0x1234, decimal 4,660 - “9ABC” = 0x9ABC, decimal -25,924 Binary Mode: *(little-endian)* - [34 12] = 0x10, decimal 4,660 - [BC 9A] = 0x9ABC, decimal -25,924
`uint32`	4	Unsigned 32-bit integer. Range is 0 to 4,294,967,295.	Text Mode: - “12345678” = 0x12345678 decimal 305,419,896 - “9ABCDEF0” = 0x9ABCDEF0, decimal 2,596,069,104 Binary Mode: *(little-endian)* - [78 56 34 12] = 0x12345678 decimal 305,419,896 - [F0 DE BC 9A] = 0x9ABCDEF0 decimal 2,596,069,104
`int32`	4	Signed 32-bit integer. Range is -2,147,438,648 to 2,147,483,647.	Text Mode: - “12345678” = 0x12345678 decimal 305,419,896 - “9ABCDEF0” = 0x9ABCDEF0, decimal -1,698,898,192 Binary Mode: *(little-endian)* - [78 56 34 12] = 0x12345678 decimal 305,419,896 - [F0 DE BC 9A] = 0x9ABCDEF0 decimal -1,698,898,192
`macaddr`	6	48-bit MAC address.	Text Mode: - “112233AABBCC” = 11:22:33:AA:BB:CC Binary Mode: *(little-endian)* - [CC BB AA 33 22 11] = 11:22:33:AA:BB:CC
`uint8a`	1+	Array of `uint8` bytes, with prefixed one-byte length value. Supported length is 0-255 bytes.	Text Mode: *(length omitted, detected automatically)* - “41424344” = Length 4, Data [41 42 43 44] - “1122334455” = Length 5, Data [11 22 33 44 55] Binary Mode: - [04 41 42 43 44] = Ln. 4, [41 42 43 44] - [05 11 22 33 44 55] = Ln. 5, [11 22 33 44 55]

Type	Bytes	Description	Example
`longuint8a`	2+	Array of `uint8` bytes, with prefixed two-byte length value. Supported length is 0-65535 bytes.	Text Mode: *(length omitted, detected automatically)* - "41424344" = Length 4, Data [41 42 43 44] - "1122334455" = Length 5, Data [11 22 33 44 55] Binary Mode: - [04 00 41 42 43 44] = Length 4, Data [41 42 43 44] - [05 00 11 22 33 44 55] = Length 5, Data [11 22 33 44 55] Note the 16-bit **length** prefix in binary mode is transmitted in little-endian byte order, so the value 0x0005 is sent as [05 00].
`string`	1+	String of `uint8` bytes, with prefixed one-byte length value. Length is 0-255 bytes.	These two datatypes are represented in binary exactly the same way as uint8a and longuint8a data, but in text mode they are entered and displayed exactly as-is, with the assumption that they contain printable ASCII characters. An example of a string value entered and displayed in this way is the Device Name value.
`longstring`	2+	String of `uint8` bytes, with prefixed two-byte length value. Length is 0-65535 bytes.	

Table 7-1. API Protocol Data Types

7.1.3 Binary Format Details

7.1.3.1 Byte Ordering and Structure Packing

The protocol implements a collection of common data types representing signed and unsigned integers, arrays of binary bytes, arrays of printable characters, and certain technology-specific data (6-byte MAC address).

In text mode, all data except string/longstring values are represented as ASCII hexadecimal characters, without a leading "`0x`" or other prefix. For example, the decimal value 154 is shown or entered as "`9A`". Leading zeros may be omitted. Also, in text mode, all multi-byte integer and MAC address data shall be entered in big-endian byte order. For example, the value 0x1234 is entered or displayed as "`1234`". The MAC address 11:22:33:AA:BB:CC is entered or displayed as "`112233AABBCC`".

In binary mode, all multi-byte integers and MAC address data must be transmitted serially in little-endian byte order. For example, the value 0x1234 is two bytes transmitted as `[ 34 12 ]`, and the MAC address 11:22:33:AA:BB:CC is six bytes transmitted as `[ CC BB AA 33 22 11 ]`.

The Bluetooth Low Energy specification mandates little-endian byte order internally, so data from the stack is naturally presented to the application layer in this byte order. Further, many common embedded processors use little-endian data storage, including the Arm® Cortex®-M0 in CYBT-4130XX-02 EZ-BT modules. As a result, host MCU firmware can read in a serial byte stream into a contiguous SRAM buffer, and define a structure like the following:

```
typedef struct {
    uint16 app;
    uint32 stack;
    uint16 protocol;
    uint8 hardware;
    uint8 cause;
    macaddr address;
} ezs_evt_system_boot_t;
```

The host MCU application can directly map this structure onto the packet buffer in memory with no additional byte-swap operations. Accessing any one of the structure members will give correct access to the data in the packet. This arrangement allows for minimal flash usage and CPU execution time.

7.1.3.2 Binary Packet Header

Table 7-2 describes the binary packet 4-byte header structure.

Byte	Field(s)	Description
0	[7:6] - Type [5:4] - Memory [2:0] - Length MSB	**Type:** The "Type" field is a 2-bit value (MSB aligned) indicating whether the packet is a command, response, or event. Options are as follows: - 00: RESERVED - 01: RESERVED - 10: Event (module-to-host) - 11: Response (module-to-host) or Command (host-to-module) Protocol methods follow this convention when the "Type" value is aligned properly: - Commands sent to the module begin with 0xC0 - Responses sent to the host begin with 0xC0 - Events sent to the host begin with 0x80 **Memory:** The "Memory" field is a 2-bit value (MSB aligned) indicating whether a command sent accesses the runtime value stored in RAM, or the boot value stored in flash. This field is ignored for commands which do not read or write configuration data stored in either flash or RAM. Options are as follows: - 00: Runtime (RAM) - 01: Boot (Flash) - 10: RESERVED - 11: RESERVED The values stored in RAM and flash may be the same, if the user has not modified the runtime value separately from the boot value since the last power-on or reset. **Length MSB:** The length MSB field contains the upper three bits of the payload length value (11 bits total). See below for length detail. The "Type", "Memory", and "Length MSB" bitfields are positioned within Byte 0 as follows: `0b` **`TTMM`** `0`**`LLL`** The remaining bit in the middle is currently reserved and should always be set to zero.
1	Length LSB	This value indicates the number of bytes in the payload. It may be 0 to indicate no payload, or any value up to the 11-bit maximum of 2047 (combining the LSB and MSB fields together). Typically, packets fit easily within a 64-byte buffer. However, a few packets such as local GATT reads and writes may potentially be much longer than this. Protocol methods which may require or generate long packets will be documented specifically.
2	Group ID	All protocol methods are organized into logically separate groups, such as GAP, GATT server, L2CAP, CYSPP, and so on. This byte represents the group ID, between 0 and 255. A single group ID applies to all commands, responses, and events within that group.
3	Method ID	Within each group and packet type, every protocol method has a unique ID between 0 and 255. Command/response pairs always have matching IDs. Command/response pairs and events are separate collections and may have overlapping method IDs, each in a set starting from 0.

Table 7-2. Binary Packet Header Structure

7.2 API Commands and Responses

All commands and responses implemented in the API protocol are described in detail in this section. API events are documented separately. See Error Codes for a master list of all possible error codes resulting from commands.

Important things to note about the reference material in the following sections:

- The 16-bit "result" code is common to every response, and always occupies the same position in the packet (immediately after the binary header or text name). For simplicity, this **result** field is omitted from each list of response parameters in the tables below.
- The "Text" column in each "Command Arguments" table contains the text code for each argument. Required arguments have a red asterisk (*) next to their text codes. Optional arguments in text mode will not have a red asterisk.
- All command arguments are required in binary mode, because binary parsing depends on predictable argument position and byte width for proper data identification and unpacking.
- The "Command-Specific Result Codes" list appearing for some commands do not include some errors that may result from command entry or protocol format mistakes. These common errors include:
  - `0x0203 - EZS_ERR_PROTOCOL_UNRECOGNIZED_COMMAND`
  - `0x0206 - EZS_ERR_PROTOCOL_SYNTAX_ERROR`
  - `0x0207 - EZS_ERR_PROTOCOL_COMMAND_TIMEOUT`
  - `0x0209 - EZS_ERR_PROTOCOL_INVALID_CHECKSUM`
  - `0x020A - EZS_ERR_PROTOCOL_INVALID_COMMAND_LENGTH`
  - `0x020B - EZS_ERR_PROTOCOL_INVALID_PARAMETER_COUNT`
  - `0x020C - EZS_ERR_PROTOCOL_INVALID_PARAMETER_VALUE`
  - `0x020D - EZS_ERR_PROTOCOL_MISSING_REQUIRED_ARGUMENT`
  - `0x020E - EZS_ERR_PROTOCOL_INVALID_HEXADECIMAL_DATA`
  - `0x020F - EZS_ERR_PROTOCOL_INVALID_ESCAPE_SEQUENCE`
  - `0x0210 - EZS_ERR_PROTOCOL_INVALID_MACRO_SEQUENCE`

See Error Codes for details on these and other error codes.

Commands and responses are broken down into the following groups:

- Protocol Group (ID=1)
- System Group (ID=2)
- OTA Group (ID=3)
- GAP Group (ID=4)
- GATT Server Group (ID=5)
- GATT Client Group (ID=6)
- SMP Group (ID=7)
- L2CAP Group (ID=8)
- GPIO Group (ID=9)
- CYSPP Group (ID=10)
- CYCommand Group (ID=11)
- iBeacon Group (ID=12)
- Eddystone Group (ID=13)

7.2.1 Protocol Group (ID=1)

Protocol methods allow you to change the way the API protocol operates while communicating with an external host over the serial interface.

Commands within this group are listed below:

- protocol_set_parse_mode (SPPM, ID=1/1)
- protocol_get_parse_mode (GPPM, ID=1/2)
- protocol_set_echo_mode (SPEM, ID=1/3)
- protocol_get_echo_mode (GPEM, ID=1/4)

Events within this group are documented Protocol Group (ID=1).

7.2.1.1 protocol_set_parse_mode (SPPM, ID=1/1)

Configure new protocol parse mode.

In binary mode, all API packets to and from the module must use a binary format with a fixed header and payload structure, as described in the reference material. In text mode, all commands, responses, and events use a human-readable format that is suitable for typing in a terminal. See Protocol Structure and Communication Flow for details.

Note: When the protocol mode is changed with this command, the effect is immediate. The response packet returned will come in the newly configured format, not the previous format.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`01`	`01`	None.
RSP	`C0`	`02`	`01`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SPPM`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	New parse mode: • 0 = Text mode (factory default) • 1 = Binary mode

Response Parameters:

None.

Related Commands:

- protocol_get_parse_mode (GPPM, ID=1/2)

7.2.1.2 protocol_get_parse_mode (GPPM, ID=1/2)

Obtain current protocol parse mode.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`01`	`02`	None.
RSP	`C0`	`03`	`01`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GPPM`	`0x000F`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	Current parse mode: • 0 = Text mode (factory default) • 1 = Binary mode

Related Commands:

- protocol_get_parse_mode (GPPM, ID=1/2)

7.2.1.3 protocol_set_echo_mode (SPEM, ID=1/3)

Configure new protocol echo mode.

The protocol echo mode applies when using text mode API protocol over UART to communicate with the module. Enabling echo will result in each input byte being sent back to the host after it is parsed. Local echo may be desirable during a terminal session, but it is typically simpler disable it for MCU communication so that the MCU only needs to parse response and event data.

Note: Local echo does not apply in CYSPP data mode or CYCommand data mode, regardless of the protocol format in use. It only affects communication over the UART interface when using the API protocol in text mode.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`01`	`03`	None.
RSP	`C0`	`02`	`01`	`03`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SPEM`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	New echo mode: • 0 = Disabled • 1 = Enabled (factory default)

Response Parameters:

None.

Related Commands:

- protocol_get_echo_mode (GPEM, ID=1/4)

7.2.1.4 *protocol_get_echo_mode (GPEM, ID=1/4)*

Obtain current protocol echo mode.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`01`	`04`	None.
RSP	`C0`	`03`	`01`	`04`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GPEM`	`0x000F`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	Current echo mode: • 0 = Disabled • 1 = Enabled (factory default)

Related Commands:

- protocol_set_echo_mode (SPEM, ID=1/3)

7.2.2 System Group (ID=2)

System methods relate to the core device and describe functionality such as boot status, setting or obtaining device address information, and resetting to an initial state.

Commands within this group are listed below:

- system_ping (/PING, ID=2/1)
- system_reboot (/RBT, ID=2/2)
- system_dump (/DUMP, ID=2/3)
- system_store_config (/SCFG, ID=2/4)
- system_factory_reset (/RFAC, ID=2/5)
- system_query_firmware_version (/QFV, ID=2/6)
- system_query_random_number (/QRND, ID=2/8)
- system_write_user_data (/WUD, ID=2/11)
- system_read_user_data (/RUD, ID=2/12)
- system_set_bluetooth_address (SBA, ID=2/13)
- system_get_bluetooth_address (GBA, ID=2/14)
- system_set_sleep_parameters (SSLP, ID=2/19)
- system_get_sleep_parameters (GSLP, ID=2/20)
- system_set_tx_power (STXP, ID=2/21)
- system_get_tx_power (GTXP, ID=2/22)
- system_set_transport (ST, ID=2/23)
- system_get_transport (GT, ID=2/24)

- system_set_uart_parameters (STU, ID=2/25)
- system_get_uart_parameters (GTU, ID=2/26)

Events within this group are documented in System Group (ID=2).

7.2.2.1 *system_ping (/PING, ID=2/1)*

Test API communication.

Pinging the module verifies that the host and the module can communicate properly in API mode. The module should immediately generate a well-formed response to this command if communication is working correctly. Host-side initialization routines often begin with this step.

The runtime values returned in the response to this command are calculated based on the built-in 32768 Hz watch clock oscillator (WCO) that is used to manage low-power operation of the BLE stack. No external hardware is required for this functionality.

Note: Pinging the module does not serve any purpose other than to verify proper communication, or to obtain runtime since reset. You do not need to ping at regular intervals to keep a connection alive or prevent the module from entering low-power states. The platform automatically maintains BLE connections unless commanded otherwise. See Managing Sleep States for sleep behavior detail.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`01`	None.
RSP	`C0`	`08`	`02`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/PING`	`0x000B`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint32`	`runtime`	`R`	Number of seconds since boot
`uint16`	`fraction`	`F`	Fraction of a second (units are 1/32768)

7.2.2.2 *system_reboot (/RBT, ID=2/2)*

Reboot module.

A module reboot takes effect immediately. Any configuration settings not stored in flash will revert to their boot-level values, and any active connections will be terminated without clean closure (remote peer will detect a supervision timeout). See Saving Runtime Settings in Flash for details on how to store settings in flash to make them persist across reboots and power-cycles.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`02`	None.
RSP	`C0`	`02`	`02`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/RBT`	`0x000A`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- system_store_config (/SCFG, ID=2/4) – Stores all configuration items in flash before rebooting, if desired

Related Events:

- system_boot (BOOT, ID=2/1) – Will occur once the reboot process completes

7.2.2.3 system_dump (/DUMP, ID=2/3)

Dump current device configuration or state information.

Performing a system dump will generate a sequence of system_dump_blob (DBLOB, ID=2/5) API events, each containing up to 16 bytes, until all data transmission is complete. You can provide this information for troubleshooting if requested by our support staff.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`02`	`03`	None.
RSP	`C0`	`04`	`02`	`03`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/DUMP`	`0x0012`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`type`	`T`	Type of information to dump: • 0 = Runtime configuration data (default) • 1 = Boot-level configuration data • 2 = Factory-level configuration data • 3 = System state data

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`length`	`L`	Number of bytes to be dumped: • Configuration data is 674 bytes (0x02A2) • State data is 1,955 bytes (0x07A3)

Related Commands:

- system_store_config (/SCFG, ID=2/4)

Related Events:

- system_dump_blob (DBLOB, ID=2/5)

7.2.2.4 *system_store_config (/SCFG, ID=2/4)*

Stores all configuration settings into flash.

This command applies all runtime settings into the boot-level configuration area stored in nonvolatile flash. See Configuration Settings, Storage, and Protection for details on different configuration areas.

> **WARNING:** This command briefly halts CPU execution, and may cause a connectivity loss for any open connections if this occurs during a precise moment when low-level BLE interrupts require processing. If possible, only use this command while not connected to avoid this potential issue.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`04`	None.
RSP	`C0`	`02`	`02`	`04`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/SCFG`	`0x000B`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- system_factory_reset (/RFAC, ID=2/5)

7.2.2.5 *system_factory_reset (/RFAC, ID=2/5)*

Resets all settings to factory defaults and reboot.

This command reverts all configuration settings back to the values stored in the factory default area. After applying these default values, the system reboots immediately.

> **WARNING:** If you have configured custom serial communication settings using the system_set_transport (ST, ID=2/23) API command, using this command will undo these changes and may prevent working communication until you reconfigure your host device to the factory default transport settings. See Factory Default Behavior for details on these settings.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`05`	None.
RSP	`C0`	`02`	`02`	`05`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/RFAC`	`0x000B`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Events:

- system_factory_reset_complete (RFAC, ID=2/3) – Occurs after the settings are reset
- system_boot (BOOT, ID=2/1) – Occurs after the system reboots

Example Usage:

- See Factory Reset via API Command

7.2.2.6 system_query_firmware_version (/QFV, ID=2/6)

Queries EZ-Serial firmware version information.

This command provides the same version details that the system_boot (BOOT, ID=2/1) event contains.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`06`	None.
RSP	`C0`	`0D`	`02`	`06`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/QFV`	`0x002C`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint32`	`app`	`E`	Application version number (0x0100010E = 1.0.1 build 14)
`uint32`	`stack`	`S`	BLE stack version number (0x030200FA = 3.2.0 build 250)
`uint16`	`protocol`	`P`	API protocol version number (0x0101 = 1.1)
`uint8`	`hardware`	`H`	Hardware identifier: • 0xE1 = CYBT-423028-02 • 0xE2 = CYBT-423054-02 • 0xE3 = CYBT-423060-02 • 0xE4 = CYBT-413034-02 • 0xE5 = CYBT-413055-02 • 0xE6 = CYBT-413061-02 • 0xE7 = CYBT-483039-02 • 0xE8 = CYBT-483056-02 • 0xE9 = CYBT-483062-02

Related Events:

- system_boot (BOOT, ID=2/1)

7.2.2.7 system_query_random_number (/QRND, ID=2/8)

Queries random number generator for 8-byte pseudo-random sequence.

This command provides simple access to the random number generator in the CYBT-4130XX-02 EZ-BT module. The query always provides exactly eight bytes of random data.

Note: This pseudo-random generation mechanism is FIPS PUB 140-2 compliant.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`08`	None.

	Type	Length	Group	ID	Notes
RSP	`C0`	`0B`	`02`	`08`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/QRND`	`0x001E`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8a`	`data`	`D`	Random 8-byte sequence (1 length byte equal to 0x08, followed by 8 data bytes) **Note:** `uint8a` data type requires one prefixed “length” byte before binary parameter payload

7.2.2.8 system_write_user_data (/WUD, ID=2/11)

Writes arbitrary data to the user flash storage area.

EZ-Serial provides 255 bytes of non-volatile flash storage for application data. This command allows writing 1-32 bytes to any position within this 255-byte area.

Note: You must specify a data offset and length which do not exceed 255 when combined. For example, if you are writing 32 bytes of data, the specified “offset” argument must be 223 (0xDF) or less. If the flash data has not been written before, the default data will always 0.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`04-23`	`02`	`0B`	Variable-length command payload, minimum of 4 (0x4), maximum of 35 (0x23).
RSP	`C0`	`02`	`02`	`0B`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/WUD`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`offset`	`O`*	Offset (0-255)
`uint8a`	`data`	`D`*	Data to write (1-32 bytes) **Note:** `uint8a` data type requires one prefixed “length” byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- system_read_user_data (/RUD, ID=2/12)

7.2.2.9 *system_read_user_data (/RUD, ID=2/12)*

Reads arbitrary data from the user flash storage area.

EZ-Serial provides 255 bytes of nonvolatile flash storage for application data. This command allows reading 1-32 bytes from any position within this 255-byte area.

Note: You must specify a data offset and length which do not exceed 255 when combined. For example, if you are reading 32 bytes of data, the specified "offset" argument must be 223 (0xDF) or less. If the data read has been written before, the return bytes will be 0.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`03`	`02`	`0C`	None.
RSP	`C0`	`03`	`02`	`0C`	Variable-length response payload, minimum of 3 (0x3), maximum of 35 (0x23).

Text Info:

Text Name	Response Length	Category	Notes
`/RUD`	`0x000D-0x004D`	ACTION	Variable-length response payload, minimum of 13 (0xD), maximum of 77 (0x4D).

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`offset`	`O`*	Offset (0-255)
`uint8`	`length`	`L`*	Number of bytes to read (1-32)

Response Parameters:

Data Type	Name	Text	Description
`uint8a`	`data`	`D`	Data read (1-32 bytes) **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Related Commands:

- system_write_user_data (/WUD, ID=2/11)

7.2.2.10 *system_set_bluetooth_address (SBA, ID=2/13)*

Configures a new Bluetooth MAC Address.

This address will be visible to the remote scanning or connected devices, if the module is not operating with privacy enabled. EZ-Serial uses a fixed public address by default, which is generated dynamically based on the chip random number generator. Normally, you do not need to change the Bluetooth address using this command.

By default, this command won't store the new Bluetooth address into the flash and make it available permanently, so this new Bluetooth address will be used and active only before the device was reset or rebooted. After the device was reset or rebooted, the old Bluetooth address will be reused. To store and make the newly configured Bluetooth address effective after reboot or reset, set the "flash" memory scope bit (bit4 of type field) in the binary command packet header or append the '$' character to command names in text mode when sending this command.

Note:

- When privacy is enabled, remote peer devices will see a random address instead of the fixed address. Central or peripheral privacy is not the same as encryption. See related commands and example usage for detail.
- When the new Bluetooth address is set, the device name will not be updated automatically. You must call the gap_set_device_name (SDN, ID=4/15) API to set new device name, if the device name is generated from part of the Bluetooth address.
- When the new Bluetooth address is set, the BLE advertisement must be stopped and restarted to apply new Bluetooth address. It is recommended to stop the advertisement first, and then then set the new Bluetooth address.

- When the Bluetooth address type is set to 0 (public address), bit7 and bit6 of the highest byte must not be set (bda_byte & ~0xC0). When the Bluetooth address type is set to 1 (static random address), bit7 and bit6 of the highest byte must be set (bda_byte | 0xC0). For more details, see Section 1.3 DEVICE ADDRESS of Core_v5.2, Vol 6, Part B.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	07	02	0D	None.
RSP	C0	02	02	0D	None.

Text Info:

Text Name	Response Length	Category	Notes
SBA	0x0009	SET	None.

Command Arguments:

Data Type	Name	Text	Description
macaddr	address	A	New public/static random Bluetooth address. Set all six 0x00 bytes to revert to factory-provided address. This MAC address must be assigned in big-endian.
uint8	type	T	0 indicates public Bluetooth address; 1 indicates static random address.

Response Parameters:

None.

Related Commands:

- system_get_bluetooth_address (GBA, ID=2/14)
- smp_set_privacy_mode (SPRV, ID=7/9)
- smp_query_random_address (/QRA, ID=7/4)

Example Usage:

- See Using Peripheral and Central Privacy

7.2.2.11 system_get_bluetooth_address (GBA, ID=2/14)

Obtains the current public Bluetooth address.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	02	0E	None.
RSP	C0	09	02	0E	None.

Text Info:

Text Name	Response Length	Category	Notes
GBA	0x001C	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
macaddr	address	A	Current public Bluetooth address
uint8	type	T	0 – indicates public Bluetooth address; 1 indicates static random address.

Related Commands:

- system_set_bluetooth_address (SBA, ID=2/13)
- smp_query_random_address (/QRA, ID=7/4)
- smp_set_privacy_mode (SPRV, ID=7/9)

7.2.2.12 system_set_sleep_parameters (SSLP, ID=2/19)

Configures new system-wide Sleep settings.

EZ-Serial automatically enters the most low-power Sleep mode available to maintain required activity (including BLE communication, PWM output, and UART output). While Deep Sleep mode provides the best power efficiency, it also restricts certain operations:

- UART RX requires one or more “dummy” bytes due to the 25 µs CPU wake-up time
- High-resolution PWM output cannot operate since the high-frequency clock is stopped

> **WARNING:** Enabling Deep Sleep with this API command can result in a seemingly non-responsive UART. To address this, prefix all transmissions from the host to the module with one or more 0x00 bytes to ensure that the CPU has enough time to wake up. See Managing Sleep States for details.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`02`	`02`	`13`	None.
RSP	`C0`	`02`	`02`	`13`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SSLP`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`level`	`L`	New maximum system-wide Sleep level: • 0 = Sleep Disabled (Factory Default) • 1 = Power Down Sleep (PDS) • 2 = Shut Down Sleep (SDS) **Note:** When the system enters PDS or SDS Sleep mode, the immediate operation with the BT or LE request may fail. It is normal. Retry and operation will succeed. The firmware BT or LE operation will cause the system wake up, and while waking up, the first operation may fail. So after the system wakes up, the second retry operation can succeed.
`uint8`	`timeout`	`T`	When the level is set to 1 or 2, the timeout value indicates that after system idle for (timeout * 15) seconds, the system will enter PDS or SDS mode. When the level is set to 0, this field is not used, and can be ignored. The value of timeout can be 0~255. When this field is set to 0, it indicates that the system will never enter Sleep mode. **Note:** The real idle timeout value in seconds must be multiplied by 15. The factory default value is 4, so system will enter Sleep mode after 60 seconds, if there is no operation.

Response Parameters:

None.

Related Commands:

- system_get_sleep_parameters (GSLP, ID=2/20)
- gpio_set_pwm_mode (SPWM, ID=9/11) – Configures PWM output
- p_cyspp_set_parameters (.CYSPPSP, ID=10/3) – Configures new CYSPP parameters, including CYSPP data mode Sleep level

Example Usage:

- See Configuring the System-Wide Sleep Level

7.2.2.13 system_get_sleep_parameters (GSLP, ID=2/20)

Obtains the current system-wide Sleep settings.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`14`	None.
RSP	`C0`	`04`	`02`	`14`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GSLP`	`0x0013`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`level`	`L`	New maximum system-wide Sleep level: • 0 = Sleep Disabled (Factory Default) • 1 = Power Down Sleep (PDS) when timeout • 2 = Shut Down Sleep (SDS) when timeout
`uint8`	`timeout`	`T`	When the level is set to 1 or 2, the timeout value indicates that after system idle for (timeout * 30) seconds, the system will enter PDS or SDS mode. When the level is set to 0, this field is not used, and can be ignored. The value of 0 indicates Sleep mode is not always allowed. **Note:** The real idle timeout value in seconds must be multiplied by15. The factory default value is 4, which means 60 seconds.

Related Commands:

- system_set_sleep_parameters (SSLP, ID=2/19)

7.2.2.14 system_set_tx_power (STXP, ID=2/21)

Configures new transmit power for all outgoing radio communications.

This power setting affects all transmissions, including advertising, scan requests and connection requests, and all packets sent during an active connection. Changes will take effect in the next transmitted packet begins.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`02`	`15`	None.
RSP	`C0`	`02`	`02`	`15`	None.

Text Info:

Text Name	Response Length	Category	Notes
`STXP`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`power`	`P`	Available power value can be set; the value must be in the range of 1 and 8. The default set value is 7. See Changing Output Power for details on the TX output power map.

Response Parameters:

None.

Related Commands:

- system_get_tx_power (GTXP, ID=2/22)

7.2.2.15 system_get_tx_power (GTXP, ID=2/22)

Obtains current transmit power for all outgoing radio communications.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`16`	None.
RSP	`C0`	`03`	`02`	`16`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GTXP`	`0x000F`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`power`	`P`	Current active power level value should be in the range of 1 and 8. See 3.1.4for details on the TX output power map.

Related Commands:

- system_get_tx_power (GTXP, ID=2/22)

7.2.2.16 system_set_transport (ST, ID=2/23)

Configures new host communication interface.

This command configures the interface used for wired external host communication. If a change is successful, EZ-Serial will send the response packet in the original configuration, and then switch to the new transport interface.

Note: The current EZ-Serial release supports only the UART transport interface. No other options are available.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`02`	`17`	None.
RSP	`C0`	`02`	`02`	`17`	None.

Text Info:

Text Name	Response Length	Category	Notes
`ST`	`0x0008`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`interface`	`I`	New host transport interface: • 1 = UART (factory default)

Response Parameters:

None.

Related Commands:

- system_get_transport (GT, ID=2/24)
- system_set_uart_parameters (STU, ID=2/25)

7.2.2.17 system_get_transport (GT, ID=2/24)

Obtains the current host transport setting.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`18`	None.
RSP	`C0`	`03`	`02`	`18`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GT`	`0x000D`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`interface`	`I`	Current host transport interface: • 1 = UART (factory default)

Related Commands:

- system_set_transport (ST, ID=2/23)
- system_get_uart_parameters (GTU, ID=2/26)

7.2.2.18 system_set_uart_parameters (STU, ID=2/25)

Configures new UART settings for host communication.

This command configures the UART peripheral behavior used for wired external host communication when the host transport interface is set to “UART” with the system_set_transport (ST, ID=2/23) API command. If a change is successful, EZ-Serial will send the response packet using the original configuration, and then apply the new UART settings.

Note: This command affects protected settings, which means you cannot immediately apply changes to flash. To store new settings in nonvolatile memory, you must send the command once without the flash storage bit/flag, and then resend the same command with the flash storage bit/flag set. This prevents accidental permanent communication lock-out resulting from flash-stored settings that the connected host cannot use. For detail, see Protected Configuration Settings.

WARNING: If you have enabled Deep Sleep using the system_set_sleep_parameters (SSLP, ID=2/19) API command and you are relying on UART data reception to wake the module up from Deep Sleep, the number of dummy bytes needed for wake-up depends on the baud rate chosen, and the recommended dummy byte depends on whether you have enabled even parity.

WARNING: Selecting a baud rate below 9600 and using API protocol communication can result in a situation where EZ-Serial generates API response and event packets faster than the UART interface can transmit them to the host. If this occurs, data will flow continuously out of the module, but it will not respond to incoming commands. The most likely trigger for this is by activating a scan with gap_start_scan (/S, ID=4/10) or starting CYSPP client mode operation (which also begins a scan), which generate scan result events rapidly.

This non-responsive behavior will be improved in a future release. There is a workaround using either of the following:

- If using CYSPP, keep the **CYSPP** pin externally asserted to suppress API output.

- If possible, select a faster baud rate.

- If possible, reduce the quantity of devices in the environment to decrease the scan result count.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	0A	02	19	None.
RSP	C0	02	02	19	None.

Text Info:

Text Name	Response Length	Category	Notes
STU	0x0009	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint32	baud	B	UART baud rate: • Minimum = 300 baud (0x12C) • Factory default = 115,200 baud (0x1C200) • Maximum = 2,000,000 baud (0x1E8480)
uint8	autobaud	A	Auto-detect UART baud rate at boot: • 0 = Disabled (factory default, must always be disabled in current version)
uint8	autocorrect	C	Auto-correct UART clock to compensate for wide temperature variation: • 0 = Disabled (factory default, must always be disabled in current version)
uint8	flow	F	UART RTS/CTS flow control: • 0 = Disabled (factory default) • 1 = Enabled
uint8	databits	D	UART data bits: • 7 = 7 data bits • 8 = 8 data bits (factory default, must always be 8 data bits in current version) • 9 = 9 data bits
uint8	parity	P	UART parity: • 0 = Disabled (factory default) • 1 = Odd parity • 2 = Even parity
uint8	stopbits	S	UART stop bits: • 1 = 1 stop bit (factory default) • 2 = 2 stop bits

Response Parameters:

None.

Related Commands:

- system_set_transport (ST, ID=2/23)
- system_get_uart_parameters (GTU, ID=2/26)

Example Usage:

- See Changing Serial Communication Parameters

7.2.2.19 system_get_uart_parameters (GTU, ID=2/26)

Obtains the current UART settings for host communication.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`02`	`1A`	None.
RSP	`C0`	`0C`	`02`	`1A`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GTU`	`0x0032`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint32`	`baud`	`B`	UART baud rate: • Minimum = 300 baud (0x12C) • Factory default = 115,200 baud (0x1C200) • Maximum = 2,000,000 baud (0x1E8480)
`uint8`	`autobaud`	`A`	Auto-detect UART baud rate at boot: • 0 = Disabled (factory default, must always be disabled in current version)
`uint8`	`autocorrect`	`C`	Auto-correct UART clock to compensate for wide temperature variation: • 0 = Disabled (factory default, must always be disabled in current version)
`uint8`	`flow`	`F`	UART RTS/CTS flow control: • 0 = Disabled (factory default) • 1 = Enabled
`uint8`	`databits`	`D`	UART data bits: • 7 = 7 data bits • 8 = 8 data bits (factory default, must always be 8 data bits in current version) • 9 = 9 data bits
`uint8`	`parity`	`P`	UART parity: • 0 = Disabled (factory default) • 1 = Odd parity • 2 = Even parity
`uint8`	`stopbits`	`S`	UART stop bits: • 1 = 1 stop bit (factory default) • 2 = 2 stop bits

Related Commands:

- system_get_transport (GT, ID=2/24)
- system_set_uart_parameters (STU, ID=2/25)

7.2.3 OTA Group (ID=3)

OTA methods relate to the firmware update process, using over-the-air GATT-based firmware transfer.

There are no commands within the device firmware upgrade (DFU) group.

Events within this group are documented in OTA Group (ID=3).

7.2.4 GAP Group (ID=4)

GAP methods relate to the Generic Access Protocol layer of the Bluetooth stack, which includes management of scanning and advertising, connection establishment, and connection maintenance.

Commands within the GAP group are listed below:

- gap_connect (/C, ID=4/1)
- gap_cancel_connection (/CX, ID=4/2)
- gap_update_conn_parameters (/UCP, ID=4/3)
- gap_disconnect (/DIS, ID=4/5)
- gap_add_whitelist_entry (/WLA, ID=4/6)
- gap_delete_whitelist_entry (/WLD, ID=4/7)
- gap_start_adv (/A, ID=4/8)
- gap_stop_adv (/AX, ID=4/9)
- gap_start_scan (/S, ID=4/10)
- gap_stop_scan (/SX, ID=4/11)
- gap_query_peer_address (/QPA, ID=4/12)
- gap_query_rssi (/QSS, ID=4/13)
- gap_query_whitelist (/QWL, ID=4/14)
- gap_set_device_name (SDN, ID=4/15)
- gap_get_device_name (GDN, ID=4/16)
- gap_set_device_appearance (SDA, ID=4/17)
- gap_get_device_appearance (GDA, ID=4/18)
- gap_set_adv_data (SAD, ID=4/19)
- gap_get_adv_data (GAD, ID=4/20)
- gap_set_sr_data (SSRD, ID=4/21)
- gap_get_sr_data (GSRD, ID=4/22)
- gap_set_adv_parameters (SAP, ID=4/23)
- gap_get_adv_parameters (GAP, ID=4/24)
- gap_set_scan_parameters (SSP, ID=4/25)
- gap_get_scan_parameters (GSP, ID=4/26)
- gap_set_conn_parameters (SCP, ID=4/27)
- gap_get_conn_parameters (GCP, ID=4/28)
- gap_configure_mtu (/GCMTU, ID=4/29)

Events within this group are documented in GAP Group (ID=4).

7.2.4.1 gap_connect (/C, ID=4/1)

Initiates a connection to a remote device.

For this command to succeed, EZ-Serial must not have other ongoing BLE activity. In other words:

- The module must not be advertising. Use gap_stop_adv (/AX, ID=4/9) to stop, if necessary.
- The module must not be scanning. Use gap_stop_scan (/SX, ID=4/11) to stop, if necessary.
- The module must not be already connected. Use gap_disconnect (/DIS, ID=4/5) to disconnect, if necessary.

After starting the connection process, the module will begin scanning for a connectable advertisement packet from the target device. This will continue until it succeeds, until the connection attempt is canceled with the gap_cancel_connection (/CX, ID=4/2) API command, the connection scan timeout period expires (if it has been set), or the connection scan is successful but fails to establish the link session (such caused by the invalid link key, then the gap_disconnected (DIS, ID=4/6) event will occur without the gap_connected (C, ID=4/5) event occurring first).

When sending this command in text mode, all omitted arguments except `address` and `type` will default to the values set using the gap_set_conn_parameters (SCP, ID=4/27) API command.

Note: If `scan_timeout` is set to zero, the connection attempt will persist forever until it succeeds or it is cancelled intentionally. The `supervision_timeout` parameter governs the link loss detection after a connection is established, and does not affect the connection attempt itself.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`13`	`04`	`01`	None.
RSP	`C0`	`03`	`04`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/C`	`0x000D`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`macaddr`	`address`	`A`	Target connection address: • Set all 0x00 bytes to use directed connection for whitelisted devices
`uint8`	`type`	`T`	Address type: • 0 = Public • 1 = Random/private
`uint16`	`interval`	`I`	Connection interval (1.25 ms units): • Minimum = 0x0006 (6 * 1.25 ms = 7.5 ms) • Maximum = 0x0C80 (3200 * 1.25 ms = 4 seconds)
`uint16`	`slave_latency`	`L`	Slave latency (connection interval count): • Minimum = 0, no intervals skipped • Maximum depends on interval and supervision timeout, such that: `[interval * slave_latency] < supervision_timeout`
`uint16`	`supervision_timeout`	`O`	Supervision timeout (10 ms units): • Minimum = 0x000A (10 * 10 ms = 100 ms) • Maximum = 0x01F4 (500 * 10 ms = 5 seconds)
`uint16`	`scan_interval`	`V`	Connection scan interval (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms)
`uint16`	`scan_window`	`W`	Connection scan window (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than `scan_interval`
`uint16`	`scan_timeout`	`M`	Connection scan timeout (seconds): • 0 to disable

Response Parameters:

None.

Related Commands:

- gap_disconnect (/DIS, ID=4/5)

Related Events:

- gap_connected (C, ID=4/5) – Occurs when an outgoing connection attempt succeeds
- gap_disconnected (DIS, ID=4/6) – Occurs when the remote device scanned, but failed to setup the link session. At this time, the gap_connected event won't occur.

Example Usage:

- See Connecting to a Peripheral Device

7.2.4.2 gap_cancel_connection (/CX, ID=4/2)

Cancels a pending connection attempt.

Use this command to manually end a pending connection attempt to a remote peer device which you previously initiated with the gap_connect (/C, ID=4/1) API command. This command takes no parameters because it is not possible to have more than one pending outgoing connection attempt at a time.

Note: This command applies only when ending a connection attempt that has not succeeded yet. To close an established connection, use the gap_disconnect (/DIS, ID=4/5) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`02`	None.
RSP	`C0`	`02`	`04`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/CX`	`0x0009`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- gap_connect (/C, ID=4/1)
- gap_disconnect (/DIS, ID=4/5)

Related Events:

- gap_connected (C, ID=4/5)

Example Usage:

- See Cancelling Pending Connection to a Peripheral Device

7.2.4.3 gap_update_conn_parameters (/UCP, ID=4/3)

Requests a connection parameter update for an active connection.

Use this command to change the connection interval, slave latency, and supervision timeout for an active connection. If the parameter update is successful, EZ-Serial will generate the gap_connection_updated (CU, ID=4/8) API event after applying new parameters. This will only occur if one or more of the parameters change from its previous value.

The behavior of this command should be the same as the following:

- New connection parameters will always be applied.
- Remote peer (slave) will generate gap_connection_updated (CU, ID=4/8) event if running EZ-Serial.
- Local device will generate gap_connection_updated (CU, ID=4/8) event after new parameter application.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	07	04	03	None.
RSP	C0	02	04	03	None.

Text Info:

Text Name	Response Length	Category	Notes
/UCP	0x000A	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Handle of connection to update
uint16	interval	I*	Connection interval
uint16	slave_latency	L*	Slave latency
uint16	supervision_timeout	O*	Supervision timeout

Response Parameters:

None.

Related Commands:

- gap_connect (/C, ID=4/1)

Related Events:

- gap_connection_update_requested (UCR, ID=4/7)
- gap_connection_updated (CU, ID=4/8)

7.2.4.4 gap_disconnect (/DIS, ID=4/5)

Closes an open connection to a remote device.

Use this command to cleanly close an established connection with a remote peer device. The connection must have been fully opened, indicated by the gap_connected (C, ID=4/5) API event.

Note: This command only applies when closing a connection that is fully open. To cancel a pending connection attempt, use the gap_cancel_connection (/CX, ID=4/2) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01	04	05	None.
RSP	C0	02	04	05	None.

Text Info:

Text Name	Response Length	Category	Notes
/DIS	0x000A	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Handle of connection to be disconnected

Response Parameters:

None.

Related Commands:

- gap_connect (/C, ID=4/1)
- gap_cancel_connection (/CX, ID=4/2)

Related Events:

- gap_disconnected (DIS, ID=4/6)

7.2.4.5 gap_add_whitelist_entry (/WLA, ID=4/6)

Adds a new Bluetooth address to the whitelist.

The whitelist is an optional filter for determining the remote peers that can connect. When whitelist filtering is active, any device which is not on the whitelist will not be allowed to connect to the module. You can control the whitelist filter usage during advertising, scanning, or outgoing connect attempts.

Note: You can only use this command while disconnected. Changes to the whitelist are not allowed during a connection.

Each whitelist entry is made up of two parts: the peer's Bluetooth address and the type of address (public or private). You must specify the correct address type for each peer based on the type of address it is using. This information is available in scan results and connection details.

Note: The BLE stack in EZ-Serial automatically mirrors the bonded device list into the whitelist. This behavior accommodates the most common use case for the whitelist, and manual additions or removals from the whitelist may not be required.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`07`	`04`	`06`	None.
RSP	`C0`	`03`	`04`	`06`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/WLA`	`0x000F`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`macaddr`	`address`	`A`*	Bluetooth address
`uint8`	`type`	`T`	Address type: • 0 = Public (default) • 1 = Random/private

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`count`	`C`	Updated whitelist entry count

Command-Specific Result Codes:

None.

Related Commands:

- gap_connect (/C, ID=4/1) – Connect to any whitelisted device by setting target address to all 0x00 bytes
- gap_delete_whitelist_entry (/WLD, ID=4/7)
- gap_query_peer_address (/QPA, ID=4/12)
- gap_set_adv_parameters (SAP, ID=4/23) – Configure whitelist filter for advertising
- gap_set_scan_parameters (SSP, ID=4/25) – Configure whitelist filter for scanning

Related Events:

- gap_scan_result (S, ID=4/4) – Contains Bluetooth address and type details prior to connecting
- gap_connected (C, ID=4/5) – Contains Bluetooth address and type details after connecting

7.2.4.6 gap_delete_whitelist_entry (/WLD, ID=4/7)

Removes a Bluetooth address from the whitelist.

Use this command to remove a specific device from the whitelist if it is already present. Specify all 0x00 bytes for the address or leave the argument off in text mode to remove all entries from the whitelist. For details on whitelist behavior, see the gap_add_whitelist_entry (/WLA, ID=4/6) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`07`	`04`	`07`	None.
RSP	`C0`	`03`	`04`	`07`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/WLD`	`0x000F`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`macaddr`	`address`	`A`	Bluetooth address
`uint8`	`type`	`T`	Address type: • 0 = Public (default) • 1 = Random/private

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`count`	`C`	Updated whitelist entry count

Related Commands:

- gap_add_whitelist_entry (/WLA, ID=4/6)

7.2.4.7 gap_start_adv (/A, ID=4/8)

Starts advertising.

This command begins advertising using the specified parameters, or using the pre-configured default advertising parameters if in text mode and some arguments are omitted. EZ-Serial must not already be advertising for this command to succeed. However, it is possible to advertise and scan simultaneously.

If you have enabled beaconing (iBeacon or Eddystone) with the p_ibeacon_set_parameters (.IBSP, ID=12/1) API command or the p_eddystone_set_parameters (.EDDYSP, ID=13/1) API command, EZ-Serial will automatically rotate between enabled advertisement payloads with one change per second. If you start advertising using this command and have iBeacon and Eddystone beaconing enabled, it will take three seconds to rotate through all advertisement payloads, with each payload active for one second.

EZ-Serial will generate the gap_adv_state_changed (ASC, ID=4/2) API event when the advertising state changes.

Note: You can start advertising while connected, only if you specify "0" (broadcast-only) for the `mode` argument. The BLE stack does not support being connected and connectable at the same time.

Note: When using the "scannable, undirected" type or "non-connectable, undirected" setting for the `type` argument, the advertisement interval must be 100 ms (0xA0) or greater, per the Bluetooth specification. Intervals shorter than this will result in an error response.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`08`	`04`	`08`	None.
RSP	`C0`	`02`	`04`	`08`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/A`	`0x0008`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`adv_mode`	`M`	Advertisement mode: • 1 = Directed advertisement (high duty cycle, reserved for future usage, not support in current release) • 2 = Directed advertisement (low duty cycle, reserved for future usage, not support in current release) • 3 = Undirected advertisement (high duty cycle) • 4 = Undirected advertisement (low duty cycle) • 5 = Non-connectable advertisement (high duty cycle) • 6 = Non-connectable advertisement (low duty cycle) • 7 = Discoverable advertisement (scannable, high duty cycle) • 8 = Discoverable advertisement (scannable, low duty cycle) Note that when BLE connection is established, the value of adv_mode will be ignored and the value of bit4-7 of flags of GATTS parameters will be used. See gatts_set_parameters (SGSP, ID=5/14) for more details. When the directed advertisement type is used, the directed advertisement address and type should be set correctly first through the gap_set_adv_parameters (SAP, ID=4/23) command.
`uint16`	`interval`	`I`	Advertisement interval (625 µs units): • Minimum = 0x0020 (32 * 0.625 ms = 20 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.

Data Type	Name	Text	Description
`uint8`	`channels`	`C`	Advertisement channel selection bitmask (at least one bit must be set): • Bit 0 (0x1) = Channel 37 • Bit 1 (0x2) = Channel 38 • Bit 2 (0x4) = Channel 39 Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`filter`	`F`	Advertisement filter policy: • 0 = Process scan and connection requests from all devices (factory default, must always be set to this value in the current version, other values are not supported yet) • 1 = Process connection requests from all devices and only scan requests from devices that are in the whitelist. • 2 = Process scan requests from all devices and only connection requests from devices that are in the whitelist. • 3 = Process scan and connection requests only from devices in the whitelist. Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint16`	`timeout`	`O`	Advertisement timeout (seconds): • 0 for infinite (factory default)

Response Parameters:

None.

Related Commands:

- gap_stop_adv (/AX, ID=4/9)
- gap_set_adv_data (SAD, ID=4/19)
- gap_set_sr_data (SSRD, ID=4/21)
- gap_set_adv_parameters (SAP, ID=4/23)

Related Events:

- gap_adv_state_changed (ASC, ID=4/2)

Example Usage:

- See Advertising as Peripheral Device

7.2.4.8 gap_stop_adv (/AX, ID=4/9)

Stops advertising.

This command immediately stops advertising if it is currently active. Note that advertising may have started because of the gap_start_adv (/A, ID=4/8) API command, or due to specific configuration settings (GAP parameters, CYSPP profile, iBeacon, or Eddystone) that automatically began advertising.

EZ-Serial will generate the gap_adv_state_changed (ASC, ID=4/2) API event when the advertising state changes.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`09`	None.
RSP	`C0`	`02`	`04`	`09`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/AX`	`0x0009`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)

Related Events:

- gap_adv_state_changed (ASC, ID=4/2)

7.2.4.9 gap_start_scan (/S, ID=4/10)

Starts scanning.

This command begins scanning using the specified parameters, or using the pre-configured default scan parameters if in text mode and some arguments are omitted. EZ-Serial must not already be scanning for this command to succeed. However, it is possible to advertise and scan simultaneously.

EZ-Serial will generate the mgap_scan_state_changed (SSC, ID=4/3) API event when the scanning state changes.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`0A`	`04`	`0A`	None.
RSP	`C0`	`02`	`04`	`0A`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/S`	`0x0008`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	Discovery mode: • 0 = Not scanning • 1 = High duty cycle scan • 2 = Low duty cycle scan
`uint16`	`interval`	`I`	Scan interval (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint16`	`window`	`W`	Scan window (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than `interval` Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`active`	`A`	Active scanning: • 0 = Passive scanning • 1 = Active scanning Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`filter`	`F`	Whitelist filter policy**:** • 0 = Accept all advertising packets from all devices • 1 = Accept advertising packets only from whitelisted devices • 2 = Accept advertising packets only from devices sending directed advertisements to this device • 3 = Accept advertising packets only from whitelisted devices sending directed advertisements to this device

Data Type	Name	Text	Description
			Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`nodupe`	`D`	Duplicate filter policy: • 0 = Disable duplicate result filtering • 1 = Enable duplicate result filtering
`uint16`	`timeout`	`O`	Scan timeout (seconds): • 0 for infinite

Response Parameters:

None.

Related Commands:

- gap_stop_scan (/SX, ID=4/11)
- gap_set_scan_parameters (SSP, ID=4/25)

Related Events:

- mgap_scan_state_changed (SSC, ID=4/3)
- gap_scan_result (S, ID=4/4)

7.2.4.10 gap_stop_scan (/SX, ID=4/11)

Stops scanning.

This command immediately stops scanning if it is currently active. Note that advertising may have started because of the gap_start_scan (/S, ID=4/10) API command, or due to specific configuration settings (particularly the CYSPP profile settings if the central role is enabled).

EZ-Serial will generate the mgap_scan_state_changed (SSC, ID=4/3) API event when the scanning state changes.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`0B`	None.
RSP	`C0`	`02`	`04`	`0B`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/SX`	`0x0009`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- gap_start_scan (/S, ID=4/10)

Related Events:

- mgap_scan_state_changed (SSC, ID=4/3)

7.2.4.11 gap_query_peer_address (/QPA, ID=4/12)

Queries remote peer Bluetooth address.

This command returns the Bluetooth address of the currently connected remote peer device. An active connection is required to use this command successfully.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`04`	`0C`	None.
RSP	`C0`	`09`	`04`	`0C`	None.

Text Info:

Text Name	Response Length	Notes
`/QPA`	`0x001E`	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle for which the remote peer address is queried (Ignored in current release due to internal BLE stack functionality, set to 0)

Response Parameters:

Data Type	Name	Text	Description
`macaddr`	`address`	`A`	Peer Bluetooth address
`uint8`	`address_type`	`T`	Address type

Related Commands:

- gap_connect (/C, ID=4/1)
- gap_query_rssi (/QSS, ID=4/13)

7.2.4.12 gap_query_rssi (/QSS, ID=4/13)

This command returns the RSSI value detected in the packet received most recently from the currently connected remote peer device. An active connection is required to use this command successfully.

Note: RSSI values in real-world environment often fall in the -50 dBm to -70 dBm range. An RSSI value at this level does not necessarily indicate a poor connection.

The RSSI value reported in the consecutive gap_rssi_result (RSSI, ID=4/9) event is expressed as a signed 8-bit integer. In text mode, it will appear in two's complement form. Positive numbers in this form fall in the range [0, 127] and are as they appear. Negative numbers fall in the range [128, 255] and should have 256 subtracted from them to obtain the real value.

Examples:

- 0x03 = +3 dBm
- 0xFF = -1 dBm (0xFF = 255 - 256 = -1)
- 0xF0 = -16 dBm (0xF0 = 240 - 256 = -16)
- 0xC5 = -59 dBm (0xC5 = 197 - 256 = -59)

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`04`	`0D`	None.
RSP	`C0`	`03`	`04`	`0D`	None.

Text Info:

Text Name	Response Length	Notes
`/QSS`	`0x000F`	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle for which the signal strength is queried (Ignored in current release due to internal BLE stack functionality, set to 0)

Response Parameters:

None

Related Events:

- gap_rssi_result (RSSI, ID=4/9)

7.2.4.13 gap_query_whitelist (/QWL, ID=4/14)

Requests a list of whitelisted devices.

This command provides access to the current whitelist. The response from this command includes the number of devices on the whitelist, and the response will be followed by that many gap_whitelist_entry (WL, ID=4/1) API events which provide details for each entry.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`0E`	None.
RSP	`C0`	`03`	`04`	`0E`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/QWL`	`0x000F`	ACTION	None.

Command Arguments:
None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`count`	`C`	Whitelist entry count

Related Commands:

- gap_add_whitelist_entry (/WLA, ID=4/6)
- gap_delete_whitelist_entry (/WLD, ID=4/7)

Related Events:

- gap_whitelist_entry (WL, ID=4/1)

7.2.4.14 gap_set_device_name (SDN, ID=4/15)

Configures a new device name.

This is typically a UTF-8 string value that is stored in the Device Name characteristic (UUID 0x2A00) in the local GATT structure. This characteristic is part of the GAP service (UUID 0x1800). The GAP service is mandatory for all Bluetooth Smart devices, and the Device Name characteristic is a mandatory part of the GAP service.

Using this command affects the value in the local GATT server Device Name characteristic, and the local name field in the automatically managed scan response packed used for advertising.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01-41	04	0F	Variable-length command payload, minimum of 1 (0x01), maximum of 65 (0x41)
RSP	C0	02	04	0F	None.

Text Info:

Text Name	Response Length	Category	Notes
SDN	0x0009	SET	None.

Command Arguments:

Data Type	Name	Text	Description
string	name	N	New device name (0-64 bytes, raw ASCII data when in text mode) See Changing Device Name and Appearance to set the device name with partial MAC address.

Response Parameters:

None.

Related Commands:

- gap_get_device_name (GDN, ID=4/16)

Example Usage:

- See Changing Device Name and Appearance

7.2.4.15 gap_get_device_name (GDN, ID=4/16)

Obtains the current device name.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	04	10	None.
RSP	C0	03-43	04	10	Variable-length response payload, minimum of 3 (0x03), maximum of 67 (0x43)

Text Info:

Text Name	Response Length	Category	Notes
GDN	0x000C-0x004C	GET	Variable-length response payload, minimum of 12 (0x0C), maximum of 76 (0x4C)

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
string	name	N	Current device name (0-64 bytes, raw ASCII data when in text mode)

Related Commands:

- gap_set_device_name (SDN, ID=4/15)

7.2.4.16 gap_set_device_appearance (SDA, ID=4/17)

Sets new appearance for current device.

Defines the device appearance value. This is a 16-bit value which is stored in the Appearance characteristic (UUID 0x2A01) in the local GATT structure. This characteristic is part of the GAP service (UUID 0x1800). The GAP service is mandatory for every Bluetooth Smart device, and the Appearance characteristic is a mandatory part of the GAP service.

Using this command affects the value in the local GATT server Device Appearance characteristic.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	02	04	11	None.
RSP	C0	02	04	11	None.

Text Info:

Text Name	Response Length	Category	Notes
SDA	0x0009	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint16	appearance	A	New device appearance value (factory default is 0x0000)

Response Parameters:

None.

Related Commands:

- gap_get_device_appearance (GDA, ID=4/18)

7.2.4.17 gap_get_device_appearance (GDA, ID=4/18)

Obtains the current device appearance value.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	04	12	None.
RSP	C0	04	04	12	None.

Text Info:

Text Name	Response Length	Category	Notes
GDA	0x0010	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint16	appearance	A	Current device appearance value

Related Commands:

- gap_set_device_appearance (SDA, ID=4/17)

7.2.4.18 gap_set_adv_data (SAD, ID=4/19)

Configures a new custom advertisement packet data.

Defines a new byte sequence for the primary advertisement packet data payload. This content will be visible to all scanning devices performing a passive or active scan when the CYBT-4130XX-02 EZ-BT module is in an advertising state.

Note: EZ-Serial automatically manages advertisement content unless you enable the use of user-defined data with the gap_set_adv_parameters (SAP, ID=4/23) API command. If you only set custom data, but do not enable user-defined content, the data here will remain unused.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01-20`	`04`	`13`	Variable-length command payload, minimum of 1 (0x01), maximum of 32 (0x20)
RSP	`C0`	`02`	`04`	`13`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SAD`	`0x0009`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8a`	`data`	`D`	New advertisement payload data (0-31 bytes) **Note:** `uint8a` data type requires one prefixed “length” byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_get_adv_data (GAD, ID=4/20)
- gap_set_sr_data (SSRD, ID=4/21)
- gap_set_adv_parameters (SAP, ID=4/23)

Example Usage:

- See Customizing Advertisement and Scan Response Data

7.2.4.19 gap_get_adv_data (GAD, ID=4/20)

Obtains the current custom advertisement packet data.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`14`	None.
RSP	`C0`	`03-22`	`04`	`14`	Variable-length response payload, minimum of 3 (0x03), maximum of 34 (0x22)

Text Info:

Text Name	Response Length	Category	Notes
`GAD`	`0x000D-0x004B`	GET	Variable-length response payload, minimum of 13 (0x0D), maximum of 75 (0x4B)

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8a`	`data`	`D`	Current advertisement payload data (0-31 bytes) **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Related Commands:

- gap_set_adv_data (SAD, ID=4/19)

7.2.4.20 gap_set_sr_data (SSRD, ID=4/21)

Configures new custom scan response packet payload.

This command defines a new byte sequence for the scan response packet. This content will be visible to all scanning devices performing an active scan when the CYBT-4130XX-02 EZ-BT module is in a scannable advertising state.

Note: EZ-Serial automatically manages scan response content unless you enable the use of user-defined data with the gap_set_adv_parameters (SAP, ID=4/23) API command. If you only set custom data, but do not enable user-defined content, the data here will remain unused.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01-20`	`04`	`15`	Variable-length command payload, minimum of 1 (0x01), maximum of 32 (0x20)
RSP	`C0`	`02`	`04`	`15`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SSRD`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8a`	`data`	`D`	New scan response payload data (0-31 bytes) **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_set_adv_data (SAD, ID=4/19)
- gap_get_sr_data (GSRD, ID=4/22)
- gap_set_adv_parameters (SAP, ID=4/23)

Example Usage:

- See Customizing Advertisement and Scan Response Data

7.2.4.21 gap_get_sr_data (GSRD, ID=4/22)

Obtains the current custom scan response packet data.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`16`	None.
RSP	`C0`	`03-22`	`04`	`16`	Variable-length response payload, minimum of 3 (0x03), maximum of 34 (0x22)

Text Info:

Text Name	Response Length	Category	Notes
`GSRD`	`0x000D-0x004B`	GET	Variable-length response payload, minimum of 13 (0xD), maximum of 75 (0x4B)

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8a`	`data`	`D`	Current scan response payload data (0-31 bytes) **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Related Commands:

- gap_set_sr_data (SSRD, ID=4/21)

7.2.4.22 gap_set_adv_parameters (SAP, ID=4/23)

Configures new default advertisement parameters.

These parameters will be used when sending the gap_start_adv (/A, ID=4/8) API command in text mode without specifying non-default arguments.

Note: Setting Bit 0 (0x01) of the flags value using this command will enable automatic advertisement on boot, as described. However, advertisements may automatically start even if this bit is cleared, and if the enable setting of CYSPP, iBeacon, or Eddystone is set to the "enable + autostart" setting. Factory default settings include this value for the CYSPP feature.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`09`	`04`	`17`	None.
RSP	`C0`	`02`	`04`	`17`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SAP`	`0x0009`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`adv_mode`	`M`	Advertisement mode: • 1 = Directed advertisement (high duty cycle, resereved for future usage, not support in current release) • 2 = Directed advertisement (low duty cycle, resereved for future usage, not support in current release) • 3 = Undirected advertisement (high duty cycle) • 4 = Undirected advertisement (low duty cycle) • 5 = Non-connectable advertisement (high duty cycle) • 6 = Non-connectable advertisement (low duty cycle) • 7 = Discoverable advertisement (scannable, high duty cycle) • 8 = Discoverable advertisement (scannable, low duty cycle) Note that when BLE connection has established, the value of adv_mode will be ignored and the value of bit4-7 of flags of GATTS parameters will be used. See gatts_set_parameters (SGSP, ID=5/14) for more details. When the directed advertisement type is used, the directed advertisement address and type should be set correctly first through the gap_set_adv_parameters (SAP, ID=4/23) command.

Data Type	Name	Text	Description
`uint16`	`interval`	`I`	Advertisement interval (625 µs units): • Minimum = 0x0020 (32 * 0.625 ms = 20 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0030 (48 * 0.625 ms = 30 ms) Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`channels`	`C`	Advertisement channel selection bitmask: • Bit 0 (0x1) = Channel 37 • Bit 1 (0x2) = Channel 38 • Bit 2 (0x4) = Channel 39 **Note**: At least one bit must be set, factory default is all 0x07 (all bits set) Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`filter`	`L`	Advertisement filter policy: • 0 = Process scan and connection requests from all devices (factory default, must always be set to this value in current version, other values are not supported yet) • 1 = Process connection requests from all devices and only scan requests from devices that are in the whitelist. • 2 = Process scan requests from all devices and only connection requests from devices that are in the whitelist. • 3 = Process scan and connection requests only from devices in the whitelist. Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint16`	`timeout`	`O`	Advertisement timeout (seconds): • 0 for infinite (factory default)
`uint8`	`flags`	`F`	Advertisement behavior flags bitmask: • Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection • Bit 1 (0x2) = Use user set scan response data. When this bit is set, the user set scan response data through the gap_set_sr_data (SSRD, ID=4/21) command will be used; When this bit is cleared, the default scan response data will be used. **Note**: Factory default = 0x01 (enable automatic advertising mode upon boot)
`macaddr`	`address`	`A`	Directed advertisement Bluetooth Address: • Set all 0x00 bytes indicates using undirected method.
`uint8`	`type`	`T`	Directed advertisement Bluetooth Address type: • 0 = Public • 1 = Random/private

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_get_adv_parameters (GAP, ID=4/24)

7.2.4.23 gap_get_adv_parameters (GAP, ID=4/24)

Obtains the current advertisement parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`18`	None.
RSP	`C0`	`0B`	`04`	`18`	None.

Text Info:

Text Name	Response Length	Category	Notes
GAP	0x0030	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8	adv_mode	M	Advertisement mode: • 1 = Directed advertisement (high duty cycle, reserved for future usage, not support in current release) • 2 = Directed advertisement (low duty cycle, reserved for future usage, not support in current release) • 3 = Undirected advertisement (high duty cycle) • 4 = Undirected advertisement (low duty cycle) • 5 = Non-connectable advertisement (high duty cycle) • 6 = Non-connectable advertisement (low duty cycle) • 7 = Discoverable advertisement (scannable, high duty cycle) • 8 = Discoverable advertisement (scannable, low duty cycle) Note that when BLE connection has established, the value of adv_mode will be ignored and the value of bit4-7 of flags of GATTS parameters will be used. See gatts_set_parameters (SGSP, ID=5/14) for more details. When the directed advertisement type is used, the directed advertisement address and type should be set correctly first through the gap_set_adv_parameters (SAP, ID=4/23) command.
uint16	interval	I	Advertisement interval (625 µs units): • Minimum = 0x0020 (32 * 0.625 ms = 20 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0030 (48 * 0.625 ms = 30 ms) Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
uint8	channels	C	Advertisement channel selection bitmask: • Bit 0 (0x1) = Channel 37 • Bit 1 (0x2) = Channel 38 • Bit 2 (0x4) = Channel 39 **Note**: At least one bit must be set; factory default is all 0x07 (all bits set). Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
uint8	filter	L	Advertisement filter policy: • 0 = Process scan and connection requests from all devices (factory default, must always be set to this value in the current version, other values are not supported yet) • 1 = Process connection requests from all devices and only scan requests from devices that are in the whitelist. • 2 = Process scan requests from all devices and only connection requests from devices that are in the whitelist. • 3 = Process scan and connection requests only from devices in the whitelist. Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
uint16	timeout	O	Advertisement timeout (seconds): • 0 for infinite (factory default)
uint8	flags	F	Advertisement behavior flags bitmask: • Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection • Bit 1 (0x2) = Use user set scan response data. When this bit is set, you can set scan response data through gap_set_sr_data (SSRD, ID=4/21) command will be used; When this bit is cleared, the default scan response data will be used. **Note:** Factory default = 0x01 (enable automatic advertising mode upon boot)
macaddr	address	A	Directed advertisement Bluetooth Address: • Set all 0x00 or 0xFF bytes indicates invalid address, so the undirected advertising will be used instead.

Data Type	Name	Text	Description
`uint8`	`type`	`T`	Directed advertisement Bluetooth Address type: • 0 = Public • 1 = Random/private

Related Commands:

- gap_set_adv_parameters (SAP, ID=4/23)

7.2.4.24 gap_set_scan_parameters (SSP, ID=4/25)

Configures new default scan parameters.

These parameters will be used when sending the gap_start_scan (/S, ID=4/10) API command in text mode without specifying non-default arguments.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`0A`	`04`	`19`	None.
RSP	`C0`	`02`	`04`	`19`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SSP`	`0x0009`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	Discovery mode: • 0 = Observation mode • 1 = Limited discovery mode • 2 = General discovery mode (factory default)
`uint16`	`interval`	`I`	Scan interval (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint16`	`window`	`W`	Scan window (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 secs) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than `interval` Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`active`	`A`	Active scanning: • 0 = Passive scanning • 1 = Active scanning Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`filter`	`F`	Whitelist filter policy**:** • 0 = Accept advertising packet from all devices, directed advertising packet not directed to local device is ignored • 1 = Accept advertising packet from device in whitelist, directed advertising packet not directed to local device is ignored • 2 = Accept advertising packet from all, directed advertising packet not directed to local device is ignored except direct advertising with RPA • 3 = Accept advertising packet from device in white list, directed advertising packet not directed to me is ignored except direct adv with RPA

Data Type	Name	Text	Description
			Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`nodupe`	`D`	Duplicate filter policy: • 0 = Disable duplicate result filtering • 1 = Enable duplicate result filtering (factory default)
`uint16`	`timeout`	`O`	Scan timeout (seconds): • 0 for infinite (factory default)

Response Parameters:

None.

Related Commands:

- gap_start_scan (/S, ID=4/10)
- gap_get_scan_parameters (GSP, ID=4/26)

7.2.4.25 gap_get_scan_parameters (GSP, ID=4/26)

Obtains the current scan parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`1A`	None.
RSP	`C0`	`0C`	`04`	`1A`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GSP`	`0x0032`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	Discovery mode: • 0 = Observation mode • 1 = Limited discovery mode • 2 = General discovery mode (factory default)
`uint16`	`interval`	`I`	Scan interval (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint16`	`window`	`W`	Scan window (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than `interval` Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`active`	`A`	Active scanning: • 0 = Passive scanning • 1 = Active scanning Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.

Data Type	Name	Text	Description
`uint8`	`filter`	`F`	Whitelist filter policy: • 0 = Accept advertising packet from all devices, directed advertising packet not directed to local device is ignored • 1 = Accept advertising packet from device in white list, directed advertising packet not directed to local device is ignored • 2 = Accept adv packet from all, directed advertising packet not directed to local device is ignored except direct advertising with RPA • 3 = Accept adv packet from device in whitelist, directed advertising packet not directed to me is ignored except direct advertising with RPA Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.
`uint8`	`nodupe`	`D`	Duplicate filter policy: • 0 = Disable duplicate result filtering • 1 = Enable duplicate result filtering (factory default)
`uint16`	`timeout`	`O`	Scan timeout (seconds): • 0 for infinite (factory default)

Related Commands:

- gap_set_scan_parameters (SSP, ID=4/25)

7.2.4.26 gap_set_conn_parameters (SCP, ID=4/27)

Configures new default connection parameters.

These parameters will be used when sending the gap_connect (/C, ID=4/1) API command in text mode without specifying non-default arguments.

Note: The values of slave_latency (L), interval (I) and supervision_timeout (O) must follow this rule:

```
(slave_latency + 1) * interval <= (supervision_timeout * 4)
```

The values of scan_interval (V) and scan_window (W) must follow this rule:

```
scan_window <= scan_interval
```

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`0C`	`04`	`1B`	None.
RSP	`C0`	`02`	`04`	`1B`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SCP`	`0x0009`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`interval`	`I`	Connection interval (1.25 ms units): • Minimum = 0x0006 (6 * 1.25 ms = 7.5 ms, factory default) • Maximum = 0x0C80 (3200 * 1.25 ms = 4 seconds)
`uint16`	`slave_latency`	`L`	Slave latency (connection interval count): • Minimum = 0, no intervals skipped (factory default) • Maximum depends on interval and supervision timeout, such that: `[interval * slave_latency] < supervision_timeout`
`uint16`	`supervision_timeout`	`O`	Supervision timeout (10 ms units): • Minimum = 0x000A (10 * 10 ms = 100 ms) • Maximum = 0x01F4 (500 * 10 ms = 5 seconds) • Factory default = 0x064 (100 * 10 ms = 1 second)

Data Type	Name	Text	Description
`uint16`	`scan_interval`	`V`	Connection scan interval (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms)
`uint16`	`scan_window`	`W`	Connection scan window (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) • Cannot be greater than `scan_interval`
`uint16`	`scan_timeout`	`M`	Connection scan timeout (seconds): • 0 for infinite (factory default)

Response Parameters:

None.

Related Commands:

- gap_connect (/C, ID=4/1)
- gap_update_conn_parameters (/UCP, ID=4/3)
- gap_get_conn_parameters (GCP, ID=4/28)

7.2.4.27 gap_get_conn_parameters (GCP, ID=4/28)

Gets the current default connection parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`1C`	None.
RSP	`C0`	`0E`	`04`	`1C`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GCP`	`0x0033`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`interval`	`I`	Connection interval (1.25 ms units): • Minimum = 0x0006 (6 * 1.25 ms = 7.5 ms, factory default) • Maximum = 0x0C80 (3200 * 1.25 ms = 4 seconds)
`uint16`	`slave_latency`	`L`	Slave latency (connection interval count): • Minimum = 0, no intervals skipped (factory default) • Maximum depends on interval and supervision timeout, such that: `[interval * slave_latency] < supervision_timeout`
`uint16`	`supervision_timeout`	`O`	Supervision timeout (10 ms units): • Minimum = 0x000A (10 * 10 ms = 100 ms) • Maximum = 0x01F4 (500 * 10 ms = 5 seconds) • Factory default = 0x064 (100 * 10 ms = 1 second)

Data Type	Name	Text	Description
`uint16`	`scan_interval`	`V`	Connection scan interval (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms)
`uint16`	`scan_window`	`W`	Connection scan window (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) • Cannot be greater than `scan_interval`
`uint16`	`scan_timeout`	`M`	Connection scan timeout (seconds): • 0 for infinite (factory default)

Related Commands:

- gap_set_conn_parameters (SCP, ID=4/27)

7.2.4.28 gap_configure_mtu (/GCMTU, ID=4/29)

Command the GATT client uses to start negotiating the MTU size between the two connected devices. The negotiated MTU size that will be used within the connection is reported through the gap_mtu_updated (MTU, ID=4/10) event.

Note: After a new connection is established through gap_connect (/C, ID=4/1) command, the default MTU size used for the connection is 23 bytes, which means that the data size can be transferred in each packet is 20 bytes. So, to improve the transmission performance, it is highly recommended to reconfigure the MTU size to a maximum of 515 bytes or to the maximum data size that would be transferred by using this command plus 3 bytes.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`04`	`1D`	None.
RSP	`C0`	`03`	`06`	`1D`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/GCMTU`	`0x0002`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C*`	Connection handle required to negotiate with the new MTU size.
`uint16`	`mtu`	`M*`	The maximum MTU size that the GATT client accepts for the connection, and need to negotiate with the peer device. **Note**: The valid MTU value range is 23 ~ 515, so the valid data size should be 20 ~ 512.

Response Parameters:

None.

Related Commands:

None.

Related Events:

- gap_mtu_updated (MTU, ID=4/10) – Reports the negotiated MTU size for the connection.

7.2.5 GATT Server Group (ID=5)

GATT server methods relate to the server role of the Generic Attribute Protocol layer of the Bluetooth stack. These methods are used for working with the local GATT structure.

Commands within this group are listed below:

- gatts_create_attr (/CAC, ID=5/1)
- gatts_delete_attr (/CAD, ID=5/2)
- gatts_validate_db (/VGDB, ID=5/3)
- gatts_store_db (/SGDB, ID=5/4)
- gatts_dump_db (/DGDB, ID=5/5)
- gatts_discover_services (/DLS, ID=5/6)
- gatts_discover_characteristics (/DLC, ID=5/7)
- gatts_discover_descriptors (/DLD, ID=5/8)
- gatts_read_handle (/RLH, ID=5/9)
- gatts_write_handle (/WLH, ID=5/10)
- gatts_notify_handle (/NH, ID=5/11)
- gatts_indicate_handle (/IH, ID=5/12)
- gatts_send_writereq_response (/WRR, ID=5/13)
- gatts_set_parameters (SGSP, ID=5/14)
- gatts_get_parameters (GGSP, ID=5/15)

Events within this group are documented in GATT Server Group (ID=5).

7.2.5.1 gatts_create_attr (/CAC, ID=5/1)

Adds a new custom attribute to the local GATT structure.

The new attribute will be given the next available handle. All handles are assigned sequentially. Attributes must be added in order, and will always be appended to the next available position in the GATT structure.

New attributes must be entered such that the database always has a valid structure, other than possibly being incomplete while adding other required attributes. EZ-Serial will reject new attribute creation attempts which would result in an invalid structure and provide a validity report code from the list in Table 7-5. EZ-Serial System Error Codes

EZ-Serial *GATT Database Validation Error Codes*.

See Defining Custom Local GATT Services and Characteristics and Adopted Bluetooth SIG GATT Profile Structure Snippets for detailed instructions and example usage, including important guidelines for permission settings.

Note: Always configure structural declarations (types 0x2800 and 0x2803) to have unrestricted read permissions (0x01) and no write permissions (0x00) to ensure that clients can properly discover the basic GATT database structure. Special security requirements should only be applied to characteristic value attributes or, in limited cases, related configuration descriptors.

Use the gatts_dump_db (/DGDB, ID=5/5) API command to list the current local GATT database entries in a format similar to what this command requires.

Note: EZ-Serial includes a fixed set of attributes as part of the core functionality, which cannot be deleted or modified. These attributes occupy the handle range from 1 (0x0001) to 28 (0x001C) and 65280 (0xFF00) – 65535 (0xFFFF). Therefore, the first custom attribute created in a factory default state will receive the handle value 29 (0x001D), and the last custom attribute created in a factory default state will receive the handle value 65279 (0xFEFF).

Note: Additions to and removals from the GATT structure are always stored in flash. If the “result” value in the response indicates success, the change will be effective immediately and will persist through power cycles and resets. The internal CPU is occupied for approximately 15 ms during each flash write operation, and during this time no other activity will be processed (UART or BLE communication). Any UART data sent during this brief window will be lost. Therefore, you should only modify the GATT structure while disconnected, and you should allow a gap of at least 20 ms between the end of one API command and the beginning of a new one. If you have enabled hardware flow control using the system_set_uart_parameters (STU, ID=2/25) API command, EZ-Serial will block incoming data flow during flash writes to prevent serial data corruption or loss.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`09`	`05`	`01`	Variable-length command payload, value specified is minimum
RSP	`C0`	`06`	`05`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/CAC`	`0x0018`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`type`	`T*`	Attribute type: • 0 = Structure entry • 1 = Characteristic value entry Structural entries require constant data containing the definition of Structural entries optionally allow additional RAM data beyond the constant length for descriptor value information, such as two-byte CCCD values: • Characteristic value entries do not require any constant data, but may have it if a default boot-time value is desired
`uint8`	`perm`	`R*`	Permission bits: • Bit 0 (0x01) = Variable length • Bit 1 (0x02) = Readable • Bit 2 (0x04) = Write command (unacknowledged) • Bit 3 (0x08) = Write request (acknowledged) • Bit 4 (0x10) = Authenticated readable • Bit 5 (0x20) = Reliable write (includes prepared write) • Bit 6 (0x40) = Authenticated writeable **Note:** For the Characteristic Value item, the setting of the permission bits must be consistent and match with the setting of the Characteristic Properties bits created in the Characteristic Declare item. For example, when the Characteristic Properties is 0x06 (read + write without response), the Characteristic Value permission should be set to 0x06 (readable + write command); Also, the 0x0A (read + write) properties corresponding to 0x0A (readable + write request) permission. **Note:** The Write command and Write request permissions are exclusive with each other. **Note:** The prepared read and write operations are not supported in current release.
`uint16`	`length`	`L*`	Indicates the maximum length in bytes of the attribute value. The length value must be equal to or greater than the bytes of the constant data that is specified in the following data field. When this length field value is greater than the constant data length, firmware will automatically allocate corresponding RAM buffer to a newly created attribute. The default value is initialized to zero.

Data Type	Name	Text	Description
`longuint8a`	`data`	`D`*	Data includes attribute type UUID, characteristic properties byte, default attribute value, or all values as applicable). **Note:** 1) `longuint8a` data type requires two prefixed "length" bytes before binary parameter payload. 2) When creating the attribute, the data should not include the non-constant part, such as the attribute value data for Characteristic value type and Characteristic Descriptor attributes. See gatts_write_handle (/WLH, ID=5/10)0command for details on how to update the attribute value. 3) The value of the non-constant part will be allocated in firmware based on the specified length parameter. The constant and non-constant part data can be read out together using the gatts_read_handle (/RLH, ID=5/9) command. 4) The CCCD attribute must be the first descriptor created after a new GATT Characteristic is created. The UUID of the attribute and the characteristic property values are defined here, which may be used in structure and characteristic entries. **UUID values of attribute types:** • 0x2800 = Primary Service Declaration • 0x2801 = Secondary Service Declaration • 0x2802 = Include Declaration • 0x2803 = Characteristic Declaration • 0x2900 = Characteristic Extended Properties descriptor • 0x2901 = Characteristic User Description descriptor • 0x2902 = Client Characteristic Configuration descriptor • 0x2903 = Server Characteristic Configuration descriptor • 0x2904 = Characteristic Format descriptor • 0x2905 = Characteristic Aggregate Format descriptor **Characteristic properties:** • Bit 0 (0x01) = Broadcast • Bit 1 (0x02) = Read • Bit 2 (0x04) = Write without response (unacknowledged) • Bit 3 (0x08) = Write (acknowledged) • Bit 4 (0x10) = Notify • Bit 5 (0x20) = Indicate • Bit 6 (0x40) = Signed write • Bit 7 (0x80) = Extended properties (requires 0x2900) Characteristic declaration stores the UUID of the Characteristic value attribute. So, the value of this data 'D' field should be: 0x2803 (UUID)+ Characteristic properties (1 byte) + handle of value attribute (2 byte) + UUID of value attribute. **Note:** The characteristic properties must be consistent with and match the characteristic permission value. For more details on the examples, see the `perm` parameter. **Note:** The Write and Write without request properties are exclusive to each other.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`handle`	`H`	New attribute handle (0x0001-0xFEFF)
`uint16`	`valid`	`V`	GATT database validity status. For details on the validity status value, see response value of the gatts_validate_db (/VGDB, ID=5/3) command.

Related Commands:

- gatts_delete_attr (/CAD, ID=5/2)
- gatts_validate_db (/VGDB, ID=5/3)
- gatts_dump_db (/DGDB, ID=5/5)

Related Events:

- gatts_db_entry_blob (DGATT, ID=5/4)

Example Usage:

- See Defining Custom Local GATT Services and Characteristics
- See Adopted Bluetooth SIG GATT Profile Structure Snippets

7.2.5.2 gatts_delete_attr (/CAD, ID=5/2)

Removes one or more attributes from the GATT structure.

If you use this command without a handle in text mode or you supply a handle with value 0 in either text or binary mode, the highest attribute number (most recently added) will be removed. If you supply a non-zero handle, then the attribute with that handle and all higher handles will be removed. See Note.

The handles of all default fixed services, including OTA service, will not be removed in any situation. The handle of the handles will always be the latest handle.

After removing an attribute with this command, the local GATT database may no longer be strictly valid. See Table 7-5. EZ-Serial System Error Codes

EZ-Serial *GATT Database Validation Error Codes* for possible validity states. Use the gatts_dump_db (/DGDB, ID=5/5) API command to list the current local GATT database entries.

Note: EZ-Serial includes a fixed set of attributes as part of the core functionality, which cannot be deleted or modified. These attributes occupy the handle range from 1 (0x0001) to 28 (0x001C), and 65280 (0xFF00) – 65535 (0xFFFF). Therefore, you cannot delete any attribute with a handle value less than 29 (0x001D) or greater than 65279 (0xFEFF).

Note: Additions to and removals from the GATT structure are always stored in flash. If the “result” value in the response indicates success, the change will be effective immediately and will persist through power cycles and resets. The internal CPU is occupied for approximately 15 ms during each flash write operation, and during this time no other activity will be processed (UART or BLE communication). Any UART data sent during this brief window will be lost. Therefore, you should only modify the GATT structure while it is disconnected, and you should allow a gap of at least 20 ms between the end of one API command and the beginning of a new one. If you have enabled hardware flow control using the system_set_uart_parameters (STU, ID=2/25) API command, EZ-Serial will block incoming data flow during flash writes to prevent serial data corruption or loss.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`02`	`05`	`02`	None.
RSP	`C0`	`08`	`05`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/CAD`	`0x001F`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`handle`	`H`	Attribute handle to remove (includes all higher attributes)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`count`	`C`	Number of attributes deleted from GATT structure
`uint16`	`next_handle`	`H`	Next available attribute handle after removal
`uint16`	`valid`	`V`	GATT database validity status

Related Commands:

- gatts_create_attr (/CAC, ID=5/1)
- gatts_validate_db (/VGDB, ID=5/3)
- gatts_dump_db (/DGDB, ID=5/5)

7.2.5.3 *gatts_validate_db (/VGDB, ID=5/3)*

Checks to ensure that the custom GATT structure has no malformed or missing elements.

Use this command to check for errors in the custom GATT structure configured in EZ-Serial. The dynamic GATT implementation automatically tests for validity issues when making changes to the structure with the gatts_create_attr (/CAC, ID=5/1) and gatts_delete_attr (/CAD, ID=5/2) API commands, but this command will provide the same test result upon request without making or attempting any modifications. See Table 7-5. EZ-Serial System Error Codes

EZ-Serial *GATT Database Validation Error Codes* for possible validity states.

EZ-Serial allows only one non-valid state, indicated by the GATTS_DB_VALID_WARNING_NOT_ENOUGH_ATTRIBUTES code (0x0001). This non-valid state is unavoidable during custom attribute creation, since attributes must be added one at a time, and every new service or characteristic requires multiple attributes. All other non-valid states prevent the addition of a custom attribute in the first place. Therefore, running this command should only result in a valid state (0x0000) or the warning state noted here (0x0001).

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`05`	`03`	None.
RSP	`C0`	`04`	`05`	`03`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/VGDB`	`0x0012`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`valid`	`V`	GATT database validity status. • 0x0000 – Success, GATT database is validated. • 0x0001 – Warning, GATT database attributes are not enough. Note that the warning is displayed when the GATT attribute creation is in progress and not yet complete. You should consider this value as success when creating GATT attributes. • 0x0002 – Error, attribute count limitation exceeded. Maximum 128 attributes supported. • 0x0003 – Error, attribute data exceeded. No resource can be allocated for the value data. • 0x0004 – Error, constant data exceeded. No resource can be allocated to store constant data. • 0x0005 – Error, invalid CCCD value data length. Note that the length of the CCCD value should always be greater than 2. • 0x0006 – Error, a service declaration is required. • 0x0007 – Error, unexpected service declaration. • 0x0008 – Error, a characteristic declaration is required. • 0x0009 – Error, unexpected characteristic declaration. • 0x000A – Error, a characteristic value attribute required.

Data Type	Name	Text	Description
			• 0x000B – Error, unexpected characteristic descriptor declaration. • 0x000C – Error, invalid attribute properties. • 0x000D – Error, invalid attribute length. • 0x000E – Error, invalid attribute data length. Other values are reserved and indicate the GATT database data is invalid.

Related Commands:

- gatts_create_attr (/CAC, ID=5/1)
- gatts_delete_attr (/CAD, ID=5/2)
- gatts_dump_db (/DGDB, ID=5/5)

7.2.5.4 gatts_store_db (/SGDB, ID=5/4)

Stores the current custom GATT structure in flash.

Note: This command has been deprecated and has no effect when used. In the latest firmware build, GATT database changes are always written instantly to flash when using either gatts_create_attr (/CAC, ID=5/1) or gatts_delete_attr (/CAD, ID=5/2).

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`05`	`04`	None.
RSP	`C0`	`02`	`05`	`04`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/SGDB`	`0x000B`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- gatts_create_attr (/CAC, ID=5/1)
- gatts_delete_attr (/CAD, ID=5/2)
- gatts_validate_db (/VGDB, ID=5/3)
- gatts_dump_db (/DGDB, ID=5/5)

7.2.5.5 gatts_dump_db (/DGDB, ID=5/5)

Lists the current local GATT database attributes.

This command produces a series of gatts_db_entry_blob (DGATT, ID=5/4) API events, one for each attribute in the current local GATT database. The output is similar to that of the gatts_discover_descriptors (/DLD, ID=5/8) API command, but in a format that more closely matches the input parameters of the gatts_create_attr (/CAC, ID=5/1) API command.

You can choose to dump only those attributes in the user-definable range (0x001D and above), or include fixed attributes as well (0x0001 and above) for complete reference.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01	05	05	None.
RSP	C0	04	05	05	None.

Text Info:

Text Name	Response Length	Notes
/DGDB	0x0012	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	include_fixed	F	Include fixed attributes: • 0 = Start from handle 0x001D, do not include fixed attributes (default) • 1 = Start from handle 0x0001, include fixed attributes

Response Parameters:

Data Type	Name	Text	Description
uint16	count	C	Number of entries to be returned

Related Commands:

- gatts_create_attr (/CAC, ID=5/1)
- gatts_delete_attr (/CAD, ID=5/2)
- gatts_validate_db (/VGDB, ID=5/3)
- gatts_discover_descriptors (/DLD, ID=5/8)

Related Events:

- gatts_db_entry_blob (DGATT, ID=5/4)

7.2.5.6 gatts_discover_services (/DLS, ID=5/6)

Requests a list of all services in the local GATT structure.

This allows convenient discovery of services within the local GATT database. This command does not require an active connection, since it concerns only local resources. Normally, there should not be a need to use this command except during development, since the application should already know all relevant details about its own local GATT structure. To find all services in the local database, use “0” for both arguments, or explicitly set 0x0001 and 0xFFFF for the beginning and end handles.

The gatts_discover_result (DL, ID=5/1) API events resulting from this command have the same format as the client-side gattc_discover_result (DR, ID=6/1) events which result from the gattc_discover_services (/DRS, ID=6/1) API command for discovering remote GATT services.

For local GATT database information that more closely matches the input format required for the gatts_create_attr (/CAC, ID=5/1) API command, use the gatts_dump_db (/DGDB, ID=5/5) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	04	05	06	None.
RSP	C0	04	05	06	None.

Text Info:

Text Name	Response Length	Category	Notes
/DLS	0x0011	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`begin`	`B`	Handle to begin searching
`uint16`	`end`	`E`	Handle to end searching (inclusive)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`count`	`C`	Number of entries to be returned

Related Commands:

- gatts_dump_db (/DGDB, ID=5/5)
- gatts_discover_characteristics (/DLC, ID=5/7)
- gatts_discover_descriptors (/DLD, ID=5/8)

Related Events:

- gatts_discover_result (DL, ID=5/1)

Example Usage:

See GATT Server Examples (**Note**: Any attribute that requires authentication (bonding) must also require encryption. If you enable the authentication bit, make sure that you also enable the encryption bit, or the command will be rejected with an error).

7.2.5.7 gatts_discover_characteristics (/DLC, ID=5/7)

Requests a list of all characteristics in the local GATT structure.

This allows convenient discovery of characteristics within the local GATT database. This command does not require an active connection, since it concerns only local resources. Normally, there should not be a need to use this command except during development, since the application should already know all relevant details about its own local GATT structure. To find all characteristics in the local database, use “0” for both arguments, or explicitly set 0x0001 and 0xFFFF for the beginning and end handles.

The gatts_discover_result (DL, ID=5/1) API events resulting from this command have the same format as the client-side gattc_discover_result (DR, ID=6/1) events which result from the gattc_discover_characteristics (/DRC, ID=6/2) API command for discovering remote GATT characteristics.

For local GATT database information that more closely matches the input format required for the gatts_create_attr (/CAC, ID=5/1) API command, use the gatts_dump_db (/DGDB, ID=5/5) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`06`	`05`	`07`	None.
RSP	`C0`	`04`	`05`	`07`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/DLC`	`0x0011`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`begin`	`B`	Handle to begin searching
`uint16`	`end`	`E`	Handle to end searching (inclusive)
`uint16`	`service`	`S`	Service UUID filter (ignored for all), little-endian format.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`count`	`C`	Number of entries to be returned

Related Commands:

- gatts_dump_db (/DGDB, ID=5/5)
- gatts_discover_services (/DLS, ID=5/6)
- gatts_discover_descriptors (/DLD, ID=5/8)

Related Events:

- gatts_discover_result (DL, ID=5/1)

Example Usage:

- See Listing Local GATT Services, Characteristics, and Descriptors

7.2.5.8 gatts_discover_descriptors (/DLD, ID=5/8)

Requests a list of all descriptors in the local GATT structure.

This allows convenient discovery of descriptors within the local GATT database. This command does not require an active connection, since it concerns only local resources. Normally, there should not be a need to use this command except during development, since the application should already know all relevant details about its own local GATT structure. To find all descriptors in the local database, use “0” for both arguments, or explicitly set 0x0001 and 0xFFFF for the beginning and end handles, respectively.

The gatts_discover_result (DL, ID=5/1) API events resulting from this command have the same format as the client-side gattc_discover_result (DR, ID=6/1) events which result from the gattc_discover_descriptors (/DRD, ID=6/3) API command for discovering remote GATT descriptors.

For local GATT database information that more closely matches the input format required for the gatts_create_attr (/CAC, ID=5/1) API command, use the gatts_dump_db (/DGDB, ID=5/5) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`08`	`05`	`08`	None.
RSP	`C0`	`04`	`05`	`08`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/DLD`	`0x0011`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`begin`	`B`	Handle to begin searching
`uint16`	`end`	`E`	Handle to end searching (inclusive)
`uint16`	`service`	`S`	Service UUID filter (0 for all), little-endian format.
`uint16`	`characteristic`	`C`	Characteristic UUID filter (0 for all), little-endian format.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`count`	`C`	Number of entries to be returned

Related Commands:

- gatts_dump_db (/DGDB, ID=5/5)
- gatts_discover_services (/DLS, ID=5/6)
- gatts_discover_characteristics (/DLC, ID=5/7)

Related Events:

- gatts_discover_result (DL, ID=5/1)

Example Usage:

- See Listing Local GATT Services, Characteristics, and Descriptors

7.2.5.9 gatts_read_handle (/RLH, ID=5/9)

Reads the value of an attribute in the local GATT server.

This command does not require an active connection, since it concerns only local resources. To read a value from a remote attribute on a connected peer, use the gattc_read_handle (/RRH, ID=6/4) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`02`	`05`	`09`	None.
RSP	`C0`	`04+`	`05`	`09`	Variable-length response payload, value specified is minimum.

Text Info:

Text Name	Response Length	Category	Notes
`/RLH`	`0x000D+`	ACTION	Variable-length response payload, value specified is minimum.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`attr_handle`	`H*`	Handle of the attribute from which the value is read

Response Parameters:

Data Type	Name	Text	Description
`longuint8a`	`data`	`D`	Data read from the attribute. It includes the attribute value data when it exists though it was not set when creating the attribute. **Note:** `longuint8a` data type requires two prefixed “length” bytes before binary parameter payload

Related Commands:

- gatts_write_handle (/WLH, ID=5/10)
- gattc_read_handle (/RRH, ID=6/4)

7.2.5.10 gatts_write_handle (/WLH, ID=5/10)

Writes a new value to a characteristic value or a characteristic descriptor attribute in the local GATT server.

This command does not require an active connection, since it concerns only local resources. To write a value to a remote attribute on a connected peer, use the gattc_write_handle (/WRH, ID=6/5) API command.

Note: Writing data to a local characteristic value attribute will not automatically trigger a notification or indication of that data to a connected client, even if the client has subscribed to notifications or indications for the characteristic. This command only affects the value stored locally in RAM if the client performs a GATT read operation later. To push data to a client that subscribed to notifications or indications, use the gatts_notify_handle (/NH, ID=5/11) or gatts_indicate_handle (/IH, ID=5/12) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`04`	`05`	`0A`	Variable-length command payload, value specified is minimum.
RSP	`C0`	`02`	`05`	`0A`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/WLH`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`attr_handle`	`H*`	Handle of the attribute to which a new value is written
`longuint8a`	`data`	`D*`	New data to write to attribute, little endian format. When the attribute data has constant data, the total data length including the constant part of the data should not exceed the length when it was created using gatts_create_attr (/CAC, ID=5/1) command. **Note:** `longuint8a` data type requires two prefixed "length" bytes before binary parameter payload.

Response Parameters:

None.

Related Commands:

- gatts_read_handle (/RLH, ID=5/9)
- gatts_notify_handle (/NH, ID=5/11)
- gatts_indicate_handle (/IH, ID=5/12)
- gattc_write_handle (/WRH, ID=6/5)

7.2.5.11 gatts_notify_handle (/NH, ID=5/11)

Notifies a new attribute value to a remote GATT client.

Note: This command does not change any locally stored values for the notified attribute. To modify the data stored locally in RAM for the attribute in question, use the gatts_write_handle (/WLH, ID=5/10) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`06`	`05`	`0B`	Variable-length command payload, value specified is minimum.
RSP	`C0`	`02`	`05`	`0B`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/NH`	`0x0009`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for notification. See the gap_connected (C, ID=4/5) event to find the connection handle.
`uint16`	`attr_handle`	`H*`	Handle of attribute to be notified
`uint8a`	`data`	`D*`	Data to be pushed to the remote client via notification **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- gatts_write_handle (/WLH, ID=5/10)
- gatts_indicate_handle (/IH, ID=5/12)

7.2.5.12 gatts_indicate_handle (/IH, ID=5/12)

Indicates a new attribute value to a remote GATT client.

If successful, pushing an indicated value to a remote client will result in the gatts_indication_confirmed (IC, ID=5/3) API event occurring after the client acknowledges the transfer.

This method requires client acknowledgement, so you cannot attempt another GATT operation until this confirmation event arrives. A single acknowledged transfer requires two connection intervals: one for the actual data transfer, and another for the acknowledgement. Using this type of transfer has effects on potential throughput; See Maximizing Throughput to a Remote Peer for details on alternative design choices.

Note: This command does not change any locally stored values for the indicated attribute. To modify the data stored locally in RAM for the attribute in question, use the gatts_write_handle (/WLH, ID=5/10) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`06`	`05`	`0C`	Variable-length command payload, value specified is minimum.
RSP	`C0`	`02`	`05`	`0C`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/IH`	`0x0009`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for indication. See the gap_connected (C, ID=4/5) event to find the connection handle.
`uint16`	`attr_handle`	`H*`	Handle of the attribute to be used for indication
`uint8a`	`data`	`D*`	Data to be indicated **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- gatts_read_handle (/RLH, ID=5/9)
- gatts_write_handle (/WLH, ID=5/10)
- gatts_notify_handle (/NH, ID=5/11)
- gattc_confirm_indication (/CI, ID=6/6) – Used on remote client to confirm receipt of the indication

Related Events:

- gatts_indication_confirmed (IC, ID=5/3) - Occurs on the server after the remote client confirms receipt of indicated data
- gattc_data_received (D, ID=6/3) – Occurs on the remote client when indicated data is received

7.2.5.13 gatts_send_writereq_response (/WRR, ID=5/13)

Responds to a GATT client's acknowledged write request.

Use this command after receiving a gatts_data_written (W, ID=5/2) API event, an acknowledged request to write data to a local GATT server attribute (the event's **`type`** parameter will be 0x80). Sending a response value of zero indicates success, while any non-zero value indicates an error. Values 0x01 through 0x7F are errors defined in the Bluetooth specification, while values 0x80 through 0xFF are user-defined errors.

EZ-Serial will automatically respond to write requests unless **Bit 0** of the GATT server behavior flags is cleared using the **`flags`** field in the gatts_set_parameters (SGSP, ID=5/14) API command, or if the characteristic being written has **Bit 24** set for user data management in the GATT database structure entry created with the gatts_create_attr (/CAC, ID=5/1) API command.

Note: For WICED OTA service handles, EZ-Serial will always automatically respond to a write request, and the Bit0 of the GATT server behavior flags will be ignored.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`02`	`05`	`0D`	None.
RSP	`C0`	`02`	`05`	`0D`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/WRR`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for response (Ignored in current release due to internal BLE stack functionality, set to 0)
`uint8`	`response`	`R`*	GATT result code for response: • 0 = Success • 0x01-0x7F = Error from Bluetooth specification • 0x80-0xFF = Error from application (user-defined)

Response Parameters:

None.

Related Commands:

- gattc_write_handle (/WRH, ID=6/5)

Related Events:

- gatts_data_written (W, ID=5/2)

7.2.5.14 gatts_set_parameters (SGSP, ID=5/14)

Configures new GATT server parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`05`	`0E`	None.
RSP	`C0`	`02`	`05`	`0E`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SGSP`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`flags`	`F`	GATT server behavior flags. Bit 0 (0x01) = Enable automatic response to acknowledged writes. • 0 = Automatic response to acknowledged writes is disabled • 1 = Automatic response to acknowledged writes is enabled (Factory default value) Bit 1 (0x02) = Enable bit 4-7 field for connection established BLE advertisement type. • 0 = Disable bit 4-7 field, these settings will not take effect. • 1 = Enable bit 4-7 field, these settings will take effect (Factory default value) Bit 4-7 (0xF0) = BLE advertisement type when the BLE connection has been established. • 0 – Stop advertising • 1 – Directed advertisement high duty cycle • 2 – Directed advertisement low duty cycle • 3 – Undirected advertisement high duty cycle • 4 – Undirected advertisement low duty cycle • 5 – Non-connectable advertisement high duty cycle • 6 – Non-connectable advertisement low duty cycle (Factory default value) • 7 – Discoverable advertisement high duty cycle • 8 – Discoverable advertisement low duty cycle The 4-bit fields will take effect only when you manually start advertising again after a BLE connection has been established. See gap_start_adv (/A, ID=4/8) for more details.

Response Parameters:

None.

Related Commands:

- gatts_send_writereq_response (/WRR, ID=5/13) – Necessary to use for acknowledged client writes if flags Bit 0 is clear
- gatts_get_parameters (GGSP, ID=5/15)

7.2.5.15 gatts_get_parameters (GGSP, ID=5/15)

Obtains current GATT server parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`05`	`0F`	None.
RSP	`C0`	`03`	`05`	`0F`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GGSP`	`0x000F`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`flags`	`F`	GATT server behavior flags. Bit 0 (0x01) = Enable automatic response to acknowledged writes. • 0 = Automatic response to acknowledged writes is disabled • 1 = Automatic response to acknowledged writes is enabled (Factory default value) Bit 1 (0x02) = Enable bit 4-7 field for connection established BLE advertisement type. • 0 = Disable bit 4-7 field, these settings will not take effect. • 1 = Enable bit 4-7 field, these settings will take effect (Factory default value) Bit 4-7 (0xF0) = BLE advertisement type when BR/EDR connection has been established. • 0 – Stop advertising (Factory default value) • 1 – Directed advertisement high duty cycle • 2 – Directed advertisement low duty cycle • 3 – Undirected advertisement high duty cycle • 4 – Undirected advertisement low duty cycle • 5 – Non-connectable advertisement high duty cycle • 6 – Non-connectable advertisement low duty cycle • 7 – Discoverable advertisement high duty cycle • 8 – Discoverable advertisement low duty cycle

Related Commands:

- gatts_set_parameters (SGSP, ID=5/14)

7.2.6 GATT Client Group (ID=6)

GATT client methods relate to the client role of the Generic Attribute Protocol layer of the Bluetooth stack. These methods are used for working with the GATT structures on remote devices, and can only be used while a device is connected.

Commands within this group are listed below:

- gattc_discover_services (/DRS, ID=6/1)
- gattc_discover_characteristics (/DRC, ID=6/2)
- gattc_discover_descriptors (/DRD, ID=6/3)
- gattc_read_handle (/RRH, ID=6/4)
- gattc_write_handle (/WRH, ID=6/5)
- gattc_confirm_indication (/CI, ID=6/6)
- gattc_set_parameters (SGCP, ID=6/7)
- gattc_get_parameters (GGCP, ID=6/8)

Events within this group are documented in GATT Client Group (ID=6).

7.2.6.1 *gattc_discover_services (/DRS, ID=6/1)*

Requests a list of GATT services from a connected remote GATT server.

This command performs a GATT client operation, and requires a connection to a remote peer. To discover the local GATT structure instead, use the gatts_discover_services (/DLS, ID=5/6) API command.

Note: This command works with remote data, so it cannot determine, in advance, the number of records to be returned. Only local GATT server discovery operations can do this. Therefore, you must wait for the gattc_remote_procedure_complete (RPC, ID=6/2) API event to indicate that the discovery procedure is finished.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`05`	`06`	`01`	None.
RSP	`C0`	`02`	`06`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/DRS`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for discovery
`uint16`	`begin`	`B`	Handle to begin searching
`uint16`	`end`	`E`	Handle to end searching (inclusive)

Response Parameters:

None.

Related Commands:

- gatts_discover_services (/DLS, ID=5/6)
- gattc_discover_characteristics (/DRC, ID=6/2)
- gattc_discover_descriptors (/DRD, ID=6/3)

Related Events:

- gattc_discover_result (DR, ID=6/1)
- gattc_remote_procedure_complete (RPC, ID=6/2)

Example Usage:

- See Discovering a Remote Server's GATT Structure

7.2.6.2 *gattc_discover_characteristics (/DRC, ID=6/2)*

Requests a list of GATT characteristics from a connected remote GATT server.

This command performs a GATT client operation, and requires a connection to a remote peer. To discover the local GATT structure instead, use the gatts_discover_characteristics (/DLC, ID=5/7) API command.

Note: This command works with remote data, so it cannot determine, in advance, the number of records to be returned. Only local GATT server discovery operations can do this. Therefore, you must wait for the gattc_remote_procedure_complete (RPC, ID=6/2) API event to indicate that the discovery procedure is complete.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`07`	`06`	`02`	None.
RSP	`C0`	`02`	`06`	`02`	None.

Text Info:

Text Name	Response Length	Notes
`/DRC`	`0x000A`	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for discovery
`uint16`	`begin`	`B`	Handle to begin searching
`uint16`	`end`	`E`	Handle to end searching (inclusive)
`uint16`	`service`	`S`	Service UUID filter (0 for all) (Ignored in current release, set to 0)

Response Parameters:

None.

Related Commands:

- gatts_discover_characteristics (/DLC, ID=5/7)
- gattc_discover_services (/DRS, ID=6/1)
- gattc_discover_descriptors (/DRD, ID=6/3)

Related Events:

- gattc_discover_result (DR, ID=6/1)
- gattc_remote_procedure_complete (RPC, ID=6/2)

Example Usage:

- See Discovering a Remote Server's GATT Structure

7.2.6.3 gattc_discover_descriptors (/DRD, ID=6/3)

Requests a list of GATT attribute descriptors from a connected remote GATT server.

This command performs a GATT client operation, and requires a connection to a remote peer. To discover the local GATT structure, use the gatts_discover_descriptors (/DLD, ID=5/8) API command instead.

Note: This command works with remote data, so it cannot determine, in advance, the number of records to be returned. Only local GATT server discovery operations can do this. Therefore, you must wait for the gattc_remote_procedure_complete (RPC, ID=6/2) API event to indicate that the discovery procedure is complete.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`09`	`06`	`03`	None.
RSP	`C0`	`02`	`06`	`03`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/DRD`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for discovery
`uint16`	`begin`	`B`	Handle to begin searching
`uint16`	`end`	`E`	Handle to end searching (inclusive)

Data Type	Name	Text	Description
`uint16`	`service`	`S`	Service UUID filter (0 for all) (Ignored in current release, set to 0)
`uint16`	`characteristic`	`T`	Characteristic UUID filter (0 for all) (Ignored in current release, set to 0)

Response Parameters:

None.

Related Commands:

- gatts_discover_descriptors (/DLD, ID=5/8)
- gattc_discover_result (DR, ID=6/1)
- gattc_discover_characteristics (/DRC, ID=6/2)

Related Events:

- gattc_discover_result (DR, ID=6/1)
- gattc_remote_procedure_complete (RPC, ID=6/2)

Example Usage:

- See Discovering a Remote Server's GATT Structure

7.2.6.4 gattc_read_handle (/RRH, ID=6/4)

Reads the value of an attribute on a remote GATT server.

This command performs a GATT client operation, and requires a connection to a remote peer. To read a value from the local GATT structure, use the gatts_read_handle (/RLH, ID=5/9) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`03`	`06`	`04`	None.
RSP	`C0`	`02`	`06`	`04`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/RRH`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for read operation
`uint16`	`attr_handle`	`H`*	Handle of the remote attribute to be read

Response Parameters:

None.

Related Commands:

- gattc_write_handle (/WRH, ID=6/5)

Related Events:

- gattc_remote_procedure_complete (RPC, ID=6/2) – Occurs if the client read operation fails (parameters include error code)
- gattc_data_received (D, ID=6/3) – Occurs if the client read operation succeeds

7.2.6.5 gattc_write_handle (/WRH, ID=6/5)

Writes a new value to an attribute on a remote GATT server.

This command performs a GATT client operation, and requires a connection to a remote peer. To write a value to the local GATT structure, use the gatts_write_handle (/WLH, ID=5/10) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`06`	`06`	`05`	Variable-length command payload, value specified is minimum.
RSP	`C0`	`02`	`06`	`05`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/WRH`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for write operation
`uint16`	`attr_handle`	`H`*	Handle of the remote attribute to be written to
`uint8`	`type`	`T`	Type of write to perform: • 0 = Write with response (default, acknowledged) • 1 = Write without response (unacknowledged) **Note:** When the GATT write type does not match the GATT characteristic permission and properties settings, the command itself can be executed with a success result, but data might not be send out or received. Also, the expected data might not be reported. For details on the GATT write type, see Table 3-13 and the gatts_create_attr (/CAC, ID=5/1) command.
`longuint8a`	`data`	`D`*	New data to write **Note:** `longuint8a` data type requires two prefixed "length" bytes before binary parameter payload

Response Parameters:

None.

Related Commands:

- gattc_read_handle (/RRH, ID=6/4)
- gatts_send_writereq_response (/WRR, ID=5/13)

Related Events:

- gatts_data_written (W, ID=5/2) – Occurs on the remote server after using this command on the local client
- gattc_remote_procedure_complete (RPC, ID=6/2) – Occurs once the write is acknowledged, if using acknowledged write type

7.2.6.6 gattc_confirm_indication (/CI, ID=6/6)

Confirms an indication from a remote GATT server.

This command confirms receipt of indicated data from a remote server. Indicated data is pushed from a server to a client after the client has subscribed to indications for a desired characteristic and that characteristic's value has changed. Indicated data will arrive via the gattc_data_received (D, ID=6/3) API event, and you must use this command to manually confirm the indication if the `source` parameter of that event shows indication with manual confirmation needed. See the gatts_indication_confirmed (IC, ID=5/3) event for details.

EZ-Serial will automatically confirm indications unless Bit 0 of the GATT client behavior flags is cleared using the **Flags** field in the gattc_set_parameters (SGCP, ID=6/7) API command.

Note: If indicated data arrives and requires manual confirmation, you must use this command to confirm it before performing any other GATT operations.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`06`	`06`	None.
RSP	`C0`	`02`	`06`	`06`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/CI`	`0x0009`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for confirmation
`uint16`	`attr_handle`	`H`*	Handle of the remote attribute to written to
`longuint8a`	`data`	`D`	Indication value to be passed if required. This data field is not required. **Note:** `longuint8a` data type requires two prefixed "length" bytes before binary parameter payload

Response Parameters:

None.

Related Commands:

- gatts_indicate_handle (/IH, ID=5/12) – Used on a remote GATT server to indicate data to a client
- gattc_set_parameters (SGCP, ID=6/7) – Configures local GATT client parameters, including auto-confirm behavior

Related Events:

- gatts_indication_confirmed (IC, ID=5/3) – Occurs on a remote GATT server after confirming indication on the client
- gattc_data_received (D, ID=6/3) – Occurs on the local GATT client when a remote server indicates data

7.2.6.7 gattc_set_parameters (SGCP, ID=6/7)

Configures new GATT client parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`06`	`07`	None.
RSP	`C0`	`02`	`06`	`07`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SGCP`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`flags`	`F`	GATT client behavior flags bitmask: • Bit 0 (0x01) = Enable automatic confirmation to remote GATT server indications **Note:** Factory default is 0x01 (all bits set)

Response Parameters:

None.

Related Commands:

- gattc_confirm_indication (/CI, ID=6/6) – Necessary to use for indicated data if flags Bit 0 is clear
- gattc_get_parameters (GGCP, ID=6/8)

7.2.6.8 *gattc_get_parameters (GGCP, ID=6/8)*

Gets current GATT client parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`06`	`08`	None.
RSP	`C0`	`03`	`06`	`08`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GGCP`	`0x000F`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`flags`	`F`	GATT client behavior flags bitmask: • Bit 0 (0x01) = Enable automatic confirmation of remote GATT server indications **Note:** Factory default is 0x01 (all bits set)

Related Commands:

- gattc_set_parameters (SGCP, ID=6/7)

7.2.7 SMP Group (ID=7)

SMP methods relate to the Security Manager Protocol layer of the Bluetooth stack. These methods are used for working with privacy, encryption, pairing, and bonding between two devices.

Commands within this group are listed below:

- smp_query_bonds (/QB, ID=7/1)
- smp_delete_bond (/BD, ID=7/2)
- smp_pair (/P, ID=7/3)
- smp_query_random_address (/QRA, ID=7/4)
- smp_send_pairreq_response (/PR, ID=7/5)
- smp_send_passkeyreq_response (/PE, ID=7/6)
- smp_set_privacy_mode (SPRV, ID=7/9)
- smp_get_privacy_mode (GPRV, ID=7/10)
- smp_set_fixed_passkey (SFPK, ID=7/13)
- smp_get_fixed_passkey (GFPK, ID=7/14)
- system_get_bluetooth_address (GBA, ID=2/14)
- system_set_sleep_parameters (SSLP, ID=2/19)
- system_get_sleep_parameters (GSLP, ID=2/20)
- system_set_tx_power (STXP, ID=2/21)

- system_get_tx_power (GTXP, ID=2/22)
- system_set_transport (ST, ID=2/23)
- smp_get_le_security_parameters (GSLSP, ID=7/21)
- system_set_uart_parameters (STU, ID=2/25)
- smp_get_parameters (GSMPP, ID=7/33)

7.2.7.1 smp_query_bonds (/QB, ID=7/1)

Requests a list of bonded devices.

This command accesses the current bonded device list. Bonded devices are those which have previously paired (exchanged encryption data) and bonded (stored the exchanged encryption data).

The response from this command includes the number of bonded devices, and the response will be followed by many smp_bond_entry (B, ID=7/1) API events that provide details for each device.

Note: EZ-Serial currently supports a maximum of four bonded devices at the same time. To bond with additional devices after all four bond slots are full, you must delete one of the existing bonds with the smp_delete_bond (/BD, ID=7/2) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`07`	`01`	None.
RSP	`C0`	`03`	`07`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/QB`	`0x000E`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`count`	`C`	Bond entry count

Related Commands:

- smp_pair (/P, ID=7/3) – Creates a new bond entry if pairing process succeeds with bonding enabled

Related Events:

- smp_bond_entry (B, ID=7/1) – Occurs once for each bonded device after requesting bond list

7.2.7.2 smp_delete_bond (/BD, ID=7/2)

Removes a bonded device.

This command removes the stored encryption key data for a device that has previously paired (exchanged encryption data) and bonded (stored the exchanged encryption data).

Note: When the bonded device is connected, deleting the bonded key operation will disconnect after the bonded device key is removed.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`07`	`07`	`02`	None.
RSP	`C0`	`03`	`07`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
/BD	0x000E	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
Macaddr	address	A*	Bluetooth address
uint8	type	T	Address type: • 0 = Public (default) • 1 = Random/private

Response Parameters:

Data Type	Name	Text	Description
uint8	count	C	Updated bond entry count

Related Commands:

- smp_query_bonds (/QB, ID=7/1)
- smp_pair (/P, ID=7/3) – Creates a new bond entry if pairing process succeeds with bonding enabled

7.2.7.3 smp_pair (/P, ID=7/3)

Initiates pairing process with a connected device.

Note: EZ-Serial currently supports a maximum of four bonded devices at the same time. To bond with additional devices after all four bond slots are full, you must delete one of the existing bonds with the smp_delete_bond (/BD, ID=7/2) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	03	07	03	None.
RSP	C0	02	07	03	None.

Text Info:

Text Name	Response Length	Category	Notes
/P	0x0008	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for pairing
uint8	bonding	B	Bond during pairing process: • 0 = Do not bond (exchange keys and encrypt only) • 1 = Bond (permanently store exchanged encryption data)
uint8	sec_action_type	S	BLE encryption method. • 0x00 = NO encryption • 0x01 = Encrypt the link using current key • 0x02 = Encryption without MITM • 0x03 = Encryption with MITM

Response Parameters:

None.

Related Commands:

- smp_send_pairreq_response (/PR, ID=7/5) – Used when remote device initiates pairing and auto-accept flag bit is not disabled
- smp_send_passkeyreq_response (/PE, ID=7/6) – Used if MITM protection is enabled and pairing requires passkey entry

7.2.7.4 *smp_query_random_address (/QRA, ID=7/4)*

Requests the current local random address.

When peripheral or central privacy is enabled with the smp_set_privacy_mode (SPRV, ID=7/9) API command, the Bluetooth connection address is visible to remote devices while advertising, or scanning will be random (private) instead of the fixed (public) Bluetooth address that can be configured or obtained using the system_set_bluetooth_address (SBA, ID=2/13) and system_get_bluetooth_address (GBA, ID=2/14) API commands. This type of privacy helps to avoid profiling by a passive eavesdropper.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`07`	`04`	None.
RSP	`C0`	`08`	`07`	`04`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/QRA`	`0x0019`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`macaddr`	`address`	`A`	Random address

Related Commands:

- smp_set_privacy_mode (SPRV, ID=7/9)

7.2.7.5 *smp_send_pairreq_response (/PR, ID=7/5)*

Sends a response to a user confirmation pairing request from a remote device.

EZ-Serial will automatically accept pairing requests unless Bit 0 of the security behavior flags is cleared using the **`flags`** field in the smp_set_parameters (SSMPP, ID=7/32) API command. If the auto-accept feature is disabled, use this command to manually accept or deny a remotely initiated pairing process.

Note: WICED BT Stack will pair automatically after the connection is established, so this API is not used and supported in current EZ-Serial.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`03`	`07`	`05`	None.
RSP	`C0`	`02`	`07`	`05`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/PR`	`0x0009`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for sending response
`uint16`	`response`	`R*`	Response (0 = accept, non-zero = reject)

Response Parameters:

None.

Related Commands:

- smp_pair (/P, ID=7/3) – Used to initiate pairing

Related Events:

- smp_user_confirmation_requested (UCNFMREQ, ID=7/8)
- smp_pairing_requested (P, ID=7/2) – Occurs when a remote device requests pairing
- smp_pairing_result (PR, ID=7/3) – Occurs after a pairing process completes (successfully or otherwise)

7.2.7.6 smp_send_passkeyreq_response (/PE, ID=7/6)

Sends a passkey value back to a remote device that requested it.

Use this command after receiving the smp_passkey_entry_requested (PKE, ID=7/6) API event, or when I/O capabilities are set to “Display + Yes/No” to indicate acceptance after receiving the smp_passkey_display_requested (PKD, ID=7/5) API event.

Note: WICED BT Stack will pair automatically after connection established, so this API is not used and supported in current EZ-Serial.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`05`	`07`	`06`	None.
RSP	`C0`	`02`	`07`	`06`	None.

Text Info:

Text Name	Response Length	Notes
`/PE`	`0x0009`	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle to be used for sending response (Ignored in current release due to internal BLE stack functionality, set to 0)
`uint32`	`passkey`	`P*`	Passkey value (000000-999999, 0x0 – 0x0F423F)

Response Parameters:

None.

Related Commands:

- smp_pair (/P, ID=7/3)

Related Events:

- smp_passkey_display_requested (PKD, ID=7/5)
- smp_passkey_entry_requested (PKE, ID=7/6)

7.2.7.7 smp_set_privacy_mode (SPRV, ID=7/9)

Configures new privacy settings.

Updates privacy mode if device is already available in the controller resolving list.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`03`	`07`	`09`	None.
RSP	`C0`	`02`	`07`	`09`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SPRV`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	Privacy mode: • 0x00 – Network privacy mode. • 0x01 – Device privacy mode.

Response Parameters:

None.

Related Commands:

- smp_get_privacy_mode (GPRV, ID=7/10)

7.2.7.8 smp_get_privacy_mode (GPRV, ID=7/10)

Obtain current privacy settings.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`07`	`0A`	None.
RSP	`C0`	`05`	`07`	`0A`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GPRV`	`0x0016`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	Privacy mode: • 0x00 – Network privacy mode. • 0x01 – Device privacy mode.

Related Commands:

- smp_set_privacy_mode (SPRV, ID=7/9)

7.2.7.9 smp_set_fixed_passkey (SFPK, ID=7/13)

Configures new fixed passkey value.

While the Bluetooth specification describes that the passkey should be randomized during pairing, you can configure a fixed (non-random) 6-digit passkey between 000000 and 999999 using this command and configuring the local I/O capabilities to the “Display Only” value. During pairing, EZ-Serial will generate the smp_passkey_display_requested (PKD, ID=7/5) API event containing the value configured here. The remote peer must then enter this key to pair successfully.

Note:

- The fixed passkey defined here will only take effect if you enable fixed passkey by setting Bit 1 (0x02) of the security flags parameter and set the “Display Only” I/O capabilities value (0x00) using the API command. If both conditions are not met, the stack will revert to the default behavior of using a random passkey.
- WICED BT Stack will pair automatically after the connection is established, so this API is not used and supported in current EZ-Serial.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`04`	`07`	`0D`	None.
RSP	`C0`	`02`	`07`	`0D`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SFPK`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint32`	`passkey`	`P`	Fixed passkey value • Minimum = 0 (‘000000’ decimal entry during pairing) • Maximum = 0xF423F (‘999999’ decimal entry during pairing) **Note:** The passkey should be input in hexadecimal format without prefix “0x”, factory default is 0

Response Parameters:

None.

Related Commands:

- smp_pair (/P, ID=7/3)
- smp_send_pairreq_response (/PR, ID=7/5)

Related Events:

- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)
- smp_encryption_status (ENC, ID=7/4)
- smp_passkey_display_requested (PKD, ID=7/5)

Example Usage:

- See Pairing and Bonding with a Fixed Passkey

7.2.7.10 smp_get_fixed_passkey (GFPK, ID=7/14)

Obtains the current fixed passkey value.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	07	0E	None.
RSP	C0	08	07	0E	None.

Text Info:

Text Name	Response Length	Category	Notes
GFPK	0x0015	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint32	passkey	P	Fixed passkey value • Minimum = 0 ('000000' decimal entry during pairing) • Maximum = 0xF423F ('999999' decimal entry during pairing) **Note:** The passkey number is output in hexadecimal format; factory default is 0

Related Commands:

- smp_set_fixed_passkey (SFPK, ID=7/13)

7.2.7.11 smp_set_pin_code (SBTPIN, ID=7/15)

Configures the new PIN code value.

Note: WICED BT Stack will pair automatically after the connection established, so this API is not used and supported in current EZ-Serial.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	02	07	0F	None.
RSP	C0	02	07	0F	None.

Text Info:

Text Name	Response Length	Category	Notes
SBTPIN	0x000C	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint16	pin_code	P	The value of 4 bytes PIN code, in hexadecimal format, between 0x0 and 0x270F. That is, the pin code value must be between '0000' and '9999' in decimal format. Note that you must convert the decimal pin code value to hexadecimal value before using as input value.

Response Parameters:

None.

Related Commands:

- smp_pair (/P, ID=7/3)

Related Events:

- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)
- smp_encryption_status (ENC, ID=7/4)

7.2.7.12 smp_get_pin_code (GBTPIN, ID=7/16)

Obtains the current PIN code value.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`07`	`10`	None.
RSP	`C0`	`04`	`07`	`10`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GBTPIN`	`0x0013`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`pin_code`	`P`	The hexadecimal value of 4 bytes PIN code. Note that the output value is in hexadecimal format between 0x0 ~ 0x270F, which indicates that the fixed 4 bytes of PIN code is between '0000' and '9999' in decimal format.

Related Commands:

- smp_set_pin_code (SBTPIN, ID=7/15)

7.2.7.13 smp_send_pinreq_response (/BTPIN, ID=7/17)

Manually sends the PIN code response for a PIN code request in establishing a connection.

Note: WICED BT Stack will pair automatically after the connection is established, so this API is not used and supported in current EZ-Serial.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`03`	`07`	`11`	None.
RSP	`C0`	`02`	`07`	`11`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/BTPIN`	`0x000C`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Handle of the connection for which PIN code response is sent Note that the conn_handler must be the conn_handle value received in the smp_pin_entry_requested (BTPIN, ID=7/7) event.

Data Type	Name	Text	Description
`uint16`	`pin_code`	`P`	The hexadecimal value of 4 bytes PIN code. The pin code is a decimal number between ‘0000’ and ‘9999’. Note that you must convert the decimal pin code value to hexadecimal value before using as input value.

Response Parameters:

None.

Related Commands:

- smp_pair (/P, ID=7/3)

Related Events:

- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)
- smp_encryption_status (ENC, ID=7/4)

7.2.7.14 *smp_set_br_edr_security_parameters (SSBSP, ID=7/18)*

Configures new BR EDR security and bonding parameters.

These parameters will be used when the smp_pair (/P, ID=7/3) API command is used without specifying non-default arguments. These values are reported to the remote device as part of the pairing process and affect the type of key generation and exchange that takes place during pairing and bonding.

Note: Changing the I/O capabilities will affect the command/event flow necessary to complete a pairing and bonding process. See the related commands and events for more details. Also, MITM protection requires I/O capabilities other than “No Input + No Output” to function correctly.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`03`	`07`	`12`	None.
RSP	`C0`	`02`	`07`	`12`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SSBSP`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`local_io_cap`	`C`	Local IO capabilities: • 0x00 = Display only – Conveys a 6-digit number to the user • 0x01 = Display Yes/No – Displays and allows the user to indicate “yes” or “no” • 0x02 = Keyboard only – Allows the user to enter ‘0’ through ‘9’ and “yes” or “no” • 0x03 = No input, no output – Does not display or allow any input (Factory default) • 0x04 = Keyboard + Display – Provides full numeric input and display
`uint8`	`oob_data`	`O`	OOB data present (locally) for the peer device: • 0x00 = No OOB data (Factory default) • 0x01 = OOB data is present, which is from the P-192 public key • 0x02 = OOB data is present, which is from the P-256 public key • 0x03 = OOB data is present, which is from the P-192 and P256 public key • 0x04 = OOB data is unknown

Data Type	Name	Text	Description
`uint8`	`auth_req`	`A`	Authentication request (for local device) contain bonding and MITM information: • 0x00 = MITM Protection Not Required - Single Profile/non-bonding. Numeric comparison with automatic accept allowed. • 0x01 = MITM Protection Required - Single Profile/non-bonding. Use IO Capabilities to determine authentication procedure. • 0x02 = MITM Protection Not Required - All Profiles/dedicated bonding. Numeric comparison with automatic accept allowed. • 0x03 = MITM Protection Required - All Profiles/dedicated bonding. Use IO Capabilities to determine authentication procedure. • 0x04 = MITM Protection Not Required - Single Profiles/general bonding Numeric comparison with automatic acceptance is allowed. (Factory default) • 0x05 = MITM Protection Required - Single Profiles/general bonding. Use IO Capabilities to determine authentication procedure.

Response Parameters:

None.

Related Commands:

- smp_pair (/P, ID=7/3)
- smp_send_pairreq_response (/PR, ID=7/5)
- smp_send_passkeyreq_response (/PE, ID=7/6)
- smp_set_fixed_passkey (SFPK, ID=7/13)

Related Events:

- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)
- smp_encryption_status (ENC, ID=7/4)
- smp_passkey_display_requested (PKD, ID=7/5)
- smp_passkey_entry_requested (PKE, ID=7/6)

7.2.7.15 smp_get_br_edr_security_parameters (GSBSP, ID=7/19)

Obtains the current BR EDR security and bonding parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`07`	`13`	None.
RSP	`C0`	`05`	`07`	`13`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GSBSP`	`0x0028`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`local_io_cap`	`C`	Local IO capabilities: • 0x00 = Display only • 0x01 = Display Yes/No • 0x02 = Keyboard only • 0x03 = No input, no output • 0x04 = Keyboard + Display

Data Type	Name	Text	Description
`uint8`	`oob_data`	`O`	OOB data present (locally) for the peer device: • 0x00 = No OOB data • 0x01 = OOB data is present, which is from the P-192 public key • 0x02 = OOB data is present, which is from the P-256 public key • 0x03 = OOB data is present, which is from the P-192 and P256 public key • 0x04 = OOB data unknown
`uint8`	`auth_req`	`A`	Authentication request (for local device) contain bonding and MITM information: • 0x00 = MITM Protection Not Required - Single Profile/non-bonding. • 0x01 = MITM Protection Required - Single Profile/non-bonding. • 0x02 = MITM Protection Not Required - All Profiles/dedicated bonding. • 0x03 = MITM Protection Required - All Profiles/dedicated bonding. • 0x04 = MITM Protection Not Required - Single Profiles/general bonding. • 0x05 = MITM Protection Required - Single Profiles/general bonding.

Related Commands:

- smp_set_br_edr_security_parameters (SSBSP, ID=7/18)

7.2.7.16 smp_set_le_security_parameters (SSLSP, ID=7/20)

Configures new LE security and bonding parameters.

These parameters will be used when the smp_pair (/P, ID=7/3) API command is used without specifying non-default arguments. These values are reported to the remote device as part of the pairing process and affect the type of key generation and exchange that takes place during pairing and bonding.

Note: Changing the I/O capabilities will affect the command/event flow necessary to complete a pairing and bonding process. See the related commands and events for more details. Also, MITM protection requires I/O capabilities other than "No Input + No Output" to function correctly.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`06`	`07`	`14`	None.
RSP	`C0`	`02`	`07`	`14`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SSLSP`	`0x000A`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`local_io_cap`	`C`	Local IO capabilities: • 0x00 = Display only • 0x01 = Display Yes/No • 0x02 = Keyboard only • 0x03 = No input, no output • 0x04 = Keyboard + Display (For BLE SMP)
`uint8`	`oob_data`	`O`	OOB data present (locally) for the peer device: • 0x00 = No OOB data (Factory default) • 0x01 = OOB data is present, which is from the P-192 public key • 0x02 = OOB data is present, which is from the P-256 public key • 0x03 = OOB data is present, which is from the P-192 and P256 public key • 0x04 = OOB data unknown

Data Type	Name	Text	Description
`uint8`	`auth_req`	`A`	Authentication request (for local device) contain bonding and MITM information: • 0x00 = Not required – No bond (Factory default) • 0x01 = Required – General Bond • 0x04 = MITM required – Auth Y/N • 0x08 = LE secure connection, no MITM, no bonding • 0x09 = LE secure connection, no MITM, bonding • 0x0C = LE secure connection, MITM, no bonding • 0x0D = LE secure connection, MITM, bonding
`uint8`	`max_key_size`	`S`	Max encryption key size from 7 to 16 bytes. **Note:** Factory default is 16 bytes
`uint8`	`init_keys`	`I`	BLE key types (these bits can be set together): • Bit0 (0x01) = Encryption information of the peer device • Bit1 (0x02) = Identity key of the peer device • Bit2 (0x04) = Peer SRK • Bit3 (0x08) = Public key • Bit4 (0x10) = Master role security information • Bit5 (0x20) = Master device ID key • Bit6 (0x40) = Local CSRK key • Bit7 (0x80) = Link layer key **Note:** Factory default value is 0x17.
`uint8`	`resp_keys`	`R`	BLE keys to be distributed (bit mask, these bits can be masked together): • Bit0 (0x01) = Encryption information of the peer device • Bit1 (0x02) = Identity key of the peer device • Bit2 (0x04) = Peer SRK • Bit3 (0x08) = Public key • Bit4 (0x10) = Master role security information • Bit5 (0x20) = Master device ID key • Bit6 (0x40) = Local CSRK key • Bit7 (0x80) = Link layer key **Note:** The factory default value is 0x17.

Response Parameters:

None.

Related Commands:

- smp_pair (/P, ID=7/3)
- smp_send_pairreq_response (/PR, ID=7/5)
- smp_send_passkeyreq_response (/PE, ID=7/6)
- smp_set_fixed_passkey (SFPK, ID=7/13)

Related Events:

- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)
- smp_encryption_status (ENC, ID=7/4)
- smp_passkey_display_requested (PKD, ID=7/5)
- smp_passkey_entry_requested (PKE, ID=7/6)

7.2.7.17 smp_get_le_security_parameters (GSLSP, ID=7/21)

Obtains the current LE security and bonding parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`07`	`15`	None.
RSP	`C0`	`08`	`07`	`15`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GSLSP`	`0x0028`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`local_io_cap`	`C`	Local IO capabilities: • 0 = Display only • 1 = Display Yes/No • 2 = Keyboard only • 3 = No input, no output • 4 = Keyboard + Display
`uint8`	`oob_data`	`O`	OOB data present (locally) for the peer device: • 0x00 = No OOB data (Factory default) • 0x01 = OOB data is present, which is from the P-192 public key • 0x02 = OOB data is present, which is from the P-256 public key • 0x03 = OOB data is present, which is from the P-192 and P256 public key • 4 = OOB data unknown
`uint8`	`auth_req`	`A`	Authentication request (for local device) contain bonding and MITM information: • 0x00 = Not required – No bond • 0x01 = Required – General Bond • 0x04 = MITM required – Auth Y/N • 0x08 = LE secure connection, no MITM, no bonding • 0x09 = LE secure connection, no MITM, bonding • 0x0C = LE secure connection, MITM, no bonding • 0x0D = LE secure connection, MITM, bonding
`uint8`	`max_key_size`	`S`	Maximum encryption key size from 7 to 16 bytes.
`uint8`	`init_keys`	`I`	BLE Key types: • Bit0 (0x01) = Encryption information of peer device • Bit1 (0x02) = Identity key of the peer device • Bit2 (0x04) = Peer SRK • Bit3 (0x08) = Public key • Bit4 (0x10) = Master role security information • Bit5 (0x20) = Master device ID key • Bit6 (0x40) = Local CSRK key • Bit7 (0x80) = Link layer key
`uint8`	`resp_keys`	`R`	BLE Keys to be distributed (bit mask, these bits can be masked together): • Bit0 (0x01) = Encryption information of the peer device • Bit1 (0x02) = Identity key of the peer device • Bit2 (0x04) = Peer SRK • Bit3 (0x08) = Public key • Bit4 (0x10) = Master role security information • Bit5 (0x20) = Master device ID key • Bit6 (0x40) = Local CSRK key • Bit7 (0x80) = Link layer key

Related Commands:

- smp_set_le_security_parameters (SSLSP, ID=7/20)

7.2.7.18 *smp_set_parameters (SSMPP, ID=7/32)*

Writes the remote device OOB data for LE secure connection to the WICED module.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`81`	`07`	`20`	None.
RSP	`C0`	`02`	`07`	`20`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SSMPP`	`0x000F`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`flags`	`F`	SMP flags parameter: • Bit 0 (0x01) – Module device should automatically confirm to accept any pairing request. Note that this control bit must be always 1. If the value is changed to 0 there will take no effect on WICED platforms. • Bit 1 (0x02) – In passkey request event in pairing, the configuration fixed passkey should be used. Otherwise, the smp_passkey_entry_requested (PKE, ID=7/6) event will be received, and you should manually reply the passkey through the smp_send_passkeyreq_response (/PE, ID=7/6) API command. • Bit2 (0x04) – In the PIN code request event in pairing, the configuration fixed PIN code should be used. Otherwise, the smp_pin_entry_requested (BTPIN, ID=7/7) will be received, and you should manually reply the passkey through the smp_send_passkeyreq_response (/PE, ID=7/6) or smp_send_pinreq_response (/BTPIN, ID=7/17) API command. These SMP flags bits can be ORed together to set the parameter. The default value of this flag parameter is 0x07, that is the module device will try to automatically reply the pairing requests based on the reconfigured or set fixed passkey and PIN code.

Response Parameters:

None.

Related Commands:

None.

7.2.7.19 smp_get_parameters (GSMPP, ID=7/33)

Obtains remote device OOB data for LE secure connection paring.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`07`	`21`	None.
RSP	`C0`	`83`	`07`	`21`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GSMPP`	`0x0002`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`flags`	`F`	SMP flags parameter: • Bit 0 (0x01) – Module device should automatically confirm to accept any pairing request. • Bit 1 (0x02) – In passkey request event in pairing, the configuration fixed passkey should be used. Otherwise, the smp_passkey_entry_requested (PKE, ID=7/6) event will be received, and you should manually reply the passkey through the smp_send_passkeyreq_response (/PE, ID=7/6) API command. • Bit2 (0x04) – In the PIN code request event in pairing, the configuration fixed PIN code should be used. Otherwise, the smp_pin_entry_requested (BTPIN, ID=7/7) will be received, and you should manually reply the passkey through the smp_send_passkeyreq_response (/PE, ID=7/6) or smp_send_pinreq_response (/BTPIN, ID=7/17) API command. These SMP flags bits can be ORed together to set the parameter. The default value of this flag parameter is 0x07, that is the module device will try to automatically reply the pairing requests based on reconfigured or set fixed passkey and PIN code.

Related Commands:

None.

7.2.8 L2CAP Group (ID=8)

L2CAP methods relate to the Logical Link Control and Adaptation Protocol layer of the Bluetooth stack. These methods are used for working directly with low-level data transfer between two connected devices.

Note: L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The API methods described in this section will not function on devices with only 128K of flash.

Commands within this group are listed below:

- l2cap_le_register_psm (/LLERPSM, ID=8/30)
- l2cap_le_unregister_psm (/LLEURPSM, ID=8/31)
- l2cap_le_connect (/LLEC, ID=8/32)
- l2cap_le_disconnect (/LLEDISC, ID=8/33)
- l2cap_le_send_data (/LLESD, ID=8/34)

Events within this group are documented in L2CAP Group (ID=8).

7.2.8.1 *l2cap_le_register_psm (/LLERPSM, ID=8/30)*

Registers a BLE PSM value for L2CAP channel.

Note: Currently, only the dynamic LE_PSM can be used and registered.

- Fixed LE_PSMs are in the range 0x0001 – 0x007F
- Dynamic LE_PSM are in the range 0x0080 – 0x00FF
- The values 0x0000 and 0x0100 – 0xFFFF are reserved

For details on LE_PSM, see Part A, Section 4.22 Connection Request of the Bluetooth Core Specification v4.2, Vol 3.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`02`	`08`	`1E`	None.
RSP	`C0`	`04`	`08`	`1E`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/LLERPSM`	`0x0009`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`psm`	`P`	A dynamic PSM value that used for LE L2CAP channel.
`uint8`	`register_for_client`	`R`	Indicates the role the device acts in the LE L2CAP channel. • 0x00 – Registers the L2CAP for LE server usage, acts as server. • 0x01 – Registers the L2CAP for LE client usage, acts as client. Note that the return LE_PSM will be “virtual” PSM which is between 0x0100 and 0xFFFF, and these LE_PSM are for outgoing-only connections. Normally, this value should not be used.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`psm`	`P`	A dynamic PSM value (0x80 ~ 0xFF) that is used for LE L2CAP channel. Note that the returned PSM value may be different from the command set PSM value when the specific PSM value has been used.

Related Events:

None.

7.2.8.2 *l2cap_le_unregister_psm (/LLEURPSM, ID=8/31)*

Unregisters a registered LE PSM for a L2CAP channel.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`02`	`08`	`1F`	None.
RSP	`C0`	`04`	`08`	`1F`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/LLEURPSM`	`0x0009`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`psm`	`P`	The LE PSM value that has been registered for unregistering.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`psm`	`P`	The LE PSM value that has been unregistered.

Related Events:

None.

7.2.8.3 l2cap_le_connect (/LLEC, ID=8/32)

Sends a command to create a LE L2CAP channel between two devices.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`0E`	`08`	`20`	None.
RSP	`C0`	`04`	`08`	`20`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/LLEC`	`0x0009`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`psm`	`P`	The LE PSM value that has been registered.
`macaddr`	`bd_addr`	`A`	BT address of the remote device to which the L2CAP channel will be connected.
`uint8`	`addr_type`	`T`	BT address type of remote device. • 0x00 – Public address. • 0x01 – Random address.
`uint8`	`conn_mode`	`M`	BLE connection mode. • 0x01 – Slow connection scan mode. • 0x02 – Fast connection scan mode.
`uint16`	`rx_mtu`	`U`	RX MTU value, maximum 512 (=0x0200) bytes.
`uint8`	`req_security`	`S`	Security required. • 0x00 – Security not required. • 0x01 – Security required.
`uint8`	`req_encr_key_size`	`K`	Key size. Range of the value is 1 to 16.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`cid`	`C`	The channel Identifier (CID) value was generated by WICED stack to represent the created channel.

Related Events:

- l2cap_le_connected (LLEC, ID=8/20) – Occurs on the connected established with remote device

7.2.8.4 l2cap_le_disconnect (/LLEDISC, ID=8/33)

Sends a command to disconnect a LE L2CAP connection between two devices.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`02`	`08`	`21`	None.
RSP	`C0`	`02`	`08`	`21`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/LLEDISC`	`0x0009`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`cid`	`C`	Indicates the L2CAP connection to be disconnected.

Response Parameters:

None.

Related Events:

- l2cap_le_disconnected (LLEDISC, ID=8/21) – Occurs on the disconnected remote device

7.2.8.5 l2cap_le_send_data (/LLESD, ID=8/34)

Sends command to send a block data through the specific LE L2CAP connection.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`04`	`08`	`22`	Variable-length command payload.
RSP	`C0`	`02`	`08`	`22`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/LLESD`	`0x000C`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`cid`	`C`	Indicates the L2CAP connection that the block data has to be sent to.
`longuint8a`	`data`	`D`	A block data that should be sent through the L2CAP connection. **Note:** `longuint8a` data type requires two prefixed “length” bytes before binary parameter payload

Response Parameters:

None.

Related Events:

- l2cap_le_data_received (LLERX, ID=8/22) - Occurs on the remote device after data arrives
- l2cap_le_congestion_status (LLECNGS, ID=8/23) – Occurs on the local sending data queue is full

7.2.9 GPIO Group (ID=9)

GPIO methods relate to the physical pins on the module.

Commands within this group are listed below:

- gpio_query_logic (/QIOL, ID=9/1)
- gpio_query_adc (/QADC, ID=9/2)
- gpio_set_function (SIOF, ID=9/3)
- gpio_get_function (GIOF, ID=9/4)
- gpio_set_drive (SIOD, ID=9/5)
- gpio_get_drive (GIOD, ID=9/6)
- gpio_set_logic (SIOL, ID=9/7)
- gpio_get_logic (GIOL, ID=9/8)
- gpio_set_interrupt_mode (SIOI, ID=9/9)
- gpio_get_interrupt_mode (GIOI, ID=9/10)
- gpio_set_pwm_mode (SPWM, ID=9/11)
- gpio_get_pwm_mode (GPWM, ID=9/12)

Events within this group are documented in GPIO Group (ID=9).

GPIO API Method Guidelines

All GPIO methods follow the same basic argument pattern for port and pin selection and modification (except for those relating to PWM and ADC behavior, which use channel numbers for predefined pins). These API methods have the following common features:

- The initial `port` ("P") argument is a zero-based index for the port number.
- If present, the following `mask` ("M") argument is a bitmask for selecting which pins to modify.
- If present, all additional arguments are also bitmasks to apply to the selected pin range.
- SET command responses return the `affected` ("A") parameter, a bitmask showing which pins were affected.

Some ports do not have all pins physically exposed on the module. If you select any non-exposed pins, the command processor will silently ignore them (they will be cleared from the **`mask`** value and the **`affected`** return value).

Some pins have special functions assigned to them and enabled by default from the factory. If you select any special-function pins for modification, the command processor will store the new values in the general configuration settings, but the new values will not take effect unless you disable the special functions on those pins using the gpio_set_function (SIOF, ID=9/3) API command. See GPIO Reference for details on the pins that have these functions and how to disable them.

Using bitmasks for selection and new value application allows a single command to affect multiple pins in a complex way. Many single operations would otherwise require multiple commands. The example below illustrates how one gpio_set_logic (SIOL, ID=9/7) API command can set alternating logic state output levels across Port 2 on the CYBT-4130XX-02 EZ-BT module. Note that the CYBT-4130XX-02 EZ-BT module does not expose P2.0, P2.3, P2.4, P2.5, and P2.7, and the P2.1 has been fixed used as UART, so it cannot be operated also.

Step	Result							
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0

Command received:
 	
`SIOL,P=2,M=FF,L=AA`
`    Port:  2`
`    Pins:  FF (select all)`
	HIGH	LOW	HIGH	LOW	HIGH	LOW	HIGH	LOW

Step	Result							
`Logic: AA (0b10101010)`								
Command processor clears bits from the selection mask for any non-exposed pins to avoid unexpected behavior	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
	X		**X**	**X**	**X**		**X**	**X**

Logic states applied, response sent:
 	
`@R,000F,SIOL,0000,A=0044`
`Result:   0000 (success)`
`Affected: 0044 (0b01000100)`
	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0

	n/a	LOW	*n/a*	*n/a*	*n/a*	LOW	*n/a*	*n/a*

7.2.9.1 *gpio_query_logic (/QIOL, ID=9/1)*

Reads the active LOW/HIGH logic state of pins on the selected port.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability.

Note: This command returns immediate logic state of the pins on the specified port by reading that port's status register. This may be different from the pulled/driven states that you have configured using the gpio_set_logic (SIOL, ID=9/7) API command, due to external drive signals and strengths. To obtain the configured logic output settings rather than the immediate logic states, use the gpio_get_logic (GIOL, ID=9/8) API command. When the input capability of the GPIO is not enabled, the return value of the logic query is unknown.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`09`	`01`	None.
RSP	`C0`	`04`	`09`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/QIOL`	`0x0012`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`port`	`P*`	GPIO port (0-3)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`logic`	`L`	Pin logic mask (set bit for HIGH, clear for LOW)

Related Commands:

- gpio_set_logic (SIOL, ID=9/7) – Used to set output/pull logic state internally (may be overridden by external connections)
- gpio_get_logic (GIOL, ID=9/8) – Used to get output logic settings (not the same as actual logic levels)

Related Events:

- gpio_interrupt (INT, ID=9/1) – Includes port logic state at moment interrupt occurred

7.2.9.2 *gpio_query_adc (/QADC, ID=9/2)*

Read the immediate analog voltage level on the selected channel.

EZ-Serial provides dedicated ADC input pins (ADC0 to ADC19) for reading analog voltages. The ADC supports an input voltage range of 0 V minimum to 3.6 V maximum. Use this command to perform a single ADC conversion. Once the conversion completes, the module will transmit the result back in response parameters.

You can use the ADC pins as a normal digital GPIO, but performing an analog read with this command will reconfigure the pin back to a high-impedance analog input state.

See GPIO Pin Map for Supported Modules for a pin map table showing ADC pin assignment.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`09`	`02`	None.
RSP	`C0`	`08`	`09`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/QADC`	`0x001D`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`channel`	`N`*	ADC channel. The valid ADC channel number can be selected from the following table:
`uint8`	`reference`	`R`	Input voltage range selection. • 0 – Input voltage range is between 0 ~ 1.8 V (factory default) • 1 – Input voltage range is between 0 ~ 3.6 V

ADC Channel	**ADC Mapped to GPIO Pin**
0x00	P0
0x01	P1
0x02	P10
0x03	P16
0x04	P17
0x05	P28
0x06	P29
0x07	P34
0x08	P38
0x09	Internal VBAT VDDIO
0x0A	Internal VDDC
0x0B	Internal GBREF
0x0C	Internal REFGND

Note: Other GPIO pins that not listed in the above ADC channel map table cannot be used as ADC input.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`value`	`A`	Raw ADC conversion value, 0x0 – 0xFFFF (0 – 65535)
`uint32`	`uvolts`	`U`	Scaled ADC result in microvolts, 0x0 – 0xFFFFFFFF (0 – 78642000)

7.2.9.3 *gpio_set_function (SIOF, ID=9/3)*

Configures new special function assignment on selected pins.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment. See the general overview in GPIO Group (ID=9) for guidelines on how pin selection and configuration masks work.

NOTE: When an enabled GPIO function is disabled using this command, it will not change the current status of the GPIO pin. When a GPIO function is disabled using gpio_set_drive (SIOD, ID=9/5), gpio_set_logic (SIOL, ID=9/7), and gpio_set_interrupt_mode (SIOI, ID=9/9) API commands, the parameters of the GPIO configuration is changed, but the changes will not take effect immediately until the GPIO is enabled using this API command.

If a GPIO is not used again, before disabling the GPIO function using this command, it is recommended that you reconfigure the GPIO pin to default output value and disable the input and output capabilities of the GPIO pin by using gpio_set_drive (SIOD, ID=9/5) and gpio_set_logic (SIOL, ID=9/7) API commands.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`07`	`09`	`03`	None.
RSP	`C0`	`04`	`09`	`03`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SIOF`	`0x000F`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`port`	`P*`	GPIO port (0-3)
`uint16`	`mask`	`M*`	Pin selection mask (set bit to select pin for modification)
`uint16`	`enable`	`E`	Pin function mask (set bit to enable, clear to disable)
`uint16`	`drive`	`D`	Pin function drive mode (set bit for strong drive, clear for 5.6k pull)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`affected`	`A`	Affected pin mask (set bit for affected, clear for unaffected)

Related Commands:

- gpio_get_function (GIOF, ID=9/4)

7.2.9.4 *gpio_get_function (GIOF, ID=9/4)*

Gets the current special function assignment on selected pins.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment.

Note: This API command will take effect immediately only when the GPIO pin has been enabled using the gpio_set_function (SIOF, ID=9/3) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`09`	`04`	None.
RSP	`C0`	`06`	`09`	`04`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GIOF`	`0x0018`	GET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`port`	`P`*	GPIO port (0-3)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`enable`	`E`	Pin function mask (set bit indicates enabled, clear indicates disabled)
`uint16`	`drive`	`D`	Pin function drive mode (set bit indicates strong drive, clear indicates pull)

Related Commands:

- gpio_set_function (SIOF, ID=9/3)

7.2.9.5 gpio_set_drive (SIOD, ID=9/5)

Configures a new drive mode for selected pins.

Using the last four arguments of this command, you can configure every possible drive mode supported by the chipset. Table 7-3 describes each resulting drive mode from all combinations.

D	W	U	A	Drive mode
x	*x*	*x*	**1**	Analog input, high impedance
0	0	0	0	Digital input, high impedance
0	0	**1**	0	Digital input, pull-up
0	**1**	0	0	Digital input, pull-down
0	**1**	**1**	0	Digital input, same as high impedance. No pull-up and pull-down
1	0	0	0	Digital output, strong drive
1	0	**1**	0	Digital output, open-drain drives HIGH
1	**1**	0	0	Digital output, open-drain drives LOW
1	**1**	**1**	0	Digital output with pull-up and pull-down

Table 7-3. GPIO Drive Mode Table

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment. See the general overview in GPIO Group (ID=9), for guidelines on how pin selection and configuration masks work.

Note: This API command will take effect immediately only when the GPIO pin has been enabled using gpio_set_function (SIOF, ID=9/3) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`0B`	`09`	`05`	None.
RSP	`C0`	`04`	`09`	`05`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SIOD`	`0x000F`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	port	P*	GPIO port (0-3)
uint16	mask	M*	Pin selection mask (set bit to select pin for modification)
uint16	direction	D	Pin digital direction mask (set bit for output, clear for input)
uint16	pulldrive_down	W	Pin digital pull-down/drive-low mask (set bit to enable pull-down/drive-low, clear to disable)
uint16	pulldrive_up	U	Pin digital pull-up/drive-high mask (set bit to enable pull-up/drive-high, clear to disable)
uint16	analog	A	Pin analog mode mask (set bit to enable analog HIGH-Z input mode, clear for digital settings)

Response Parameters:

Data Type	Name	Text	Description
uint16	affected	A	Affected pin mask (set bit for affected, clear for unaffected)

Related Commands:

- gpio_get_drive (GIOD, ID=9/6)

7.2.9.6 gpio_get_drive (GIOD, ID=9/6)

Gets the current new drive mode for selected pins.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01	09	06	None.
RSP	C0	0A	09	06	None.

Text Info:

Text Name	Response Length	Category	Notes
GIOD	0x0026	GET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	port	P*	GPIO port (0-3)

Response Parameters:

Data Type	Name	Text	Description
uint16	direction	D	Pin digital direction mask (set bit for output, clear for input)
uint16	pulldrive_down	W	Pin digital pull-down/drive-low mask (set bit to enable pull-down/drive-low, clear to disable)
uint16	pulldrive_up	U	Pin digital pull-up/drive-high mask (set bit to enable pull-up/drive-high, clear to disable)
uint16	analog	A	Pin analog mode mask (set bit to enable analog HIGH-Z input mode, clear for digital settings)

Related Commands:

- gpio_set_drive (SIOD, ID=9/5)

7.2.9.7 gpio_set_logic (SIOL, ID=9/7)

Configures new output logic for selected pins.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment. See the general overview in GPIO Group (ID=9), for guidelines on pin selection and configuration masks.

Note: This command sets new drive/pull logic levels by writing to the data register of the selected port. Depending on the configured drive mode and external connections, the logic levels in the port status register may not match with the new configured state. Make sure you have configured the correct function behavior, drive mode, and external signals if the gpio_query_logic (/QIOL, ID=9/1) API command reports an unexpected state.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`05`	`09`	`07`	None.
RSP	`C0`	`04`	`09`	`07`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SIOL`	`0x000F`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`port`	`P*`	GPIO port (0-3)
`uint16`	`mask`	`M*`	Pin selection mask (set bit to select pin)
`uint16`	`logic`	`L`	Pin logic mask (set bit for HIGH, clear for LOW)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`affected`	`A`	Affected pin mask (set bit for affected, clear for unaffected)

Related Commands:

- gpio_get_logic (GIOL, ID=9/8)

7.2.9.8 gpio_get_logic (GIOL, ID=9/8)

Obtains the current output logic for selected pins.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment.

Note: This command does not return the immediate logic level of any pins. Instead, it returns the configured logic values set using the gpio_set_logic (SIOL, ID=9/7) API command. To obtain the actual logic states reported by the port status register, use the gpio_query_logic (/QIOL, ID=9/1) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`09`	`08`	None.
RSP	`C0`	`04`	`09`	`08`	None.

Text Info:

Text Name	Response Length	Notes
`GIOL`	`0x0011`	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`port`	`P*`	GPIO port (0-3)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`logic`	`L`	Pin logic mask (set bit for HIGH, clear for LOW)

Related Commands:

- gpio_query_logic (/QIOL, ID=9/1)
- gpio_set_logic (SIOL, ID=9/7)

7.2.9.9 gpio_set_interrupt_mode (SIOI, ID=9/9)

Configures new edge detection interrupt settings on selected pins.

Use this command to enable or disable edge change interrupts on available pins. All exposed pins support both rising and falling edge detection, reported via the gpio_interrupt (INT, ID=9/1) API event. Note, due to the platform limitation, each exposed pin only can be set once during the FW running time, and cannot be changed until reboot or reset.

Note: The pins used for interrupt must have capacitors to stabilize the signal, otherwise many interrupts will be triggered shortly, which will cause interrupt HW module to stop working until reboot. Once the pins are registered successfully, you cannot unregister; registration is meant to be a start activity. To stop receiving notifications, reconfigure the pin and disable the interrupt using gpio_set_function (SIOF, ID=9/3), gpio_set_drive (SIOD, ID=9/5), and gpio_set_interrupt_mode (SIOI, ID=9/9) API commands.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment. See the general overview in GPIO Group (ID=9), for guidelines on pin selection and configuration masks.

Note: Pins with certain special functions enabled will generate interrupts internally for processing. These interrupts occur regardless of whether you enable or disable them with this API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`07`	`09`	`09`	None.
RSP	`C0`	`04`	`09`	`09`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SIOI`	`0x000F`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`port`	`P*`	GPIO port (0-3)
`uint16`	`mask`	`M*`	Pin selection mask (set bit to select pin)
`uint16`	`rising`	`R`	Rising-edge interrupts (set bit to enable, clear to disable)
`uint16`	`falling`	`F`	Falling-edge interrupts (set bit to enable, clear to disable)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`affected`	`A`	Affected pin mask (set bit for affected, clear for unaffected)

Related Commands:

- gpio_get_interrupt_mode (GIOI, ID=9/10)

Related Events:

- gpio_interrupt (INT, ID=9/1)

7.2.9.10 gpio_get_interrupt_mode (GIOI, ID=9/10)

Obtains the current edge detection interrupt settings on selected pins.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment.

Note: This API command will take effect immediately only when the GPIO pin has been enabled using the gpio_set_function (SIOF, ID=9/3) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`09`	`0A`	None.
RSP	`C0`	`06`	`09`	`0A`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GIOI`	`0x0018`	GET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`port`	`P`*	GPIO port (0-3)

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`rising`	`R`	Rising-edge interrupts (set bit to enable, clear to disable)
`uint16`	`falling`	`F`	Falling-edge interrupts (set bit to enable, clear to disable)

Related Commands:

- gpio_set_interrupt_mode (SIOI, ID=9/9)

Related Events:

- gpio_interrupt (INT, ID=9/1)

7.2.9.11 gpio_set_pwm_mode (SPWM, ID=9/11)

Configures new PWM output behavior for selected channel on selected GPIO pin.

EZ-Serial provides six dedicated PWM output pins (PWM0, PWM1, PWM2, PWM3, PWM4, and PWM5). You can enable PWM output on any of the six PWM channels using this API command. PWM channels are controlled via independent 24 MHz clocks, and each can use separate input clock frequency, PWM output frequency, and duty cycle percentage settings for complete flexibility.

Enabling PWM on each channel means that you cannot use that pin for other generic I/O. To return a PWM channel pin to standard functionality, use the gpio_set_pwm_mode (SPWM, ID=9/11) API command to disable PWM output on that pin. See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment.

To reconfigure a started PWM, use the gpio_set_pwm_mode (SPWM, ID=9/11) API command with argument “E” set to 0 to disable PWM output on that pin first, then reconfigure and start the PWM on the same PWM channel and GPIO.

Note: Enabling PWM output on one or more channels will automatically prevent the CPU from entering Deep Sleep under any circumstances. This happens because the high-frequency clock required to generate the PWM signal cannot operate while the CPU is in Deep Sleep. To allow Deep Sleep mode again, you must disable all PWM outputs. See Managing Sleep States for more details.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	0C	09	0B	None.
RSP	C0	02	09	0B	None.

Text Info:

Text Name	Response Length	Category	Notes
SPWM	0x000A	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	channel	N*	Channel number (0-5)
uint8	gpio	G	GPIO pin number (0-38(0x26)) to which the PWM output signal will be routed. This GPIO pin number should not conflict with other function which has taken this pin.
uint8	enable	E	Control flags to the PWM channel. • Bit 0 (0x01) = Enable (1) or disable (0) this PWM channel output. • Bit 1 (0x02) = Invert the PWM channel output signal. 1 - Invert; 0 - Do not invert. **Note**: These bits can be ORed together. Other bits that are reserved and not used must be set to 0.
uint32	clock_in	C	Input clock frequency for driving the PWM module. The input frequency clock is divided from internal ACKL1 24 MHz clock. The value range must be between 1 ~ 24000000.
uint32	freq_out	F	Desire PWM output frequency. **Note**: The PWM output frequency must be less than the input clock frequency. The value range must be between 1 ~24000000.
uint8	percentage	P	PWM output duty cycle in percentage (0 ~ 100).

Response Parameters:

None.

Related Commands:

- gpio_get_pwm_mode (GPWM, ID=9/12)

7.2.9.12 gpio_get_pwm_mode (GPWM, ID=9/12)

Obtains the current PWM output behavior for selected channel.

See GPIO Pin Map for Supported Modules for a pin map table showing pin availability and default assignment.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01	09	0C	None.
RSP	C0	0D	09	0C	None.

Text Info:

Text Name	Response Length	Category	Notes
GPWM	0x002F	GET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	channel	N*	Channel number (0-5)

Response Parameters:

Data Type	Name	Text	Description
uint8	gpio	G	GPIO pin number to which the PWM output signal will be routed .
uint8	enable	E	Control flags to the PWM channel. • Bit 0 (0x01) = Enable (1) or disable (0) this PWM channel output. • Bit 1 (0x02) = Invert the PWM channel output signal. 1-Invert; 0-Do not invert. **Note**: These bits can be ORed together. Other bits that are reserved and not used must be set to 0.
uint32	clock_in	C	Input clock frequency for driving the PWM module. The input frequency clock is divided from internal ACKL1 24 MHz clock. The value range must be between 1 ~ 24000000.
uint32	freq_out	F	Desire PWM output frequency. **Note**: The PWM output frequency must be less than the input clock frequency. The value range must be between 1 ~24000000.
uint8	percentage	P	PWM output duty cycle in percentage (0 ~ 100).

Related Commands:

- gpio_set_pwm_mode (SPWM, ID=9/11)

7.2.10 CYSPP Group (ID=10)

CYSPP methods relate to the CYSPP Profile.

Commands within this group are listed below:

- p_cyspp_check (.CYSPPCHECK, ID=10/1)
- p_cyspp_start (.CYSPPSTART, ID=10/2)
- p_cyspp_set_parameters (.CYSPPSP, ID=10/3)
- p_cyspp_get_parameters (.CYSPPGP, ID=10/4)
- p_cyspp_set_client_handles (.CYSPPSH, ID=10/5)
- p_cyspp_get_client_handles (.CYSPPGH, ID=10/6)

Events within this group are documented in CYSPP Group (ID=10).

For more details and examples related to CYSPP operation, see:

- Using CYSPP Mode
- Configuring the CYSPP Data Mode Sleep Level
- Cable Replacement Examples with CYSPP

7.2.10.1 p_cyspp_check (.CYSPPCHECK, ID=10/1)

The CYSPP central checks whether a connected peer device includes support for the CYSPP service.

This command requires an active connection, and performs a service and descriptor discovery to identify the required elements for CYSPP operation. If detection completes successfully, EZ-Serial will generate the p_cyspp_status (.CYSPP, ID=10/1) API event with the "CYSPP peer support verified" bit set. However, it will not automatically enter CYSPP mode even upon verifying remote peer compatibility.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0A`	`01`	None.
RSP	`C0`	`02`	`0A`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.CYSPPCHECK`	`0x0011`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- p_cyspp_start (.CYSPPSTART, ID=10/2)
- p_cyspp_set_parameters (.CYSPPSP, ID=10/3)
- p_cyspp_set_client_handles (.CYSPPSH, ID=10/5)

Related Events:

- p_cyspp_status (.CYSPP, ID=10/1)

7.2.10.2 p_cyspp_start (.CYSPPSTART, ID=10/2)

Activates CYSPP operation.

Use this command to start CYSPP via the API protocol, rather than asserting the **CYSPP** pin or configuring automatic start with the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command. EZ-Serial will choose the role used for CYSPP operation based on the logic state of the **CP_ROLE** pin, or if that pin is floating, the **`role`** setting configured with the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command.

See CYSPP State Machine for details on the movement of CYSPP between different operational states.

Note: To execute this command successfully, make sure that the device's advertising and scanning operations are stopped by gap_stop_adv (/AX, ID=4/9) and gap_stop_scan (/SX, ID=4/11) commands. When advertising or scanning is in progress, the CYSPP will not rerun these operations, and the existing advertising or scanning state will be kept. This might cause the operation to timeout quickly or advertise or scan in low frequency. Also, the device might have been scanned by a previous scan operation. Therefore, CYSPP will be unable to receive the same device scan response data anymore, by default.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0A`	`02`	None.

	Type	Length	Group	ID	Notes
RSP	`C0`	`02`	`0A`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.CYSPPSTART`	`0x0011`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- p_cyspp_check (.CYSPPCHECK, ID=10/1)
- p_cyspp_set_parameters (.CYSPPSP, ID=10/3)
- p_cyspp_set_client_handles (.CYSPPSH, ID=10/5)

Related Events:

- p_cyspp_status (.CYSPP, ID=10/1)

7.2.10.3 p_cyspp_set_parameters (.CYSPPSP, ID=10/3)

Configures new CYSPP behavior settings.

Use this command to control the behavior of CYSPP. See Using CYSPP Mode and Cable Replacement Examples with CYSPP for example usage and explanations on how the settings affect the behavior of CYSPP.

Note: Disabling CYSPP with this API method will cause EZ-Serial to hide the relevant GATT database attributes from client discovery. All other visible attributes will remain the same with their original handles, but those inside the CYSPP attribute range will be hidden and will be unusable by connected clients. This will remain in effect until you enable the profile again or assert the CYSPP pin.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`13`	`0A`	`03`	None.
RSP	`C0`	`02`	`0A`	`03`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.CYSPPSP`	`0x000E`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`enable`	`E`	Enable CYSPP profile: • 0 = Disable • 1 = Enable • 2 = Enable + auto-start (factory default)
`uint8`	`role`	`G`	GAP role to use: • 0 = Peripheral/server (factory default) • 1 = Central/client
`uint16`	`company`	`C`	Company ID value for automatic advertisement payload Manufacturer Data. **Note:** Factory default is 0x0131 (Cypress Semiconductor).
`uint32`	`local_key`	`L`	Local connection key to be present while advertising (peripheral role)

Data Type	Name	Text	Description
`uint32`	`remote_key`	`R`	Remote connection key to search for while scanning (central role)
`uint32`	`remote_mask`	`M`	Bitmask for bits in remote key which must match for a central-role connection
`uint8`	`sleep_level`	`P`	Sleep enabled or disabled while connected with open CYSPP data pipe: • 0 = Sleep disabled • 1 = Sleep enabled
`uint8`	`server_security`	`S`	CYSPP server security requirement to allow writing CYSPP data from a client: • 0 = No security required • 1 = Encryption required • 2 = Bonding required • 3 = Encryption and bonding required
`uint8`	`client_flags`	`F`	Client GATT usage flags while operating CYSPP in the central role • Bit 0 (0x01) = Use acknowledged data transfer when CYSPP is established automatically. By default, this bit is cleared, so the unacknowledged data transfer will be used. • Bit 1 (0x02) = Enable CYSPP RX flow control **Note:** Factory default is 0x02 (RX flow only); these bits are exclusive. Changes to this value will take effect immediately if CYSSP has been in active.

Response Parameters:

None.

Related Commands:

- p_cyspp_start (.CYSPPSTART, ID=10/2)
- p_cyspp_get_parameters (.CYSPPGP, ID=10/4)
- p_cyspp_set_client_handles (.CYSPPSH, ID=10/5)

Related Events:

- gap_adv_state_changed (ASC, ID=4/2) – May occur if CYSPP is set to start automatically in peripheral role
- mgap_scan_state_changed (SSC, ID=4/3) – May occur if CYSPP is set to start automatically in central role
- p_cyspp_status (.CYSPP, ID=10/1)

Example Usage:

- See Using CYSPP Mode
- See Configuring the CYSPP Data Mode Sleep Level
- See Cable Replacement Examples with CYSPP

7.2.10.4 *p_cyspp_get_parameters (.CYSPPGP, ID=10/4)*

Obtains the current CYSPP behavior settings.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`0A`	`04`	None.
RSP	`C0`	`15`	`0A`	`04`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.CYSPPGP`	`0x004F`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`enable`	`E`	Enable CYSPP profile: • 0 = Disable • 1 = Enable • 2 = Enable + auto-start (factory default)
`uint8`	`role`	`G`	GAP role to use: • 0 = Peripheral/server (factory default) • 1 = Central/client
`uint16`	`company`	`C`	Company ID value for automatic advertisement packet payload Manufacturer Data. **Note:** Factory default is 0x0131 (Cypress Semiconductor)
`uint32`	`local_key`	`L`	Local connection key to be present while advertising (peripheral role)
`uint32`	`remote_key`	`R`	Remote connection key to be searched while scanning (central role)
`uint32`	`remote_mask`	`M`	Bitmask for bits in remote key which must match for a central-role connection
`uint8`	`sleep_level`	`P`	Maximum Sleep level while connected with open CYSPP data pipe: • 0 = Sleep disabled • 1 = Normal Sleep when possible (factory default) • 2 = Deep Sleep when possible **Note:** System-wide Sleep overrides this if it is set to a lower level.
`uint8`	`server_security`	`S`	CYSPP server security requirement for writing CYSPP data from a client: • 0 = No security required • 1 = Encryption required • 2 = Bonding required • 3 = Encryption and bonding required
`uint8`	`client_flags`	`F`	Client GATT usage flags while operating CYSPP in the central role • Bit 0 (0x01) = Use acknowledged data transfers • Bit 1 (0x02) = Enable CYSPP RX flow control **Note:** Factory default is 0x02 (RX flow only)

Related Commands:

- p_cyspp_set_parameters (.CYSPPSP, ID=10/3)

7.2.10.5 p_cyspp_set_client_handles (.CYSPPSH, ID=10/5)

Configures a new preset attribute handle for CYSPP central/client operation.

Use this command to manually specify the remote GATT server handles for data and optional RX flow control. If you know these handles in advance and can guarantee that they will not change, then configuring them here causes EZ-Serial to skip the GATT discovery process that normally occurs during CYSPP client operation.

EZ-Serial's internal GATT structure has the following attribute handles:

	Acknowledged Data	Unacknowledged Data	RX Flow Control
Value	`0x000E`	`0x0011`	`0x0014`
Configuration	`0x000F`	`0x0012`	`0x0015`

To disable preset attribute handles and allow automatic discovery for every CYSPP client connection, set all four handle values to 0 (factory default).

Note: EZ-Serial uses the `data_value_handle` and `data_cccd_handle` settings for client-role data pipe setup and data transfer, regardless of whether you have configured the `client_flags` setting that requires acknowledged data to use the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command. In other words, if you configure unacknowledged data transfers (factory default), set these values to the unacknowledged handles; or, if you configure acknowledged data transfers, you should set these values to the acknowledged handles. When the `client_flags` value is changed, the client handles will be discarded and reset to default 0 automatically.

Note: These settings only apply when operating CYSPP in the central/client role. They have no impact on CYSPP peripheral/server behavior.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`08`	`0A`	`05`	None.
RSP	`C0`	`02`	`0A`	`05`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.CYSPPSH`	`0x000E`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint16`	`data_value_handle`	`A`	Data characteristic value handle
`uint16`	`data_cccd_handle`	`B`	Data characteristic configuration handle
`uint16`	`rxflow_value_handle`	`C`	RX flow control characteristic value handle
`uint16`	`rxflow_cccd_handle`	`D`	RX flow control characteristic configuration handle

Response Parameters:

None.

Related Commands:

- p_cyspp_start (.CYSPPSTART, ID=10/2)
- p_cyspp_set_parameters (.CYSPPSP, ID=10/3)

Related Events:

- p_cyspp_status (.CYSPP, ID=10/1)

7.2.10.6 p_cyspp_get_client_handles (.CYSPPGH, ID=10/6)

Obtains the current preset attribute handles for CYSPP central/client operation.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0A`	`06`	None.
RSP	`C0`	`0A`	`0A`	`06`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.CYSPPGH`	`0x002A`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint16`	`data_value_handle`	`A`	Data characteristic value handle
`uint16`	`data_cccd_handle`	`B`	Data characteristic configuration handle
`uint16`	`rxflow_value_handle`	`C`	RX flow control characteristic value handle
`uint16`	`rxflow_cccd_handle`	`D`	RX flow control characteristic configuration handle

Related Commands:

- p_cyspp_set_client_handles (.CYSPPSH, ID=10/5)

7.2.10.7 p_cyspp_set_packetization (.CYSPPSK, ID=10/7)

Controls how the incoming serial data from an external host is packetized for CYSPP transmission.

Use this command to control whether or how incoming serial data is assembled into specific packets for transmission to the remote peer over a CYSPP connection. Packetization does not affect the content or ordering of serial data in any way, but only affects certain buffering and transmission timing.

Note: CYSPP packetization does not affect any outgoing UART serial data (module-to-host), nor does it affect incoming serial data while in command mode (that is, the CYSPP data pipe is not open). It impacts only the incoming serial data while CYSPP data mode is active.

At 115200 baud, a single byte takes about 80 us to transfer. EZ-Serial checks for new bytes at least every 20 us and processes the available bytes. Due to this, a continuous serial byte stream from an external host may be delivered to a remote CYSPP peer with multiple GATT transfers even if all the data could fit in a single packet (for example, two bytes sent as two single-byte transfers). Although the data will always be delivered completely and in the correct order, this results in potentially unnecessary complexity on the receiving end, which must buffer and combine incoming data if it does not handle it as a continuous data stream.

To address this behavior, EZ-Serial provides this API command to control incoming data packetization. There are five different modes:

- Mode 0: Immediate

  This mode reads and transmits data as quickly as possible, always sending as much data as is available when the BLE stack allows a new transmission. In this mode, the first byte or two bytes of a new transmission will usually be sent in a single packet even if more data is arriving at the same time.

  The *[wait]* and *[length]* settings are irrelevant in this mode.

- Mode 1: Anticipate (factory default with 5 ms wait and 20-byte length)

  This mode waits up to *[wait]* milliseconds in anticipation for at least *[length]* bytes to arrive from the external host. If the target byte count is reached before the wait time expires, all available bytes will be transmitted immediately. If the configured wait time expires before reaching the target byte count, all available bytes will be transmitted at that time. Anticipate mode is suitable for most general operations and will not negatively impact throughput if the incoming serial data arrives fast enough to keep the UART receive buffer full.

  The *[wait]* setting must be between 1 and 255. The *[length]* setting must be between 1 and 128, which is the internal UART RX software buffer size.

- Mode 2: Fixed

  This mode waits indefinitely until at least *[length]* bytes have been read, then transmits exactly that many bytes. Fixed mode is best used in cases where the host sends chunks of data which are always of the same size. Setting a *[length]* value that is greater than the GATT MTU payload size will result in multiple transmissions once all data has been buffered. For example, a fixed packet length of 32 bytes with the default GATT MTU size of 23 bytes (usable payload size of 20 bytes) will result in one 20-byte packet followed by one 12-byte packet. The MTU depends on the value negotiated by the client after connection.

  The *[length]* setting must be between 1 and 128, which is the internal UART RX software buffer size. The *[wait]* setting is irrelevant in this mode.

- Mode 3: Variable

  This mode requires an additional *length* value from the host before each packet to indicate the number of bytes to expect. EZ-Serial consumes this byte (it is *not* transmitted to the remote peer), and then waits until exactly that many bytes to have been read before transmitting them. Variable mode is suitable for applications that require packets of differing lengths and which can accommodate an extra transmitted byte from the host indicating each packet's length.

  For example, the host can send [***04*** 61 62 63 64] to transmit the 4-byte ASCII string "abcd" to the remote peer in a single packet. Or, the host can send [***05*** 61 62 63 64 65 ***03*** 66 67 68] to transmit "abcdefgh" in two packets ("abcde" followed by "def").

  The prefixed packet length byte must not be greater than 128. Values greater than this will be capped at 128. The *[wait]* and *[length]* settings are irrelevant in this mode.

  This mode is mainly designed to be used by the host program, not to be used in the terminal. The preceding variable length byte cannot be input correctly through the terminal.

  When you really want to send data through the terminal, set the mode value set to 0x83. That is, bit7 of the mode must be set to indicate that the input data is in hexadecimal format, and each byte must be supplied in two hexadecimal characteristics, for example, 0 must be 00, 1 must be 01, and so on. The EZ-Serial will parse internally and convert the hexadecimal string into correct alphanumeric characteristics. The peer device will only receive the converted data, and not the input hexadecimal string.

- Mode 4: End-of-packet

  This mode buffers data until the configured end-of-packet byte is encountered in the data stream, or until either the MTU payload size or UART RX buffer has filled. End-of-packet (EOP) mode allows variable-length packets without knowing in advance the length of the packet.

  The EOP byte defaults to 0x0D (for example, 0A – LF; 0D - CR. The carriage return (CR) byte, often expressed as '`\r`' in code; The line feed (LF) byte, often expressed as '\n' in the code). However, you can change it to any value between 0x00 and 0xFF. When the EOP byte occurs in the data stream, all buffered data up to that point including the EOP byte itself will be transmitted to the remote side.

  In this mode, EZ-Serial will also transmit buffered data under two other conditions:

  1. If the GATT MTU payload size is less than the UART RX buffer size (128 bytes) and enough data is buffered to fill a single GATT packet, one packet's worth of data will be transmitted. The default GATT MTU is 23 bytes with a usable payload size of 20 bytes.
  2. If the GATT MTU payload size is greater than the UART RX buffer size (128 bytes) and the RX buffer is full, 128 bytes of data will be transmitted. This can only occur in cases where the connected client has negotiated a GATT MTU greater than 131 bytes (actual transmit payload is MTU - 3 bytes).

For the "Anticipate" mode (1), you must consider the UART baud rate when choosing the *[wait]* and *[length]* values. A 5 ms wait time is suitable for a 20-byte target length at 115200 baud, but this is not enough time to read in 20 bytes at 9600 baud (for example). If you change the baud rate, make sure to choose a *[wait]* value that allows the target packet length to be filled under normal operating conditions. Table 7-4 contains "safe" wait values for 20-byte packets at common baud rates for reference.

Baud Rate	Single Bit Duration	20 Bytes at 8/N/1 (200 Bits)	Safe Wait Value Example
9600	104 us	~21 ms	32 ms (0x20)
38400	26.1 us	~5.2 ms	10 ms (0x0A)
57600	17.4 us	~3.5 ms	5 ms (0x05)
115200	8.68 us	~1.7 ms	5 ms (0x05)
230400	4.34 us	868 us	2 ms (0x02)
460800	2.17 us	434 us	1 ms (0x01)
921600	1.09 us	217 us	1 ms (0x01)

Table 7-4. Common UART Timing for 20-Byte Packets

The single-bit duration for any baud rate can be calculated using this equation:

```
Bit time = 1,000,000 us / [baud]
```

Standard UART settings of 8 data bits, no parity, and 1 stop bit yield a total of 10 bits per byte. For a 20-byte packet, this requires an allowance for 200 bits.

Note: If the packet length used in Anticipate, Fixed, Variable, or End-of-Packet mode exceeds the GATT MTU usable payload size (20 bytes on many platforms), the packets will be broken apart to fit within this lower-level constraint. For example, using Fixed mode with *[length]* set to 32 bytes will result in two transmitted packets each time the target length is reached: first a 20-byte packet and then a 12-byte packet.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	03	0A	07	None.
RSP	C0	02	0A	07	None.

Text Info:

Text Name	Response Length	Category	Notes
.CYSPPSK	0x000E	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	mode	M	Packetization mode (bit6-0): • 0 = Immediate: Transmit incoming data as soon as possible • 1 = Anticipate: Wait for a short time to attempt a minimum buffer threshold • 2 = Fixed: Buffer and send packets of exactly one size • 3 = Variable: Specify the size of every packet with a prefixed length byte • 4 = End-of-packet: Transmit data when specific byte occurs in stream **Note:** Factory default is 1 (Anticipate) Use hexadecimal string input (bit7): • 0 = Indicates that the input value byte is raw value byte. There is no need to convert it from hexadecimal string to real value. (Factory default) • 1 = Indicates that the input data in hexadecimal string format when the packetization mode is set to 3, and each byte must be input in two hexadecimal characteristics, so the string length must be even. **Note:** This bit7 is only valid when the packetization mode is set to 3. In other modes, it will be ignored.
uint8	wait	W	Anticipation delay (ms), used only in “Anticipate” mode: • Minimum = 0x01 (1 ms) • Maximum = 0x80 (128 bytes) **Note:** Factory default is 0x5 (5 ms)
uint8	length	L	Fixed/anticipated packet length (bytes), used only in “Anticipate” or “Fixed” mode: • Minimum = 0x01 (1 byte) • Maximum = 0x80 (128 bytes) **Note:** Factory default is 0x14 (20 bytes, standard GATT MTU)
uint8	eop	E	The EOP byte when mode is set to 4. The default value is to 0x0D.

Response Parameters:

None.

Related Commands:

- p_cyspp_get_packetization (.CYSPPGK, ID=10/8)

7.2.10.8 p_cyspp_get_packetization (.CYSPPGK, ID=10/8)

Obtains the current CYSPP packetization settings.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0A`	`08`	None.
RSP	`C0`	`05`	`0A`	`08`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.CYSPPGK`	`0x0022`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`mode`	`M`	Packetization mode (bit6-0): • 0 = Immediate: Transmit incoming data as soon as possible • 1 = Anticipate: Wait for a short time to attempt a minimum buffer threshold • 2 = Fixed: Buffer and send packets of exactly one size • 3 = Variable: Specify the size of every packet with a prefixed length byte • 4 = End-of-packet: Transmit data when specific byte occurs in stream **Note:** Factory default is 1 (Anticipate) Use hexadecimal string input (bit7): • 0 = Indicates the input value byte is raw value byte. There is no need to convert it from hexadecimal string to real value. (Factory default) • 1 = Indicates the input data in hexadecimal string format when the packetization mode is set to 3, and each byte must be input in two hexadecimal characteristics, so the string length must be even. **Note:** This bit7 is valid only when the packetization mode is set to 3. In other modes, it will be ignored.
`uint8`	`wait`	`W`	Anticipation delay (ms), used only in "Anticipate" mode: • Minimum = 0x01 (1 ms) • Maximum = 0x80 (128 bytes) **Note:** Factory default is 0x5 (5 ms)
`uint8`	`length`	`L`	Fixed/anticipated packet length (bytes), used only in "Anticipate" and "Fixed" modes: • Minimum = 0x01 (1 byte) • Maximum = 0x80 (128 bytes) **Note:** Factory default is 0x14 (20 bytes, standard GATT MTU)
`uint8`	`eop`	`E`	The EOP byte when mode is set to 4. The default value is to 0x0D.

Related Commands:

- p_cyspp_set_packetization (.CYSPPSK, ID=10/7)

7.2.11 CYCommand Group (ID=11)

CYCommand methods relate to CYCommand remote configuration channel behavior.

Commands within this group are listed below:

- p_cycommand_set_parameters (.CYCOMSP, ID=11/1)
- p_cycommand_get_parameters (.CYCOMGP, ID=11/2)

Events within this group are documented in CYCommand Group (ID=11).

7.2.11.1 p_cycommand_set_parameters (.CYCOMSP, ID=11/1)

Configures a new CYCommand remote configuration channel behavior.

The CYCommand profile allows a remote device to configure and control the module using the API protocol. CYCommand supports both text mode and binary mode, with the same formats and data flow requirements. Opening a CYCommand session logically disconnects the API parser from the wired serial interface and provides a GATT-based channel instead.

While CYCommand data mode is active, you cannot send any API commands over the wired serial interface. EZ-Serial will buffer incoming API data (up to 136 bytes) and release it for parsing only after closing the CYCommand session. However, you can allow real-time outgoing response and event data that occurs during a CYCommand session, using the `hostout` argument of this API command. This allows you to monitor remote activity from a local wired host device.

Note: Disabling CYCommand with this API will cause EZ-Serial to hide the relevant GATT database attributes from client discovery. All other visible attributes will remain the same with their original handles, but those inside the CYCommand attribute range will be hidden and will be unusable by connected clients. This will remain in effect until you enable the profile again.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	08-1C	0B	01	Variable-length command payload, minimum of 8 (0x08), maximum of 28 (0x1C)
RSP	C0	02	0B	01	None.

Text Info:

Text Name	Response Length	Category	Notes
.CYCOMSP	0x000E	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	enable	E	Enable CYCommand profile: • 0 = Disable • 1 = Enable (factory default)
uint8	hostout	H	Host output while CYCommand data channel is active: • 0 = All responses and events are suppressed • 1 = Only responses are shown, events are suppressed • 2 = Only events are shown, responses are suppressed • 3 = Both responses and events are shown (factory default)
uint16	timeout	T	Access timeout after boot, in seconds (always set to 0 in the current release) • 0 = Disable
uint8	safemode	F	Enforce safe mode (no remote lockout) • 0 = Disable (factory default) • 1 = Enable
uint8	challenge	C	CYCommand challenge type • 0 = None (factory default) • 1 = Passphrase. Maximum 20 characters passphrase can be supported.

Data Type	Name	Text	Description
`uint8`	`security`	`S`	CYCommand security requirement to allow writing API protocol data from a client: • 0 = No security required (factory default) • 1 = Encryption required • 2 = Bonding required • 3 = Encryption and bonding required
`uint8a`	`secret`	`R`	CYCommand secret (0-20 bytes) **Note**: `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- p_cycommand_get_parameters (.CYCOMGP, ID=11/2)

Related Events:

- p_cycommand_status (.CYCOM, ID=11/1)

Example Usage:

- See Remote Control Examples with CYCommand

7.2.11.2 p_cycommand_get_parameters (.CYCOMGP, ID=11/2)

Obtains the current CYCommand remote configuration channel behavior.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0B`	`02`	None.
RSP	`C0`	`0A-1E`	`0B`	`02`	Variable-length response payload, minimum of 10 (0x0A), maximum of 30 (0x1E).

Text Info:

Text Name	Response Length	Category	Notes
`.CYCOMGP`	`0x0031-0x0059`	GET	Variable-length response payload, minimum of 49 (0x31), maximum of 89 (0x59)

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`enable`	`E`	Enable CYCommand profile: • 0 = Disable • 1 = Enable (factory default)
`uint8`	`hostout`	`H`	Host output while CYCommand data channel is active: • 0 = Responses and events suppressed • 1 = Responses shown, events suppressed • 2 = Responses suppressed, events shown • 3 = Responses and events shown (factory default)
`uint16`	`timeout`	`T`	Access timeout after boot, in seconds (always set to 0 in the current release) • 0 = Disable
`uint8`	`safemode`	`F`	Enforce safe mode (no remote lockout) • 0 = Disable (factory default) • 1 = Enable

Data Type	Name	Text	Description
`uint8`	`challenge`	`C`	CYCommand challenge type • 0 = None (factory default) • 1 = Passphrase
`uint8`	`security`	`S`	CYCommand security requirement to allow writing API protocol data from a client: • 0 = No security required (factory default) • 1 = Encryption required • 2 = Bonding required • 3 = Encryption and bonding required
`uint8a`	`secret`	`R`	CYCommand secret (0-20 bytes) **Note**: `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Related Commands:

- p_cycommand_set_parameters (.CYCOMSP, ID=11/1)

7.2.11.3 iBeacon Group (ID=12)

iBeacon methods relate to iBeacon setup and operation.

Commands within this group are listed below:

- p_ibeacon_set_parameters (.IBSP, ID=12/1)
- p_ibeacon_get_parameters (.IBGP, ID=12/2)

Events within this group are documented in iBeacon Group (ID=12).

7.2.11.4 p_ibeacon_set_parameters (.IBSP, ID=12/1)

Configures a new iBeacon behavior.

For details on iBeacon broadcasting, see the example usage and the official documentation from Apple.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`1A`	`0C`	`01`	None
RSP	`C0`	`02`	`0C`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.IBSP`	`0x000B`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`enable`	`E`	Enable iBeacon broadcast: • 0 = Disable (factory default) • 1 = Enable • 2 = Enable + auto-start
`uint16`	`interval`	`I`	Advertisement interval for iBeacon broadcasting (625 µs units): • Minimum = 0x00A0 (160 * 0.625 ms = 100 ms, factory default) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
`uint16`	`company`	`C`	Company ID value in broadcast packet payload Manufacturer Data: • Factory default is 0x0131 (Cypress Semiconductor)
`uint8`	`major`	`J`	iBeacon 16-bit major value: • Factory default is 0x0001
`uint8`	`minor`	`N`	iBeacon 16-bit minor value:

Data Type	Name	Text	Description
			• Factory default is 0x0001
`uint8a`	`uuid`	`U`	iBeacon UUID (must contain 16 bytes of data): • Factory default is E2C56DB5-DFFB-48D2-B060-D0F5A71096E0 (AirLocate) **Note**: `uint8a` data type requires one prefixed “length” byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- p_ibeacon_get_parameters (.IBGP, ID=12/2)

Related Events:

- gap_adv_state_changed (ASC, ID=4/2) – May occur if iBeacon is set to start automatically

Example Usage:

- See Configuring iBeacon Transmissions

7.2.11.5 p_ibeacon_get_parameters (.IBGP, ID=12/2)

Sets up iBeacon behavior.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0C`	`02`	None.
RSP	`C0`	`1C`	`0C`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.IBGP`	`0x004F`	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`enable`	`E`	Enable iBeacon broadcast: • 0 = Disable (factory default) • 1 = Enable • 2 = Enable + auto-start
`uint16`	`interval`	`I`	Advertisement interval for iBeacon broadcasting (625 µs units): • Minimum = 0x00A0 (160 * 0.625 ms = 100 ms, factory default) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
`uint16`	`company`	`C`	Company ID value in broadcast packet payload Manufacturer Data: • Factory default is 0x0131 (Cypress Semiconductor)
`uint8`	`Major`	`J`	iBeacon 16-bit major value: • Factory default is 0x0001
`uint8`	`Minor`	`N`	iBeacon 16-bit minor value: • Factory default is 0x0001
`uint8a`	`Uuid`	`U`	iBeacon UUID (must contain 16 bytes of data): • Factory default is E2C56DB5-DFFB-48D2-B060-D0F5A71096E0 (AirLocate) **Note**: `uint8a` data type requires one prefixed “length” byte before binary parameter payload

Related Commands:

- p_ibeacon_set_parameters (.IBSP, ID=12/1)

7.2.12 Eddystone Group (ID=13)

Eddystone methods relate to Eddystone beacon set up and operation.

Commands within this group are listed below:

- p_eddystone_set_parameters (.EDDYSP, ID=13/1)
- p_eddystone_get_parameters (.EDDYGP, ID=13/2)

Events within this group are documented in Eddystone Group (ID=13).

7.2.12.1 p_eddystone_set_parameters (.EDDYSP, ID=13/1)

Configures a new Eddystone beacon behavior.

For details on Eddystone frame types and data, see the example usage and the official documentation from Google.

Note: Eddystone telemetry (TLM) frames typically contain data that updates frequently. EZ-Serial does not automatically change any data contained in Eddystone beacon packets. If you wish to broadcast telemetry data, you must regularly update its content from an external host device with this API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`05-18`	`0D`	`01`	Variable-length command payload, minimum of 5 (0x05), maximum of 24 (0x18).
RSP	`C0`	`02`	`0D`	`01`	None.

Text Info:

Text Name	Response Length	Category	Notes
`.EDDYSP`	`0x000D`	SET	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`enable`	`E`	Enable Eddystone beacon broadcast: • 0 = Disable (factory default) • 1 = Enable • 2 = Enable + auto-start
`uint16`	`interval`	`I`	Advertisement interval for Eddystone broadcasting (625 µs units): • Minimum = 0x00A0 (160 * 0.625 ms = 100 ms, factory default) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
`uint8`	`type`	`T`	Eddystone frame type: • 0x00 = UID • 0x10 = URL (factory default) • 0x20 = Telemetry
`uint8a`	`data`	`D`	Eddystone frame data (0-19 bytes) • Factory default value results in www.cypress.com. **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- p_eddystone_get_parameters (.EDDYGP, ID=13/2)

Related Events:

- gap_adv_state_changed (ASC, ID=4/2) – May occur if Eddystone beaconing is set to start automatically

Example Usage:

- See Configuring Eddystone Transmissions

7.2.12.2 p_eddystone_get_parameters (.EDDYGP, ID=13/2)

Obtains the current Eddystone beacon behavior.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0D`	`02`	None.
RSP	`C0`	`07-1A`	`0D`	`02`	Variable-length response payload, minimum of 7 (0x07), maximum of 26 (0x1A)

Text Info:

Text Name	Response Length	Category	Notes
`.EDDYGP`	`0x0021-0x0047`	GET	Variable-length response payload, minimum of 33 (0x21), maximum of 71 (0x47)

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`enable`	`E`	Enable Eddystone beacon broadcast: • 0 = Disable (factory default) • 1 = Enable • 2 = Enable + auto-start
`uint16`	`interval`	`I`	Advertisement interval for Eddystone broadcasting (625 µs units): • Minimum = 0x00A0 (160 * 0.625 ms = 100 ms, factory default) • Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
`uint8`	`type`	`T`	Eddystone frame type: • 0x00 = UID • 0x10 = URL (factory default) • 0x20 = Telemetry
`uint8a`	`data`	`D`	Eddystone frame data (0-19 bytes) • Factory default value results in www.cypress.com. **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Related Commands:

- p_eddystone_set_parameters (.EDDYSP, ID=13/1)

7.2.13 BT Group (ID=14)

BT methods relate to the BR/EDR functions and operations of the Bluetooth stack, which includes managing basic BT configurations, inquiring device, establishing connection, and maintaining connection.

Note: When BR/EDR functions are required, including the BT SPP function, the Bluetooth address must be set to public address. The random address should not be used, otherwise it may cause connection authentication issue. The random address is only supported by BLE functions.

Commands within this group are listed below:

- bt_start_inquiry (/BTI, ID=14/1)
- bt_cancel_inquiry (/BTIX, ID=14/2)
- bt_query_name (/BTQN, ID=14/3)
- bt_connect (/BTC, ID=14/4)
- bt_disconnect (/BTDIS, ID=14/6)
- bt_query_connections (/BTQC, ID=14/7)
- bt_query_peer_address (/BTQPA, ID=14/8)
- bt_query_rssi (/BTQSS, ID=14/9)
- bt_set_parameters (SBTP, ID=14/10)
- bt_get_parameters (GBTP, ID=14/11)
- bt_set_device_class (SBTDC, ID=14/12)
- bt_get_device_class (GBTDC, ID=14/13)

Events within this group are documented in BT Group (ID=14).

7.2.13.1 *bt_start_inquiry (/BTI, ID=14/1)*

Begins a Bluetooth BR/EDR inquiry.

This command begins inquiring using the specified parameters. EZ-Serial must not already be inquiring for this command to succeed.

EZ-Serial will generate the bt_inquiry_result (BTIR, ID=14/1) API event when the inquiring response packet data is received.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	08	0E	01	Variable-length command payload, minimum of 5 (0x05), maximum of 24 (0x18).
RSP	C0	02	0E	01	None.

Text Info:

Text Name	Response Length	Category	Notes
/BTI	0x000A	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	duration	T*	Inquiry duration time (1.28 seconds increments): • Minimum = 0x03 (3 * 1.28s = 3.84 seconds) • Maximum = 0x1E (30 * 1.28s = 38.4 seconds)

Data Type	Name	Text	Description
`uint8`	`filter_type`	`F`	Inquiry filter condition type, can be one of following listed value: • 0x00 = No inquiry filter (Factory default value.) • 0x01 = Filter on device class • 0x02 = Filter on device address Note that if the filter_type parameter is ignored in the text command, the previously set value or the factory default value will be used. If this value is changed, the filter_data value should also be set at same time.
`uint8a`	`filter_data`	`D`	Inquiry filter condition data, which is fixed to 6 bytes length. The meaning of the value depending on the inquiry filter condition type value: • Factory default value: 0x00 0x00 0x00 0x00 0x00 0x00 • When the filter_type value is 0x01 – Filter on device class. The filter_data format should be: Format of the 6 bytes filter_data value byte0 byte1 byte2 (device_class) byte3 byte4 byte5 (device_class_mask) device_class: Class of the device for inquiring. device_class_mask: Class of the device filter mask. • When the filter_type value is 0x02 – Filter on device address. The filter_data format should be: Format of the 6 bytes filter_data value byte0 byte1 byte2 byte3 byte4 byte5 (bluetooth_address) Bluetooth_address: Bluetooth address for inquiring. Note that if the filter_data parameter is ignored in the text command, the previously set value or the factory default value will be used. This value must follow the changes of filter_type value.

Response Parameters:

None.

Related Commands:

- bt_cancel_inquiry (/BTIX, ID=14/2)

Related Events:

- bt_inquiry_result (BTIR, ID=14/1) – May occur when BT device is discovered nearby

7.2.13.2 bt_cancel_inquiry (/BTIX, ID=14/2)

Cancels the Bluetooth BR/EDR inquiry.

If the Bluetooth BR/EDR inquiry is not running or has been stopped, this command will return error code.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0E`	`02`	None.
RSP	`C0`	`02`	`0E`	`02`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/BTIX`	`0x000B`	ACTION	None.

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- bt_start_inquiry (/BTI, ID=14/1)

Related Events:

None.

7.2.13.3 bt_query_name (/BTQN, ID=14/3)

Gets a Bluetooth-friendly name from a remote BR/EDR device.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`06`	`0E`	`03`	None.
RSP	`C0`	`02`	`0E`	`03`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/BTQN`	`0x000B`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`macaddr`	`address`	`A*`	Bluetooth address of remote device.

Response Parameters:

None.

Related Commands:

- bt_start_inquiry (/BTI, ID=14/1)
- bt_cancel_inquiry (/BTIX, ID=14/2)

Related Events:

- bt_name_result (BTINR, ID=14/2)

7.2.13.4 bt_connect (/BTC, ID=14/4)

Initiates a Bluetooth BR/EDR a connection to a remote device which has SPP service support.

Note: It is known that the connection operation may be disconnected automatically after the first command is run. A reconnection is required by the BT stack after paring information is exchanged. In this situation, retry the same command to successfully establish a connection. During second time, the previously paired information will be used, so the second connection request will be established and no reconnection is required. This will happen especially when the remote device uses the random Bluetooth address. These two steps are similar to the two steps performed on Windows platform. First, connect and add the BT device to the system, and then, use a COM terminal to establish a connection to device through the SPP profile.

The random address is only supported by BLE subsystem, so the ER/EDR functions always use the public address. You can convert the remote random Bluetooth address to public address when using it to connect to the remote ER/EDR device. For example, when the remote device has a random address A=D988EBB91176,T=1, the converted the public address should be A=1988EBB91176. You can then use the command "/BTC,A=1988EBB91176,P=0" instead of "/BTC,A=D988EBB91176,P=0" to establish the BT SPP connection with the remote device.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`07`	`0E`	`04`	None.
RSP	`C0`	`02`	`0E`	`04`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/BTC`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`macaddr`	`address`	`A`*	Bluetooth address of the remote device.
`uint8`	`profile`	`P`	Bluetoot profile type to connect to. • 0x00 – SPP Profile (factory default) • 0x01 – A2DP Profile (not supported currently) • 0x02 – AVRCP Profile (not supported currently) • 0x03 – HFP Profile (not supported currently) • 0x04 – HID Profile (not supported currently) Other values are reserved and are currently invalid.

Response Parameters:

None.

Related Commands:

None.

Related Events:

- bt_connected (BTCON, ID=14/4)
- bt_connection_status (BTCS, ID=14/5)
- bt_connection_failed (BTCF, ID=14/6)

7.2.13.5 bt_disconnect (/BTDIS, ID=14/6)

Closes an established Bluetooth BR/EDR connection from the remote peer device.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`0E`	`06`	None.
RSP	`C0`	`02`	`0E`	`06`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/BTDIS`	`0x000C`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Handle of connection which needs to be disconnected.
`uint8`	`profile`	`P`	BT profile type. • 0x00 – SPP Profile (factory default) • 0x01 – A2DP Profile (not supported currently) • 0x02 – AVRCP Profile (not supported currently) • 0x03 – HFP Profile (not supported currently) • 0x04 – HID Profile (not supported currently)

Data Type	Name	Text	Description
			Other values are reserved and are currently invalid.

Response Parameters:

None.

Related Commands:

- bt_connect (/BTC, ID=14/4)

Related Events:

- bt_disconnected (BTDIS, ID=14/7)
- bt_connection_status (BTCS, ID=14/5)

7.2.13.6 bt_query_connections (/BTQC, ID=14/7)

Queries all established BT profile connections.

The command response returns the number of the BT profile connections. Then, each BT profile connection information will be reported through bt_disconnect (/BTDIS, ID=14/6) event. This command is aimed to supply the connection information, especially the conn_hanndle (C) parameter, so that the host can do further operations based on the connections.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0E`	`07`	None.
RSP	`C0`	`03`	`0E`	`07`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/BTQC`	`0x0010`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`count`	`C`	Number of established BT profile connections**.**

Related Commands:

- bt_connect (/BTC, ID=14/4)
- bt_disconnect (/BTDIS, ID=14/6)

Related Events:

- bt_connection_status (BTCS, ID=14/5)

7.2.13.7 bt_query_peer_address (/BTQPA, ID=14/8)

Gets the Bluetooth address of the remote peer device.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`0E`	`08`	None.
RSP	`C0`	`09`	`0E`	`08`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/BTQPA`	`0x001E`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Handle of the connection for which peer address is queried.

Response Parameters:

Data Type	Name	Text	Description
`macaddr`	`addr`	`A`	Bluetooth address of peer device.

Related Commands:

None.

Related Events:

None.

7.2.13.8 bt_query_rssi (/BTQSS, ID=14/9)

This command returns the RSSI value detected in the packet received most recently from the currently connected remote peer device. An active connection is required to use this command successfully.

Note: RSSI values in real-world environments often fall in the -50 dBm to -70 dBm range. An RSSI value at this level does not necessarily indicate a poor connection.

The RSSI value returned in the response is expressed as a signed 8-bit integer. In text mode, it will appear in two's complement form. Positive numbers in this form fall in the range [0, 127] and are as they appear. Negative numbers fall in the range [128, 255] and should have 256 subtracted from them to obtain the real value.

Examples:

- 0x03 = +3 dBm
- 0x00 = 0 dBm
- 0xFF = -1 dBm (0xFF = 255 - 256 = -1)
- 0xF0 = -16 dBm (0xF0 = 240 - 256 = -16)
- 0xC5 = -59 dBm (0xC5 = 197 - 256 = -59)

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`01`	`0E`	`09`	None.
RSP	`C0`	`02`	`0E`	`09`	None.

Text Info:

Text Name	Response Length	Category	Notes
`/BTQSS`	`0x000C`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Handle of the connection for which signal strength is queried

Response Parameters:

None.

Related Events:

None.

7.2.13.9 bt_set_parameters (SBTP, ID=14/10)

Configures Bluetooth BR/EDR behavior settings.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`04`	`0E`	`0A`	None.
RSP	`C0`	`02`	`0E`	`0A`	None.

Text Info:

Text Name	Response Length	Category	Notes
`SBTP`	`0x000A`	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
`uint8`	`discoverable`	`D`	Bit 0-3 (0x0F) = BR/EDR is discoverable or not by default. • 0 = Non-discoverable • 1 = Limited BR/EDR discoverable • 2 = General BR/EDR discoverable (Factory default value) Bit 4-7 (0xF0) = BR/EDR is discoverable or not when BR/EDR connection has been established. • 0 = Non-discoverable (Factory default value) • 1 = Limited BR/EDR discoverable • 2 = General BR/EDR discoverable **Note**: Only the BT SPP profile and one BT SPP connection is supported currently. So, the value of this Bit 4-7 should be always set to 0.
`uint8`	`connectable`	`C`	Bit 0-3 (0x0F) = BR/EDR is connectable or not by default. • 0 = Not connectable • 1 = BR/EDR connectable (Factory default value) Bit 4-7 (0xF0) = BR/EDR is connectable or not when BR/EDR connection has been established. • 0 = Not connectable (Factory default value) • 1 = BR/EDR connectable **Note**: Only the BT SPP profile and one BT SPP connection is supported currently. Even when Bits 4-7 is set to 1, the second BT SPP connection will still fail. So, the value of this Bits 4-7 should be always set to 0.
`uint8`	`flags`	`F`	Bit flags for BT subsystem. • Bit 0 (0x01) = Enable Bits 4-7 fields of the discoverable and connectable parameters. • 0 = Bits 4-7 fields of the discoverable and connectable parameters are invalid • 1 = Bits 4-7 fields of the discoverable and connectable parameters are valid (Factory default value)
`uint8`	`sniff`	`S`	Allow BT connections to enter sniff mode to save power after being idle for some time. • 0x00 = Does not allow BT connection to automatically enter sniff mode • 0x01 = Allow BT connection to automatically enter sniff mode (Factory Default) Note that this parameter is not supported currently, and is reserved for future use. Do not change the default value.

Response Parameters:

None.

Related Commands:

- bt_get_parameters (GBTP, ID=14/11)

Related Events:

None.

7.2.13.10 bt_get_parameters (GBTP, ID=14/11)

Gets the current Bluetooth BR/EDR behavior settings.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	`C0`	`00`	`0E`	`0B`	None.
RSP	`C0`	`06`	`0E`	`0B`	None.

Text Info:

Text Name	Response Length	Category	Notes
`GBTP`	`0x0001A`	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
`uint8`	`discoverable`	`D`	Bit 0-3 (0x0F) = BR/EDR is discoverable or not by default. • 0 = Non-discoverable • 1 = Limited BR/EDR discoverable • 2 = General BR/EDR discoverable (Factory default value) Bit 4-7 (0xF0) = BR/EDR is discoverable or not when BLE connection has been established. • 0 = Non-discoverable (Factory default value) • 1 = Limited BR/EDR discoverable • 2 = General BR/EDR discoverable
`uint8`	`connectable`	`C`	Bit 0-3 (0x0F) = BR/EDR is connectable or not by default. • 0 = Not connectable • 1 = BR/EDR connectable (Factory default value) Bit 0-3 (0xF0) = BR/EDR is connectable or not when BLE connection has been established. • 0 = Not connectable (Factory default value) • 1 = BR/EDR connectable
`uint8`	`flags`	`F`	Bit flags for BT subsystem. • Bit 0 (0x01) = Enable Bits 4-7 fields of the discoverable and connectable parameters. • 0 = Bits 4-7 fields of the discoverable and connectable parameters are invalid • 1 = Bits 4-7 fields of the discoverable and connectable parameters are valid (Factory default value)
`uint8`	`sniff`	`S`	Allow BT connections to enter sniff mode to save power after being idle for some time. • 0x00 = Does not allow BT connection to automatically enter sniff mode • 0x01 = Allow BT connection to automatically enter sniff mode (Factory Default) Note that this parameter is not supported currently, and is reserved for future use.

Related Commands:

- bt_set_parameters (SBTP, ID=14/10)

Related Events:

None.

7.2.13.11 bt_set_device_class (SBTDC, ID=14/12)

Configures the device class for BT device.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	04	0E	0C	None.
RSP	C0	02	0E	0C	None.

Text Info:

Text Name	Response Length	Category	Notes
SBTDC	0x000B	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint32	dev_class	C	Class of the device. For example, if the device class is { 0x24, 0x04, 0x18 }, set the device class as:. SBTDC,C=240418

Response Parameters:

None.

Related Commands:

- bt_get_device_class (GBTDC, ID=14/13)

Related Events:

None.

7.2.13.12 bt_get_device_class (GBTDC, ID=14/13)

Gets the device class of the BT device.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	0E	0D	None.
RSP	C0	06	0E	0D	None.

Text Info:

Text Name	Response Length	Category	Notes
GBTDC	0x00016	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint32	dev_class	C	Class of the device. For example, if the device class is { 0xF1, 0xE2, 0xC3 }, the return device class value will be: @R,0016,GBTDC,0000,C=00F1E2C3 Ignore the first byte 00.

Related Commands:

- bt_set_device_class (SBTDC, ID=14/12)

Related Events:

None.

7.3 API Events

All events implemented in the API protocol are described in detail below. API commands and responses are documented separately in API Commands and Responses.

For the master list of all possible error codes appearing in certain events, see Error Codes.

Commands and responses are broken down into the following groups:

- Protocol Group (ID=1)
- System Group (ID=2)
- OTA Group (ID=3)
- GAP Group (ID=4)
- GATT Server Group (ID=5)
- GATT Client Group (ID=6)
- SMP Group (ID=7)
- L2CAP Group (ID=8)
- GPIO Group (ID=9)
- CYSPP Group (ID=10)
- CYCommand Group (ID=11)
- iBeacon Group (ID=12)
- Eddystone Group (ID=13)

7.3.1 Protocol Group (ID=1)

Protocol methods allow you to change the way the API protocol operates while communicating with an external host over the serial interface.

The protocol group currently has no events. Commands within this group are documented in Protocol Group (ID=1).

7.3.2 System Group (ID=2)

System methods relate to the core device, and describe boot, device address information, and resetting to an initial state.

Events within this group are listed below:

- system_boot (BOOT, ID=2/1)
- system_error (ERR, ID=2/2)
- system_factory_reset_complete (RFAC, ID=2/3)
- system_factory_test_entered (TFAC, ID=2/4)
- system_dump_blob (DBLOB, ID=2/5)

Commands within this group are documented in System Group (ID=2).

7.3.2.1 system_boot (BOOT, ID=2/1)

EZ-Serial module has booted and is ready to process commands.

Binary Header:

Type	Length	Group	ID	Notes
80	12	02	01	None.

Text Info:

Text Name	Event Length	Notes
BOOT	0x003B	None.

Event Parameters:

Data Type	Name	Text	Description
uint32	app	E	Application version number. The data format is: app_major(byte3).app_minor(byte2).app_revision(byte1).app_build(byte0)
uint32	stack	S	BLE stack version number. The data format is: stack_major (byte3).stack_minor (byte2).0.stack_build (byte1-byte0) **Note:** The stack_revision value is always 0. So, this value is not stored in the stack area. The stack_build takes two bytes.
uint16	protocol	P	API protocol version number The data format is: protocol_major (byte1).protocol_minor (byte0)
uint8	hardware	H	Hardware identifier: • 0xE1 = CYBT-423028-02 • 0xE2 = CYBT-423054-02 • 0xE3 = CYBT-423060-02 • 0xE4 = CYBT-413034-02 • 0xE5 = CYBT-413055-02 • 0xE6 = CYBT-413061-02 • 0xE7 = CYBT-483039-02 • 0xE8 = CYBT-483056-02 • 0xE9 = CYBT-483062-02
uint8	cause	C	Cause of boot event: • 0x00 = Cold boot/Reset • 0x01 = Warn/Fast boot
macaddr	address	A	Bluetooth address
uint8	type	T	Bluetooth address type

Related Commands:

- system_reboot (/RBT, ID=2/2)
- system_factory_reset (/RFAC, ID=2/5)

7.3.2.2 *system_error (ERR, ID=2/2)*

System error has occurred.

This may be triggered by a malformed command, an operation that failed or could start due to an invalid operational state, or a low-level hardware failure. See Error Codes for a list of all possible errors.

Binary Header:

Type	Length	Group	ID	Notes
80	02	02	02	None.

Text Info:

Text Name	Event Length	Notes
ERR	0x000B	None.

Event Parameters:

Data Type	Name	Text	Description
`uint16`	`error`	`E`	Error code describing what went wrong

7.3.2.3 system_factory_reset_complete (RFAC, ID=2/3)

Factory reset is complete.

This event will occur after sending the system_factory_reset (/RFAC, ID=2/5) API command, or asserting (LOW) the **FACTORY_TR** and **CYSPP** pins at boot time. EZ-Serial transmits this event using the originally configured host interface settings (if different from the default). After generating this event, the module will reboot immediately and the default settings will take effect.

Note: If you triggered a factory reset using the GPIO method at boot time, the final reboot back into an operational state will only occur after you de-assert one or both pins. This safeguard prevents an endless loop of factory resets if both pins remain asserted.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`00`	`02`	`03`	None.

Text Info:

Text Name	Event Length	Notes
`RFAC`	`0x0005`	None.

Event Parameters:

None.

Related Commands:

- system_factory_reset (/RFAC, ID=2/5)

7.3.2.4 system_factory_test_entered (TFAC, ID=2/4)

Manufacturing test mode is active.

This event occurs if you assert (LOW) the FACTORY_TR pin at boot time. The module will remain in this state until you reset or power-cycle it. Test mode is currently only intended for internal use during the manufacturing.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`00`	`02`	`04`	None.

Text Info:

Text Name	Event Length	Notes
`TFAC`	`0x0005`	None.

Event Parameters:

None.

7.3.2.5 *system_dump_blob (DBLOB, ID=2/5)*

Single data blob of requested configuration type or system state.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`04-14`	`02`	`05`	Variable-length event payload, minimum of 4 (0x04), maximum of 20 (0x14).

Text Info:

Text Name	Event Length	Notes
`DBLOB`	`0x0015-0x0035`	Variable-length event payload, minimum of 21 (0x15), maximum of 53 (0x35)

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`type`	`T`	Type of information being dumped: • 0 = Runtime configuration data • 1 = Boot-level configuration data • 2 = Factory-level configuration data • 3 = System state data
`uint16`	`offset`	`O`	Blob start offset
`uint8a`	`data`	`D`	Dumped blob of data **Note:** `uint8a` data type requires one prefixed “length” byte before binary parameter payload

Related Commands:

- system_dump (/DUMP, ID=2/3)

7.3.3 OTA Group (ID=3)

OTA methods relate to the firmware update process, using over-the-air GATT-based firmware transfer.

There are no events within this group.

Commands within this group are documented OTA Group (ID=3).

7.3.4 GAP Group (ID=4)

GAP methods relate to the Generic Access Protocol layer of the Bluetooth stack, which includes managing scanning, advertising, establishing connection, and maintaining connection.

Events within this group are listed below:

- gap_whitelist_entry (WL, ID=4/1)
- gap_adv_state_changed (ASC, ID=4/2)
- mgap_scan_state_changed (SSC, ID=4/3)
- gap_scan_result (S, ID=4/4)
- gap_connected (C, ID=4/5)
- gap_disconnected (DIS, ID=4/6)
- gap_connection_update_requested (UCR, ID=4/7)
- gap_connection_updated (CU, ID=4/8)
- gap_mtu_updated (MTU, ID=4/10)

Commands within this group are documented in GAP Group (ID=4).

7.3.4.1 gap_whitelist_entry (WL, ID=4/1)

Details about a single entry in the whitelist table.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`07`	`04`	`01`	None.

Text Info:

Text Name	Event Length	Notes
`WL`	`0x0017`	None.

Event Parameters:

Data Type	Name	Text	Description
`macaddr`	`address`	`A`	Bluetooth address
`uint8`	`type`	`T`	Address type: • 0 = Public • 1 = Random/private

Related Commands:

- gap_add_whitelist_entry (/WLA, ID=4/6)
- gap_query_whitelist (/QWL, ID=4/14)

7.3.4.2 gap_adv_state_changed (ASC, ID=4/2)

Indicates that the module has started or stopped advertising due to a scheduled timeout, automated process, or intentional action.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`02`	`04`	`02`	None.

Text Info:

Text Name	Event Length	Notes
`ASC`	`0x000E`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`state`	`S`	Advertising state: • 0 = Stopped • 1 = Active Directed HIGH • 2 = Active Directed LOW • 3 = Active Undirected HIGH • 4 = Active Undirected LOW • 5 = Active Non-connectable HIGH • 6 = Active Non-connectable LOW • 7 = Active Discoverable HIGH • 8 = Active Discoverable LOW

Data Type	Name	Text	Description
`uint8`	`reason`	`R`	Reason for state change: • 0 = User command • 1 = GAP automatic advertisement enabled • 2 = Configured timeout expired • 3 = CYSPP operation state change • 4 = iBeacon operation state change • 5 = Eddystone operation state change • 6 = Disconnection

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_stop_adv (/AX, ID=4/9)
- gap_set_adv_parameters (SAP, ID=4/23)
- p_cyspp_start (.CYSPPSTART, ID=10/2)
- p_cyspp_set_parameters (.CYSPPSP, ID=10/3)

7.3.4.3 mgap_scan_state_changed (SSC, ID=4/3)

Indicates that the module has started or stopped scanning, due to a scheduled timeout or an intentional action.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`02`	`04`	`03`	None.

Text Info:

Text Name	Event Length	Notes
`SSC`	`0x000E`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`state`	`S`	Scanning state: • 0 = Stopped • 1 = High Duty Scan • 2 = Low Duty Scan
`uint8`	`reason`	`R`	Reason for state change: • 0 = User command • 1 = NOT USED • 2 = Configured timeout expired • 3 = CYSPP operation state change

Related Commands:

- gap_start_scan (/S, ID=4/10)
- gap_stop_scan (/SX, ID=4/11)
- p_cyspp_start (.CYSPPSTART, ID=10/2)
- p_cyspp_get_parameters (.CYSPPGP, ID=10/4)

7.3.4.4 gap_scan_result (S, ID=4/4)

Details about an advertisement or scan response packet.

This event occurs while scanning for remote devices. If you have enabled active scanning, most peripherals will provide two separate packets delivered via this API: one advertisement packet and one scan response packet. Passive scanning will result in only the first of those two. Scan response packets typically contain less critical data, such as the friendly name of the device or its transmit power.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`0B-2A`	`04`	`04`	Variable-length event payload, minimum of 11 (0x0B), maximum of 42 (0x2A)

Text Info:

Text Name	Event Length	Notes
`S`	`0x0028-0x0047`	Variable-length event payload, minimum of 40 (0x28), maximum of 71 (0x47)

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`result_type`	`R`	Scan result type: • 0 = Connectable undirected advertisement packet • 1 = Connectable directed advertisement packet • 2 = Scannable undirected advertisement packet • 3 = Non-connectable undirected advertisement packet • 4 = Scan response packet
`macaddr`	`address`	`A`	Bluetooth address
`uint8`	`address_type`	`T`	Address type: • 0 = Public • 1 = Random/private
`int8`	`rssi`	`S`	RSSI (If the value ≥0x7F, it is invalid and should be ignored.)
`uint8`	`bond`	`B`	Bond entry (0 for no bond)
`uint8a`	`data`	`D`	Advertisement payload data (0-31 bytes) **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Related Commands:

- gap_connect (/C, ID=4/1)
- gap_start_scan (/S, ID=4/10)
- gap_stop_scan (/SX, ID=4/11)
- gap_set_scan_parameters (SSP, ID=4/25)

Example Usage:

- See Scanning for Peripheral Devices

7.3.4.5 *gap_connected (C, ID=4/5)*

Connection established with a remote device.

Binary Header:

Type	Length	Group	ID	Notes
80	0F	04	05	None.

Text Info:

Text Name	Event Length	Notes
C	0x0035	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle. It is required in other link-related operation commands.
macaddr	address	A	Bluetooth address
uint8	type	T	Address type: • 0 = Public • 1 = Random/private
uint16	interval	I	Connection interval
uint16	slave_latency	L	Slave latency
uint16	supervision_timeout	O	Supervision timeout
uint8	bond	B	Bond entry (0 for no bond)

Related Commands:

- gap_connect (/C, ID=4/1)
- gap_update_conn_parameters (/UCP, ID=4/3)
- gap_disconnect (/DIS, ID=4/5)

Related Events:

- gap_disconnected (DIS, ID=4/6)
- gap_connection_update_requested (UCR, ID=4/7)
- gap_connection_updated (CU, ID=4/8)

Example Usage:

- See Connecting to a Peripheral Device

7.3.4.6 *gap_disconnected (DIS, ID=4/6)*

Connection to a remote device has been closed.

For a list of possible disconnection reasons, see the 0x900 range of codes (EZ-Serial System Error Codes). The most common reasons are:

- 0x0908 – Page timeout (unexpected loss of connectivity, no response within supervision timeout)
- 0x0913 – Remote user terminated connection (cleanly closed from remote side)
- 0x0916 – Connection terminated by local host (cleanly closed from local side)
- 0x093E – Connection failed to be established (connection initiated locally, but peer did not respond to request)

Binary Header:

Type	Length	Group	ID	Notes
80	03	04	06	None.

Text Info:

Text Name	Event Length	Notes
DIS	0x0010	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	reason	R	Reason for disconnection

Related Commands:

- gap_connect (/C, ID=4/1)
- gap_disconnect (/DIS, ID=4/5)

Example Usage:

- See Disconnecting from a Peripheral Device

7.3.4.7 gap_connection_update_requested (UCR, ID=4/7)

Remote peer has requested a connection parameter update.

To accept or reject the new request, use the gap_update_conn_parameters (/UCP, ID=4/3) API command. An argument of "0" for that command will accept, and non-zero will reject.

NOTE: This event and the gap_update_conn_parameters (/UCP, ID=4/3) API command for replying only apply when operating as the BLE master device. In the slave role, the specification requires that the slave accept whatever connection parameters the master supplies. When connected as a slave, a connection update request from a master will result only in the gap_connection_updated (CU, ID=4/8) API event.

Binary Header:

Type	Length	Group	ID	Notes
80	09	04	07	None.

Text Info:

Text Name	Event Length	Notes
UCR	0x0025	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Handle of the connection requesting new parameters
uint16	interval_min	I	Minimum connection interval
uint16	interval_max	X	Maximum connection interval
uint16	slave_latency	L	Slave latency
uint16	supervision_timeout	O	Supervision timeout

Related Commands:

- gap_update_conn_parameters (/UCP, ID=4/3)

Related Events:

- gap_connection_updated (CU, ID=4/8)

7.3.4.8 *gap_connection_updated (CU, ID=4/8)*

Active connection has negotiated and applied new parameters.

This event occurs on the slave side after a master requests new parameters or accepts the new parameters requested by the slave. It also occurs on the master side after a slave requests new parameters and the master accepts the request.

Note: A connection update request sent from a slave, but rejected, will not result in any events indicating the rejection. The slave must assume the original parameters are in effect until after it receives this API event.

Binary Header:

Type	Length	Group	ID	Notes
80	07	04	08	None.

Text Info:

Text Name	Event Length	Notes
CU	0x001D	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	interval	I	Connection interval
uint16	slave_latency	L	Slave latency
uint16	supervision_timeout	O	Supervision timeout

Related Commands:

- gap_update_conn_parameters (/UCP, ID=4/3)

Related Events:

- gap_connection_update_requested (UCR, ID=4/7)

7.3.4.9 *gap_rssi_result (RSSI, ID=4/9)*

This event returns the RSSI value detected in the packet received most recently from the currently connected remote peer device. An active connection is required to use the read RSSI command successfully.

Note: RSSI values in real-world environments often fall in the -50 dBm to -70 dBm range. An RSSI value at this level does not necessarily indicate a poor connection.

The RSSI value returned in the response is expressed as a signed 8-bit integer. In text mode, it will appear in two's complement form. Positive numbers in this form fall in the range [0, 127] and are as they appear. Negative numbers fall in the range [128, 255] and should have 256 subtracted from them to obtain the real value.

Examples:

- 0x03 = +3 dBm
- 0xFF = -1 dBm (0xFF = 255 - 256 = -1)
- 0xF0 = -16 dBm (0xF0 = 240 - 256 = -16)
- 0xC5 = -59 dBm (0xC5 = 197 - 256 = -59)

Binary Header:

Type	Length	Group	ID	Notes
80	01	04	09	None.

Text Info:

Text Name	Event Length	Notes
RSSI	0x0001	None.

Event Parameters:

Data Type	Name	Text	Description
int8	rssi	R	RSSI value in dBm (between -95 and +10), or 0 if used not connected.

Related Commands:

- gap_query_rssi (/QSS, ID=4/13)

7.3.4.10 gap_mtu_updated (MTU, ID=4/10)

The event provides the negotiated MTU size, which will be used within the specific connection.

Binary Header:

Type	Length	Group	ID	Notes
80	01	04	0A	None.

Text Info:

Text Name	Event Length	Notes
MTU	0x0003	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle.
uint16	Mtu	M	The negotiated MTU size which will be used between the connections.

Related Commands:

- gap_configure_mtu (/GCMTU, ID=4/29) – Command to trigger MTU size negotiation between two connected devices.

7.3.5 GATT Server Group (ID=5)

GATT server methods relate to the server role of the Generic Attribute Protocol layer of the Bluetooth stack. These methods are used for working with the local GATT structure.

Events within this group are listed below:

- gatts_discover_result (DL, ID=5/1)
- gatts_data_written (W, ID=5/2)
- gatts_indication_confirmed (IC, ID=5/3)
- gatts_db_entry_blob (DGATT, ID=5/4)

Commands within this group are documented in GATT Client Group (ID=6).

7.3.5.1 gatts_discover_result (DL, ID=5/1)

Details about a single entry in the local GATT database.

This event occurs while discovering local services, characteristics, or descriptors.

Binary Header:

Type	Length	Group	ID	Notes
80	08+	05	01	Variable-length event payload, value specified is minimum.

Text Info:

Text Name	Event Length	Notes
`DL`	`0x0020+`	Variable-length event payload, value specified is minimum.

Event Parameters:

Data Type	Name	Text	Description
`uint16`	`attr_handle`	`H`	Attribute handle
`uint16`	`attr_handle_rel`	`R`	Related attribute handle: • If discovering services, the end handle for the service group • If discovering characteristics, the value handle that holds the application data • If discovering descriptors, always 0 (not applicable)
`uint16`	`type`	`T`	Attribute type: • 0x2800 = Primary Service Declaration • 0x2801 = Secondary Service Declaration • 0x2802 = Include Declaration • 0x2803 = Characteristic Declaration • 0x2900 = Characteristic Extended Properties descriptor • 0x2901 = Characteristic User Description descriptor • 0x2902 = Client Characteristic Configuration descriptor • 0x2903 = Server Characteristic Configuration descriptor • 0x2904 = Characteristic Format descriptor • 0x2905 = Characteristic Aggregate Format descriptor • 0x0000 = Characteristic value attribute or user-defined structure (see UUID)
`uint8`	`properties`	`P`	Characteristic properties bitmask, only non-zero during **characteristic** discovery: • Bit 0 (0x01) = Broadcast • Bit 1 (0x02) = Read • Bit 2 (0x04) = Write without response • Bit 3 (0x08) = Write • Bit 4 (0x10) = Notify • Bit 5 (0x20) = Indicate • Bit 6 (0x40) = Signed write • Bit 7 (0x80) = Extended properties (will have 0x2900 descriptor)
`uint8a`	`uuid`	`U`	UUID **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload. The UUID is shown in little-endian format.

Related Commands:

- gatts_discover_services (/DLS, ID=5/6)
- gatts_discover_characteristics (/DLC, ID=5/7)
- gatts_discover_descriptors (/DLD, ID=5/8)

7.3.5.2 gatts_data_written (W, ID=5/2)

Remote GATT client has written data to a local attribute.

A connected remote client can write data to a local attribute using either acknowledged or unacknowledged write operations Acknowledged writes require two full connection intervals to complete: one for the data transfer from client to server, and another for the acknowledgement back from server to client. Unacknowledged writes may occur multiple times within the same connection interval, and therefore provide greater throughput potential.

EZ-Serial automatically responds to acknowledged writes except in two cases:

- You have disabled automatic responses using the gatts_set_parameters (SGSP, ID=5/14) API command
- The attribute written to has the "User data management" bit set in its properties value, set during creation with the gatts_create_attr (/CAC, ID=5/1) API command.

In these cases, the **type** parameter of this event will have the high bit (0x80) set, indicating that you must manually respond to the write using the gatts_send_writereq_response (/WRR, ID=5/13) API command. This acknowledgement is required before any other GATT operations can occur on either the local or remote side. Failing to respond within 30 seconds will result in client disconnection.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`06`	`05`	`02`	Variable-length event payload, value specified is minimum.

Text Info:

Text Name	Event Length	Notes
`W`	`0x0016+`	Variable-length event payload, value specified is minimum.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Handle of the connection from which write came
`uint16`	`attr_handle`	`H`	Attribute handle
`uint8`	`type`	`T`	Write type: • 0x00 = Simple write – acknowledged • 0x01 = Write without response – unacknowledged • 0x80 = Simple write requiring manual response via API command
`longuint8a`	`data`	`D`	Written data **Note:** `longuint8a` data type requires two prefixed “length” bytes before binary parameter payload

Related Commands:

- gatts_send_writereq_response (/WRR, ID=5/13) – Required after acknowledged writes when manual response bit is set
- gattc_write_handle (/WRH, ID=6/5) – Used on the client side to write data to a remote GATT server attribute

7.3.5.3 gatts_indication_confirmed (IC, ID=5/3)

Remote GATT client has confirmed receipt of indicated data.

This event occurs after a client receives and confirms data pushed using the gatts_indicate_handle (/IH, ID=5/12) API command.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`03`	`05`	`03`	None.

Text Info:

Text Name	Event Length	Notes
`IC`	`0x000F`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Handle of the connection from which confirmation came
`uint16`	`attr_handle`	`H`	Attribute handle used for indication

Related Commands:

- gatts_indicate_handle (/IH, ID=5/12)

Related Events:

- gattc_data_received (D, ID=6/3) – Occurs on the remote client after receiving indicated data

7.3.5.4 gatts_db_entry_blob (DGATT, ID=5/4)

Single entry from the GATT structure definition.

This event presents local dynamic GATT attribute definition in a format which simplifies re-entry using the gatts_create_attr (/CAC, ID=5/1) API command. For details about the data provided in this event, see Defining Custom Local GATT Services and Characteristics.

Note: This event includes the attribute handle and the absolute group end value, neither of which are part of the data entered when creating a new custom attribute. Make sure to remove the handle and absolute group end if you are copying the content from these output lines into new commands manually.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`10-20`	`05`	`04`	Variable-length event payload, minimum of 16 (0x10), maximum of 32 (0x20)

Text Info:

Text Name	Event Length	Notes
`DGATT`	`0x0037-0x0057`	Variable-length event payload, minimum of 55 (0x37), maximum of 87 (0x57)

Event Parameters:

Data Type	Name	Text	Description
`uint16`	`handle`	`H`	Attribute handle (0x0001 – 0xFFFF)
`uint16`	`type`	`T`*	Attribute type: • 0x2800 = Primary Service Declaration • 0x2801 = Secondary Service Declaration • 0x2802 = Include Declaration • 0x2803 = Characteristic Declaration • 0x2900 = Characteristic Extended Properties descriptor • 0x2901 = Characteristic User Description descriptor • 0x2902 = Client Characteristic Configuration descriptor • 0x2903 = Server Characteristic Configuration descriptor • 0x2904 = Characteristic Format descriptor • 0x2905 = Characteristic Aggregate Format descriptor • 0x0000 = Characteristic value attribute or user-defined structure with SRAM value storage (auto-managed) • 0x0001 = Characteristic value attribute or user-defined structure with no value storage (user-managed)
`uint8`	`read_permissions`	`R`*	Attribute read permissions: • Bit 0 (0x01) = Read permitted • Bit 1 (0x02) = Encryption required • Bit 2 (0x04) = Authentication required • Bit 3 (0x08) = Authorization required • Bit 4 (0x10) = LE secure connection authentication required • Bits 5-7 (0xE0) = *RESERVED*
`uint8`	`write_permissions`	`W`*	Attribute write permissions: • Bit 0 (0x01) = Write permitted • Bit 1 (0x02) = Encryption required • Bit 2 (0x04) = Authentication required • Bit 3 (0x08) = Authorization required • Bit 4 (0x10) = LE secure connection authentication required • Bit 5-7 (0xE0) = *RESERVED*

Data Type	Name	Text	Description
`uint8`	`char_properties`	`C`*	Characteristic properties (byte 1) • Bit 0 (0x01) = Broadcast • Bit 1 (0x02) = Read • Bit 2 (0x04) = Write without response • Bit 3 (0x08) = Write • Bit 4 (0x10) = Notify • Bit 5 (0x20) = Indicate • Bit 6 (0x40) = Signed write • Bit 7 (0x80) = Extended properties (requires 0x2900 descriptor)
`uint16`	`length`	`L`	Maximum length
`longuint8a`	`data`	`D`	Data (UUID or default attribute value where applicable) **Note:** `longuint8a` data type requires two prefixed "length" bytes before binary parameter payload

Related Commands:

- gatts_dump_db (/DGDB, ID=5/5)

7.3.6 GATT Client Group (ID=6)

GATT client methods relate to the client role of the Generic Attribute Protocol layer of the Bluetooth stack. These methods are used for working with the GATT structures on remote devices, and can only be used while a device is connected.

Events within this group are listed below:

- gattc_discover_result (DR, ID=6/1)
- gattc_remote_procedure_complete (RPC, ID=6/2)
- gattc_data_received (D, ID=6/3)
- gattc_write_response (WRR, ID=6/4)

Commands within this group are documented in GATT Client Group (ID=6).

7.3.6.1 gattc_discover_result (DR, ID=6/1)

Details about a single entry in the remote GATT database.

This event occurs while you are discovering remote services, characteristics, or descriptors.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`09-19`	`06`	`01`	Variable-length event payload, minimum of 9 (0x09), maximum of 25 (0x19)

Text Info:

Text Name	Event Length	Notes
`DR`	`0x0025-0x0044`	Variable-length event payload, minimum of 37 (0x25), maximum of 69 (0x45)

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle
`uint16`	`attr_handle`	`H`	Attribute handle of the discovered Service, Included Service Characteristic, or Descriptor.

Data Type	Name	Text	Description
`uint16`	`attr_handle_rel`	`R`	Related attribute handle: • If discovering services, it is the **end handle** for the service group • If discovering included services, it is the **start handle** of the include service. The value of the properties field is reused as attr_count, which contains the number of attributes that are counted within the included service. • If discovering characteristics, it is the **value handle** of the characteristic value attribute handle • If discovering descriptors, always 0 (not applicable)
`uint8`	`type`	`T`	GATT discover type: • 1 = Discovery all service • 2 = Discovery service by UUID • 3 = Discovery an included service within a service • 4 = Discovery characteristics of a service with/without type requirement • 5 = Discovery characteristics descriptors of a characteristic.
`uint8`	`Properties (attr_count)`	`P`	When the value of **type** field is 3, then the **properties** field is reused as the attributes count (**attr_count**) of the included service. So, the included service's **end handle** should be the value of **attr_handle_rel** field adds the value of **attr_count** field. In other case, it is the characteristic properties bitmask, see following description. Characteristic properties bitmask, only non-zero during characteristic discovery: • Bit 0 (0x01) = Broadcast • Bit 1 (0x02) = Read • Bit 2 (0x04) = Write (No Response) • Bit 3 (0x08) = Write • Bit 4 (0x10) = Notify • Bit 5 (0x20) = Indicate • Bit 6 (0x40) = Authenticate • Bit 7 (0x80) = Extended properties (will have 0x2900 descriptor)
`uint8a`	`uuid`	`U`	UUID (16-bit, 32-bit, or 128-bit) The meaning of the UUID depends on the type value of GATT discovery type. The UUID is: • Service UUID when the value of type is 1 or 2 • Included Service UUID when the value of type is 3 • Empty when the value of type is 4 • Descriptor type UUID when the value of type is 5 **Note:** `uint8a` data type requires one prefixed "length" byte before binary parameter payload

Related Commands:

- gattc_discover_services (/DRS, ID=6/1)
- gattc_discover_characteristics (/DRC, ID=6/2)
- gattc_discover_descriptors (/DRD, ID=6/3)

Related Events:

- gattc_remote_procedure_complete (RPC, ID=6/2)

Example Usage:

- See Discovering a Remote Server's GATT Structure

7.3.6.2 gattc_remote_procedure_complete (RPC, ID=6/2)

Remote GATT client operation has completed.

This event occurs after requesting a GATT client operation that may require an unknown length of time or quantity of returned results before it is finished, such as a remote GATT descriptor discovery. Since you cannot perform multiple GATT client operations simultaneously, your application logic must wait for this event and only continue with additional client operations after the event occurs.

See Related Commands for specific commands which trigger this event.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`04`	`06`	`02`	None.

Text Info:

Text Name	Event Length	Notes
`RPC`	`0x000F`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle
`uint16`	`result`	`R`	GATT result code for procedure: • 0 = Success • 0x01-0x7F = Error from Bluetooth specification • 0x80-0xFF = Error from application (user-defined)
`uint8`	`type`	`T`	GATT discover type: • 1 = Discover all service • 2 = Discover service by UUID • 3 = Discover an included service within a service • 4 = Discover characteristics of a service with/without type requirement • 5 = Discover characteristics descriptors of a characteristic

Related Commands:

- gattc_discover_services (/DRS, ID=6/1) – Always triggers this event upon completion
- gattc_discover_characteristics (/DRC, ID=6/2) – Always triggers this event upon completion
- gattc_discover_descriptors (/DRD, ID=6/3) – Always triggers this event upon completion
- gattc_read_handle (/RRH, ID=6/4) – Triggers this event if read fails, otherwise triggers gattc_data_received (D, ID=6/3)

Related Events:

- gattc_discover_result (DR, ID=6/1) – Occurs during a remote GATT discovery prior to this event

Example Usage:

- See Discovering a Remote Server's GATT Structure

7.3.6.3 gattc_data_received (D, ID=6/3)

Remote GATT server has returned or pushed a value from one of its attributes.

This event occurs after sending a read request with the gattc_read_handle (/RRH, ID=6/4) API command, or when a remote GATT server pushes a data update using a notification or indication after the client subscribes to either of these transfer types on supported characteristics. The **`source`** parameter describes the operation that triggered the event.

If the data received came from a remote GATT server indication and you have disabled automatic confirmations by clearing the auto-confirm bit of the `flags` argument in the gattc_set_parameters (SGCP, ID=6/7) API command, you must manually confirm the indication before performing any other operations. If the `source` parameter of this event has the high bit (0x80) set, use the gattc_confirm_indication (/CI, ID=6/6) API command.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`06-206`	`06`	`03`	Variable-length event payload, minimum of 6 (0x06), maximum of 518 (0x206)

Text Info:

Text Name	Event Length	Notes
`D`	`0x0016-0x0416`	Variable-length event payload, minimum of 22 (0x16), maximum of 1046 (0x416)

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle
`uint16`	`handle`	`H`	Attribute handle
`uint8`	`source`	`S`	Transfer source: • 0x00 = GATT client read request • 0x01 = GATT server notification • 0x02 = GATT server indication • 0x82 = GATT server indication requiring manual confirmation
`longuint8a`	`data`	`D`	Received value (0-512 bytes) **Note:** `longuint8a` data type requires two prefixed “length” bytes before binary parameter payload

Related Commands:

- gatts_notify_handle (/NH, ID=5/11)
- gatts_indicate_handle (/IH, ID=5/12)
- gattc_read_handle (/RRH, ID=6/4)
- gattc_confirm_indication (/CI, ID=6/6)

7.3.6.4 gattc_write_response (WRR, ID=6/4)

Remote GATT server acknowledged GATT client write operation.

This event occurs after attempting an acknowledged write operation with the gattc_write_handle (/WRH, ID=6/5) API command. If the write is accepted by the remote server, the **`result`** value will be 0. Any non-zero **`result`** value indicates an error.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`05`	`06`	`04`	None.

Text Info:

Text Name	Event Length	Notes
`WRR`	`0x0014`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle
`uint16`	`attr_handle`	`H`	Attribute handle
`uint16`	`result`	`R`	GATT result code: • 0 = Success • 0x601-0x067F = Error from Bluetooth specification • 0x680-0x06FF = Error from remote server application (user-defined)

Related Commands:

- gattc_write_handle (/WRH, ID=6/5)
- gatts_send_writereq_response (/WRR, ID=5/13)

7.3.7 SMP Group (ID=7)

SMP methods relate to the Security Manager Protocol layer of the Bluetooth stack. These methods are used for working with encryption, pairing, and bonding between two peers.

Events within this group are listed below:

- smp_bond_entry (B, ID=7/1)
- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)
- smp_encryption_status (ENC, ID=7/4)
- smp_passkey_display_requested (PKD, ID=7/5)
- smp_passkey_entry_requested (PKE, ID=7/6)

Commands within this group are documented in SMP Group (ID=7).

7.3.7.1 smp_bond_entry (B, ID=7/1)

Details about a single entry in the bonding table.

This event occurs once after a new bond is created because of the pairing process, or multiple times (based on bond list count) after requesting the bond list with the smp_query_bonds (/QB, ID=7/1) API command.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`09`	`07`	`01`	None.

Text Info:

Text Name	Event Length	Notes
`B`	`0x001B`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`handle`	`C`	Connection handle of the bonded device.
`macaddr`	`address`	`A`	Bluetooth address
`uint8`	`type`	`T`	Address type: • 0 = Public • 1 = Random/private
`Uint8`	`dev_type`	`D`	Bonded device type: • 0 = Unknow device type • 1 = BR/EDR device • 2 = LE device • 3 = Dual mode device

Related Commands:

- smp_query_bonds (/QB, ID=7/1)
- smp_pair (/P, ID=7/3)

7.3.7.2 *smp_pairing_requested (P, ID=7/2)*

Remote device has requested pairing.

When this event occurs, you must use the smp_send_pairreq_response (/PR, ID=7/5) API command to continue the process, unless the auto-accept bit is set in the `flags` setting of the smp_set_parameters (SSMPP, ID=7/32) API command.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`05`	`07`	`02`	None.

Text Info:

Text Name	Event Length	Notes
`P`	`0x0016`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle
`uint8`	`mode`	`M`	Security level setting reported to peer: • 0x10 = Mode 1, Level 1 – No security • 0x11 = Mode 1, Level 2 – Unauthenticated pairing with encryption (no MITM) • 0x12 = Mode 1, Level 3 – Authenticated pairing with encryption (with MITM) • 0x21 = Mode 2, Level 2 – Unauthenticated pairing with data signing (no MITM) • 0x22 = Mode 2, Level 3 – Authenticated pairing with data signing (with MITM)
`uint8`	`bonding`	`B`	Bond during pairing process: • 0 = Do not bond (exchange keys and encrypt only) • 1 = Bond (permanently store exchanged encryption data)
`uint8`	`keysize`	`K`	Encryption key size (7-16), value is ignored if pairing initiated by slave device
`uint8`	`pairprop`	`P`	Pairing properties: • Bit 0 (0x01): MITM enabled for Secure Connections (SC)

Related Commands:

- smp_pair (/P, ID=7/3)
- smp_send_pairreq_response (/PR, ID=7/5)

Related Events:

- smp_pairing_result (PR, ID=7/3)

7.3.7.3 *smp_pairing_result (PR, ID=7/3)*

Pairing process has ended.

This event indicates that the pairing process is finished, successfully or otherwise. If the `result` parameter is 0, then pairing has completed successfully, and the smp_bond_entry (B, ID=7/1) API event will follow if bonding is enabled. Any non-zero `result` value indicates failure.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`03`	`07`	`03`	None.

Text Info:

Text Name	Event Length	Notes
`PR`	`0x000C`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle
`uint16`	`result`	`R`	Result

Related Commands:

- smp_pair (/P, ID=7/3)

Related Events:

- smp_encryption_status (ENC, ID=7/4)
- smp_bond_entry (B, ID=7/1)

7.3.7.4 smp_encryption_status (ENC, ID=7/4)

Encryption status has changed.

This event confirms that a link has transitioned between plain text and encrypted status during the pairing process.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`02`	`07`	`04`	None.

Text Info:

Text Name	Event Length	Notes
`ENC`	`0x000E`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle
`uint8`	`status`	`S`	Encryption status: • 0 = Not encrypted • 1 = Encrypted

Related Commands:

- smp_pair (/P, ID=7/3)

Related Events:

- smp_pairing_result (PR, ID=7/3)

7.3.7.5 smp_passkey_display_requested (PKD, ID=7/5)

Remote peer requires passkey display for entry or comparison during pairing.

This event provides the local device with the passkey generated as part of the pairing process, so that the local device may display or otherwise make it available to the user for entry or comparison on the remote device. This type of passkey generation and display will be used if the local I/O capabilities are set to “Display Only” or “Display + Yes/No” using the smp_set_br_edr_security_parameters (SSBSP, ID=7/18) or smp_set_le_security_parameters (SSLSP, ID=7/20) API command.

If you have configured I/O capabilities of “Display + Yes/No” for the local device and this event occurs, you must use the smp_send_passkeyreq_response (/PE, ID=7/6) API command to confirm valid comparison. In this case, the passkey argument to that command will be ignored.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`05`	`07`	`05`	None.

Text Info:

Text Name	Event Length	Notes
`PKD`	`0x0014`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle
`uint32`	`passkey`	`P`	Passkey to be displayed (should be displayed to the user in decimal format)

Related Commands:

- smp_send_passkeyreq_response (/PE, ID=7/6)

Related Events:

- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)
- smp_passkey_entry_requested (PKE, ID=7/6)

7.3.7.6 smp_passkey_entry_requested (PKE, ID=7/6)

Remote peer requested passkey entry during pairing.

This event indicates that a remote device has generated and displayed a passkey, which must be entered locally and sent back for comparison. If this occurs, you must reply with the smp_send_passkeyreq_response (/PE, ID=7/6) API command. If the pairing process completes successfully, EZ-Serial will generate the smp_pairing_result (PR, ID=7/3) API event with a success result code (0).

Binary Header:

Type	Length	Group	ID	Notes
`80`	`01`	`07`	`06`	None.

Text Info:

Text Name	Event Length	Notes
`PKE`	`0x0009`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle

Related Commands:

- smp_send_passkeyreq_response (/PE, ID=7/6)

Related Events:

- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)
- smp_passkey_display_requested (PKD, ID=7/5)

7.3.7.7 smp_pin_entry_requested (BTPIN, ID=7/7)

Remote peer requested PIN code during pairing.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`01`	`07`	`07`	None.

Text Info:

Text Name	Event Length	Notes
`BTPIN`	`0x0009`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle for which the PIN code is requested.

Related Commands:

- smp_send_pinreq_response (/BTPIN, ID=7/17)

Related Events:

- smp_pairing_requested (P, ID=7/2)
- smp_pairing_result (PR, ID=7/3)

7.3.7.8 *smp_user_confirmation_requested (UCNFMREQ, ID=7/8)*

Remote peer requested PIN code during pairing.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`01`	`07`	`08`	None.

Text Info:

Text Name	Event Length	Notes
`UCNFMREQ`	`0x000C`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle from which the user confirmation request came in.

Related Commands:

- smp_send_pairreq_response (/PR, ID=7/5)

7.3.8 L2CAP Group (ID=8)

L2CAP methods relate to the Logical Link Control and Adaptation Protocol layer of the Bluetooth stack. These methods are used for working directly with low-level data transfer between two connected devices.

Note: L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The API methods described in this section will not function on devices with only 128K of flash.

Events within this group are listed below:

- l2cap_le_connected (LLEC, ID=8/20)
- l2cap_le_disconnected (LLEDISC, ID=8/21)
- l2cap_le_data_received (LLERX, ID=8/22)
- l2cap_le_congestion_status (LLECNGS, ID=8/23)
- l2cap_le_tx_completed (LLETXC, ID=8/24)

Commands within this group are documented in L2CAP Group (ID=8).

7.3.8.1 l2cap_le_connected (LLEC, ID=8/20)

A LE L2CAP connection has been established with the remote peer.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`0A`	`08`	`14`	None.

Text Info:

Text Name	Event Length	Notes
`LLEC`	`0x0018`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint16`	`cid`	`C`	CID of the connection.
`Macaddr`	`address`	`A`	Peer device’s BD address.
`uint16`	`mtu`	`M`	Peer device’s RX MTU size.

7.3.8.2 l2cap_le_disconnected (LLEDISC, ID=8/21)

A BR/EDR L2CAP connection has been disconnected from the remote peer.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`04`	`08`	`15`	None.

Text Info:

Text Name	Event Length	Notes
`LLEDISC`	`0x0018`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint16`	`cid`	`C`	CID of the disconnected connection.
`uint16`	`result`	`R`	The L2CAP connection disconnect result.

7.3.8.3 l2cap_le_data_received (LLERX, ID=8/22)

A block data received through the LE L2CAP connection from remote peer.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`04`	`08`	`16`	Variable-length command payload.

Text Info:

Text Name	Event Length	Notes
`LLERX`	`0x0018`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint16`	`cid`	`C`	CID of the L2CAP connection from where the data is received.

Data Type	Name	Text	Description
`longuint8a`	`data`	`D`	Received block data.

7.3.8.4 *l2cap_le_congestion_status (LLECNGS, ID=8/23)*

The data transmission congestion status is received.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`03`	`08`	`17`	None.

Text Info:

Text Name	Event Length	Notes
`LLECNGS`	`0x0018`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint16`	`cid`	`C`	CID of the L2CAP connection.
`uint8`	`congested`	`G`	The L2CAP connection is congested (1) or not (0), as reported by the WICED stack. When the congested status is received, host should stop sending the data.

7.3.8.5 *l2cap_le_tx_completed (LLETXC, ID=8/24)*

Received the data transmission completed event.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`04`	`08`	`18`	None.

Text Info:

Text Name	Event Length	Notes
`LLETXC`	`0x0018`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint16`	`cid`	`C`	CID of the L2CAP connection.
`uint8`	`buffer_count`	`N`	Number of transmitted buffer count

7.3.9 GPIO Group (ID=9)

GPIO methods relate to the physical pins on the module.

Event within this group is listed below:

- gpio_interrupt (INT, ID=9/1)

Commands within this group are documented in GPIO Group (ID=9).

7.3.9.1 gpio_interrupt (INT, ID=9/1)

Configured GPIO interrupt has occurred.

This event is generated for GPIO edge changes that have enabled interrupts via the gpio_set_interrupt_mode (SIOI, ID=9/9) API command.

Note: This event is suppressed for pins which have functions enabled using the gpio_set_function (SIOF, ID=9/3) API command. While interrupts occur internally for many functional pins, the interrupt API event is disabled to prevent unintentional or unnecessary API traffic. To allow generation of this event for those pins, disable the function for those pins.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`0B`	`09`	`01`	None.

Text Info:

Text Name	Event Length	Notes
`INT`	`0x0029`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`Port`	`P`	GPIO port
`uint16`	`triggers`	`T`	Triggering pins mask (set bits indicate interrupt source)
`uint16`	`Logic`	`L`	Port logic state mask (set bits indicates HIGH)
`uint32`	`runtime`	`R`	Number of seconds since boot
`uint16`	`fraction`	`F`	Fraction of a second (units are 1/32768)

Related Commands:

- gpio_set_interrupt_mode (SIOI, ID=9/9)

7.3.10 CYSPP Group (ID=10)

CYSPP methods relate to the Cypress Serial Port Profile.

Event within this group is listed below:

- p_cyspp_status (.CYSPP, ID=10/1)

Commands within this group are documented in CYSPP Group (ID=10).

7.3.10.1 p_cyspp_status (.CYSPP, ID=10/1)

CYSPP operational status has changed.

Note: If this event occurs within EZ-Serial and data mode is active (either Bit 0 or Bit 1 set and the CYSPP GPIO pin is not externally de-asserted), then the wired serial interface will be logically disconnected from the API protocol parser and routed to CYSPP data pipe instead. For this reason, this event will never be transmitted out the serial interface with Bit 5 set (0x20), since outgoing API events are suppressed while operating in CYSPP data mode.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`01`	`0A`	`01`	None.

Text Info:

Text Name	Event Length	Notes
`.CYSPP`	`0x000C`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`status`	`S`	CYSPP status bitmask: • Bit 0 (0x01) = Unacknowledged data subscribed • Bit 1 (0x02) = Acknowledged data subscribed • Bit 2 (0x04) = RX flow subscribed • Bit 3 (0x08) = RX flow blocked by remote server • Bit 4 (0x10) = CYSPP peer support verified • Bit 5 (0x20) = Data mode active *(used internally)*

Related Commands:

- p_cyspp_check (.CYSPPCHECK, ID=10/1)
- p_cyspp_start (.CYSPPSTART, ID=10/2)
- p_cyspp_set_parameters (.CYSPPSP, ID=10/3)

Example Usage:

- See Cable Replacement Examples with CYSPP

7.3.11 CYCommand Group (ID=11)

CYCommand methods relate to CYCommand remote configuration channel behavior.

Event within this group is listed below:

- p_cycommand_status (.CYCOM, ID=11/1)

Commands within this group are documented in CYCommand Group (ID=11).

7.3.11.1 p_cycommand_status (.CYCOM, ID=11/1)

CYCommand operational status has changed.

EZ-Serial generates this event when a remote client subscribes to the CYCommand Data characteristic or completes the authentication process, if one has been configured. The event is sent to the external host via the wired interface for alerting the wired host to the change, and is not sent to the remote client.

Note: If this event occurs and Bit 0 is set (data channel active), then the wired serial interface is logically disconnected from the API protocol parser. Any serial data sent to the module while it is in API command mode with CYCommand data mode active will be buffered (up to 136 bytes) and delivered to the parser only after the remote client disconnects or unsubscribes from the data channel.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`01`	`0B`	`01`	None.

Text Info:

Text Name	Event Length	Notes
`.CYCOM`	`0x000C`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`status`	`S`	CYCommand status bitmask: • Bit 0 (0x01) = Data mode active • Bit 1 (0x02) = Data subscribed • Bit 2 (0x04) = Authentication complete • Bit 7 (0x80) = Challenge Data subscribed

Related Commands:

- p_cycommand_set_parameters (.CYCOMSP, ID=11/1)

7.3.12 iBeacon Group (ID=12)

iBeacon methods relate to iBeacon setup and operation.

There are currently no API events related to iBeacon functionality. Commands within this group are documented in iBeacon Group (ID=12).

7.3.13 Eddystone Group (ID=13)

Eddystone methods relate to Eddystone beacon setup and operation.

There are currently no API events related to Eddystone functionality. Commands within this group are documented in Eddystone Group (ID=13).

7.3.14 BT Group (ID=14)

BT methods relate to the Bluetooth ER/EDR operations of the Bluetooth stack, which includes management of inquiry, connection, and profile relative events.

Events within this group are listed below:

- bt_inquiry_result (BTIR, ID=14/1)
- bt_name_result (BTINR, ID=14/2)
- bt_inquiry_complete (BTIC, ID=14/3)
- bt_connected (BTCON, ID=14/4)
- bt_connection_status (BTCS, ID=14/5)
- bt_connection_failed (BTCF, ID=14/6)
- bt_disconnected (BTDIS, ID=14/7)
- bt_profile_connected (BTPCON, ID=14/8)
- bt_profile_disconnected (BTPDIS, ID=14/9)

Commands within this group are documented in BT Group (ID=14).

7.3.14.1 bt_inquiry_result (BTIR, ID=14/1)

Details about a BT device response packet for BT inquiry.

Binary Header:

Type	Length	Group	ID	Notes
80	15	0E	01	None.

Text Info:

Text Name	Event Length	Notes
BTIR	0x0003C	None.

Event Parameters:

Data Type	Name	Text	Description
macaddr	bt_addr	A	Bluetooth address of remote device.
uint32	dev_class	C	Class of remote device.
uint8	rssi	R	Receive signal strength index value in dBm (between -95 and +10). If the received value is 0x7F, it means that the RSSI value is not supplied and can be ignored.
uint8	eir_complte	E	EIR array is completed or not. • 0x00 = EIR array in eir_uuid is not a completed list. • 0x01 = EIR array in eir_uuid is a completed list.
uint8a	eir_uuid	U	Array of EIR UUIDs bit map mask, fixed 8 bytes in length.

Data Type	Name	Text	Description
			If the bit is set, it indicates that the EIR UUID exists. Note that the detail of the stored EIR data bit mask can refer to the definition of the enumeration values of *BTM_EIR_UUID_ENUM* which defined the *wiced_bt_dev.h* header file in WICED Studio SDK.

Related Commands:

- bt_start_inquiry (/BTI, ID=14/1)

7.3.14.2 bt_name_result (BTINR, ID=14/2)

The BT-friendly name read from remote device.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`08-48`	`0E`	`02`	None.

Text Info:

Text Name	Event Length	Notes
`BTINR`	`0x0003C`	None.

Event Parameters:

Data Type	Name	Text	Description
`macaddr`	`bt_addr`	`A`	Bluetooth address of remote device.
`uint16`	`status`	`S`	BT query name operation status. • 0 - Indicates success. The name field includes the name of the remote BT device. • Non-zero - Indicates query name failure, the name field should be ignored.
`string`	`name`	`N`	Name of the remote device, variable length.

Related Commands:

- bt_query_name (/BTQN, ID=14/3)

7.3.14.3 bt_inquiry_complete (BTIC, ID=14/3)

BT inquiry operation timed out or stopped by the Bluetooth stack.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`00`	`0E`	`03`	None.

Text Info:

Text Name	Event Length	Notes
`BTIC`	`0x0003C`	None.

Event Parameters:

None.

Related Commands:

- bt_start_inquiry (/BTI, ID=14/1)

7.3.14.4 bt_connected (BTCON, ID=14/4)

Connection established with a remote device.

Binary Header:

Type	Length	Group	ID	Notes
80	08	0E	04	None.

Text Info:

Text Name	Event Length	Notes
BTCON	0x0003C	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle of the established ER/EDR connection.
macaddr	bt_addr	A	Bluetooth address of peer device.
uint8	bond	B	Indicates whether the peer device has been bonded: • 0x00 = Not bonded • 0x01 = Bonded

Related Commands:

- bt_connect (/BTC, ID=14/4)

Note that this event will also be reported when BT SPP connection is established through PC.

7.3.14.5 bt_connection_status (BTCS, ID=14/5)

Connection status updated with a remote device.

Binary Header:

Type	Length	Group	ID	Notes
80	0B	0E	05	None.

Text Info:

Text Name	Event Length	Notes
BTCS	0x0003C	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_hanndle	C	Connection handle of the established ER/EDR profile connection.
macaddr	bt_addr	A	Bluetooth address of the peer device to which the connection established.
uint8	bt_profile_type	P	BT profile type. • 0x00 = SPP profile type • 0x01 = A2DP profile type • 0x02 = AVRCP profile type • 0x03 = HFP profile type • 0x04 = HID profile type • 0xFF = Basic link layer connection or unknown BT profile type
uint8	bond	B	Indicates the peer device has been bonded or not. • 0x00 = Not bonded • 0x01 = Bonded
uint8	role	R	Indicates the link role of the peer device (for BLE only, otherwise set to 0xFF). • 0x00 = Master • 0x01 = Slave

Data Type	Name	Text	Description
			• 0xFF = Link role unknown
uint8	sniff	S	Indicates the sniff status of the peer device. • 0 = Active mode • 2 = Sniff mode • 4 = Sniff sub-rating notification status • 5 = Pending (waiting for status from controller) • 6 = Error (controller returned error) • 7 = Smart Sniff

Related Commands:

- bt_connect (/BTC, ID=14/4)
- bt_disconnect (/BTDIS, ID=14/6)

7.3.14.6 bt_connection_failed (BTCF, ID=14/6)

Connection failed with a remote device.

Binary Header:

Type	Length	Group	ID	Notes
80	03	0E	06	None.

Text Info:

Text Name	Event Length	Notes
BTCF	0x0003C	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_hanndle	C	Connection handle of the ER/EDR connection.
Macaddr	bt_addr	A	Bluetooth address of peer device.
uint16	reason	R	Reason that causes the BT connection to fail.

Related Commands:

- bt_connect (/BTC, ID=14/4)

Note that this event will also be reported when PC tries to establish a BT SPP connection and encounters an error.

7.3.14.7 bt_disconnected (BTDIS, ID=14/7)

Connection disconnected from the remote device.

Binary Header:

Type	Length	Group	ID	Notes
80	03	0E	07	None.

Text Info:

Text Name	Event Length	Notes
BTDIS	0x0012	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_hanndle	C	Connection handle of the ER/EDR connection.
uint16	reason	R	Reason for the disconnection event.

Related Commands:

- bt_disconnect (/BTDIS, ID=14/6)

Note that when BT SPP connection is disconnected from PC, this event will also be reported.

7.3.14.8 bt_profile_connected (BTPCON, ID=14/8)

A profile connection established with a remote device.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`0B`	`0E`	`08`	None.

Text Info:

Text Name	Event Length	Notes
`BTPCON`	`0x002C`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_handle`	`C`	Connection handle of the established ER/EDR connection.
`macaddr`	`bt_addr`	`A`	Bluetooth address of peer device.
`uint8`	`bt_profile_type`	`P`	BT profile type. 0x00 = SPP profile type • 0x01 = A2DP profile type • 0x02 = AVRCP profile type • 0x03 = HFP profile type • 0x04 = HID profile type • 0xFF = Basic link layer connection or unknown BT profile type
`uint8`	`bond`	`B`	Indicates whether the peer device has been bonded. • 0x00 = Not bonded • 0x01 = Bonded
`uint16`	`handle`	`H`	Profile connection handle. Note that one profile may establish multiple profile connection handles.

Related Commands:

- bt_connect (/BTC, ID=14/4)

Note that this event will also be reported when BT SPP connection is established through PC.

7.3.14.9 bt_profile_disconnected (BTPDIS, ID=14/9)

A profile connection disconnected with a remote device.

Binary Header:

Type	Length	Group	ID	Notes
`80`	`06`	`0E`	`09`	None.

Text Info:

Text Name	Event Length	Notes
`BTPDIS`	`0x001F`	None.

Event Parameters:

Data Type	Name	Text	Description
`uint8`	`conn_hanndle`	`C`	Connection handle of the ER/EDR connection.
`uint8`	`bt_profile_type`	`P`	BT profile type. • 0xFF = Basic link layer connection or unknown BT profile type • 0x00 = SPP profile type • 0x01 = A2DP profile type • 0x02 = AVRCP profile type • 0x03 = HFP profile type • 0x04 = HID profile type
`uint16`	`handle`	`H`	Profile connection handle.
`uint16`	`reason`	`R`	Reason for the disconnection event.

Related Commands:

- bt_disconnect (/BTDIS, ID=14/6)

Note that this event will also be reported when BT SPP connection is disconnected from PC.

7.4 Error Codes

7.4.1 EZ-Serial System Error Codes

Table 7-5 lists the result/error codes generated by EZ-Serial. See the command and event reference material in API Commands and Responses and API Events for specific details on each result within the context of the responses and events where they are triggered.

Code (Hex)	Name	Description
`0000`	EZS_ERR_SUCCESS	Operation successful, no error
`0100`	EZS_ERR_CORE	Core system error category
`0101`	EZS_ERR_CORE_NULL_POINTER	Null pointer encountered *(internal error)*
`0102`	EZS_ERR_CORE_MALLOC_FAILED	Memory allocation failed *(internal error)*
`0103`	EZS_ERR_CORE_BUFFER_OVERFLOW	Buffer overflow *(internal error)*
`0104`	EZS_ERR_CORE_FEATURE_NOT_IMPLEMENTED	Unsupported feature *(internal error)*
`0105`	EZS_ERR_CORE_TASK_SCHEDULE_OVERFLOW	Task scheduling attempted but schedule is full
`0106`	EZS_ERR_CORE_TASK_QUEUE_OVERFLOW	Task queue attempted but queue is full
`0107`	EZS_ERR_CORE_INVALID_STATE	Invalid state for requested operation
`0108`	EZS_ERR_CORE_OPERATION_NOT_PERMITTED	Operation not permitted
`0109`	EZS_ERR_CORE_INSUFFICIENT_RESOURCES	Insufficient resources for requested action
`010A`	EZS_ERR_CORE_FLASH_WRITE_NOT_PERMITTED	Unable to perform flash write at this time
`010B`	EZS_ERR_CORE_FLASH_WRITE_FAILED	Flash write operation failed during write
`010C`	EZS_ERR_CORE_HARDWARE_FAILURE	Internal chipset hardware failure
`010D`	EZS_ERR_CORE_BLE_INITIALIZATION_FAILED	Could not initialize BLE stack
`010E`	EZS_ERR_CORE_REPEATED_ATTEMPTS	Repeated attempts to initialize BLE stack
`010F`	EZS_ERR_CORE_TX_POWER_READ	Could not read radio TX power
`0110`	EZS_ERR_CORE_DB_VERIFICATION_FAILED	Verification prevented custom attribute addition
`0200`	EZS_ERR_PROTOCOL	Protocol error category
`0201`	EZS_ERR_PROTOCOL_UNRECOGNIZED_PACKET_TYPE	Unsupported packet type for text parsing *(internal error)*
`0202`	EZS_ERR_PROTOCOL_UNRECOGNIZED_ARGUMENT_TYPE	Unsupported argument type for text parsing *(internal error)*
`0203`	EZS_ERR_PROTOCOL_UNRECOGNIZED_COMMAND	Command group/method not valid or unrecognized
`0204`	EZS_ERR_PROTOCOL_UNRECOGNIZED_RESPONSE	Response group/method invalid or unrecognized *(internal error)*

Code (Hex)	Name	Description
0205	EZS_ERR_PROTOCOL_UNRECOGNIZED_EVENT	Event group/method invalid or unrecognized *(internal error)*
0206	EZS_ERR_PROTOCOL_SYNTAX_ERROR	Syntax error while parsing text command
0207	EZS_ERR_PROTOCOL_COMMAND_TIMEOUT	Binary command packet transmission not completed in required time
0208	EZS_ERR_PROTOCOL_RESPONSE_PENDING	Command already sent but response still pending
0209	EZS_ERR_PROTOCOL_INVALID_CHECKSUM	Binary command packet has invalid checksum
020A	EZS_ERR_PROTOCOL_INVALID_COMMAND_LENGTH	Command length is greater than maximum
020B	EZS_ERR_PROTOCOL_INVALID_PARAMETER_COUNT	Incorrect number of parameters provided
020C	EZS_ERR_PROTOCOL_INVALID_PARAMETER_VALUE	Command parameter outside of acceptable range
020D	EZS_ERR_PROTOCOL_MISSING_REQUIRED_ARGUMENT	Text-mode command missing required arguments
020E	EZS_ERR_PROTOCOL_INVALID_HEXADECIMAL_DATA	Invalid hexadecimal data provided (not 0-9, A-F)
020F	EZS_ERR_PROTOCOL_INVALID_ESCAPE_SEQUENCE	Invalid escape sequence
0210	EZS_ERR_PROTOCOL_INVALID_MACRO_SEQUENCE	Invalid macro sequence
0211	EZS_ERR_PROTOCOL_FLASH_SETTINGS_PROTECTED	Attempted direct flash write of protected setting
0300	EZS_ERR_GPIO	GPIO error category
0301	EZS_ERR_GPIO_PORT_NOT_SUPPORTED	Selected port in GPIO command not supported
0400	EZS_ERR_LL	Link layer error category
0401	EZS_ERR_LL_CONTROLLER_BUSY	Link layer controller busy
0402	EZS_ERR_LL_NO_DEVICE_ENTITY	Device entity not available
0403	EZS_ERR_LL_NOT_IN_BOND_LIST	Device not found in bond list
0404	EZS_ERR_LL_DEVICE_ALREADY_EXISTS	Device already exists
0500	EZS_ERR_GAP	GAP error category
0501	EZS_ERR_GAP_INVALID_CONNECTION_HANDLE	Invalid connection handle specified
0502	EZS_ERR_GAP_CONNECTION_REQUIRED	Connection required, but none is available
0503	EZS_ERR_GAP_ROLE	Incorrect GAP role for this operation
0504	EZS_ERR_GAP_ADV_QUEUE_OVERFLOW	Advertisement queue attempted but queue is full
0600	EZS_ERR_GATT	GATT error category
0601	EZS_ERR_GATT_INVALID_ATTRIBUTE_HANDLE	Invalid attribute handle for GATT operation
0602	EZS_ERR_GATT_READ_NOT_PERMITTED	Read not permitted on this attribute
0603	EZS_ERR_GATT_WRITE_NOT_PERMITTED	Write not permitted on this attribute
0604	EZS_ERR_GATT_INVALID_PDU	Invalid PDU for requested operation
0605	EZS_ERR_GATT_INSUFFICIENT_AUTHENTICATION	Insufficient authentication for requested operation
0606	EZS_ERR_GATT_REQUEST_NOT_SUPPORTED	Request not supported
0607	EZS_ERR_GATT_INVALID_OFFSET	Invalid offset specified for requested operation
0608	EZS_ERR_GATT_INSUFFICIENT_AUTHORIZATION	Insufficient authorization for requested operation
0609	EZS_ERR_GATT_PREPARE_WRITE_QUEUE_FULL	Prepare write queue full, cannot prepare new write
060A	EZS_ERR_GATT_ATTRIBUTE_NOT_FOUND	Attribute not found in database
060B	EZS_ERR_GATT_ATTRIBUTE_NOT_LONG	Attribute not long when long operation requested
060C	EZS_ERR_GATT_INSUFFICIENT_ENC_KEY_SIZE	Insufficient encryption key size
060D	EZS_ERR_GATT_INVALID_ATTRIBUTE_LENGTH	Invalid attribute length
060E	EZS_ERR_GATT_UNLIKELY_ERROR	Unlikely error occurred, unknown cause
060F	EZS_ERR_GATT_INSUFFICIENT_ENCRYPTION	Insufficient encryption for requested operation
0610	EZS_ERR_GATT_UNSUPPORTED_GROUP_TYPE	Unsupported group type specified in Read By Group Type operation
0611	EZS_ERR_GATT_INSUFFICIENT_RESOURCES	Insufficient resources to perform operation
0680	EZS_ERR_GATT_CLIENT_NOT_SUBSCRIBED	Client has not subscribed to updates on characteristic (local error code when sending notifications or indications)

Code (Hex)	Name	Description
0700	EZS_ERR_L2CAP	L2CAP error category
0701	EZS_ERR_L2CAP_NOT_IN_BOND_LIST	Device not found in bond list
0702	EZS_ERR_L2CAP_PSM_WRONG_ENCODING	Wrong L2CAP PSM encoding
0703	EZS_ERR_L2CAP_PSM_ALREADY_REGISTERED	L2CAP PSM already registered
0704	EZS_ERR_L2CAP_PSM_NOT_REGISTERED	L2CAP PSM not registered
0705	EZS_ERR_L2CAP_CONNECTION_ENTITY_NOT_FOUND	L2CAP connection entity not found
0706	EZS_ERR_L2CAP_CHANNEL_NOT_FOUND	L2CAP channel not found
0707	EZS_ERR_L2CAP_PSM_NOT_IN_RANGE	L2CAP PSM is not in range
0800	EZS_ERR_SMP	SMP error category
0801	EZS_ERR_SMP_OOB_NOT_AVAILABLE	Out-of-band pairing data not available
0802	EZS_ERR_SMP_SECURITY_OPERATION_FAILED	Security operation failed
0803	EZS_ERR_SMP_MIC_AUTH_FAILED	Message integrity check authentication failed
0900	EZS_ERR_SPEC	Bluetooth Core Specification error category
0901	EZS_ERR_SPEC_UNKNOWN_HCI_COMMAND	Unknown HCI Command
0902	EZS_ERR_SPEC_UNKNOWN_CONNECTION_IDENTIFIER	Unknown Connection Identifier
0903	EZS_ERR_SPEC_HARDWARE_FAILURE	Hardware Failure
0904	EZS_ERR_SPEC_PAGE_TIMEOUT	Page Timeout
0905	EZS_ERR_SPEC_AUTHENTICATION_FAILURE	Authentication Failure
0906	EZS_ERR_SPEC_PIN_OR_KEY_MISSING	PIN or Key Missing
0907	EZS_ERR_SPEC_MEMORY_CAPACITY_EXCEEDED	Memory Capacity Exceeded
0908	EZS_ERR_SPEC_CONNECTION_TIMEOUT	Connection Timeout
0909	EZS_ERR_SPEC_CONNECTION_LIMIT_EXCEEDED	Connection Limit Exceeded
090A	EZS_ERR_SPEC_SYNCHRONOUS_CONN_LIMIT _DEVICE_EXCEEDED	Synchronous Connection Limit to a Device Exceeded
090B	EZS_ERR_SPEC_ACL_CONNECTION_ALREADY_EXISTS	ACL Connection Already Exists
090C	EZS_ERR_SPEC_COMMAND_DISALLOWED	Command Disallowed
090D	EZS_ERR_SPEC_CONNECTION_REJECTED _LIMITED_RESOURCES	Connection Rejected due to Limited Resources
090E	EZS_ERR_SPEC_CONNECTION_REJECTED _SECURITY_REASONS	Connection Rejected due to Security Reasons
090F	EZS_ERR_SPEC_CONNECTION_REJECTED _UNACCEPTABLE_BDADDR	Connection Rejected due to Unacceptable BD_ADDR
0910	EZS_ERR_SPEC_CONNECTION_ACCEPT _TIMEOUT_EXCEEDED	Connection Accept Timeout Exceeded
0911	EZS_ERR_SPEC_UNSUPPORTED_FEATURE _OR_PARAMETER_VALUE	Unsupported Feature or Parameter Value
0912	EZS_ERR_SPEC_INVALID_HCI_COMMAND_PARAMETERS	Invalid HCI Command Parameters
0913	EZS_ERR_SPEC_REMOTE_USER_TERMINATED _CONNECTION	Remote User Terminated Connection
0914	EZS_ERR_SPEC_REMOTE_DEVICE_TERMINATED _LOW_RESOURCES	Remote Device Terminated Connection due to Low Resources
0915	EZS_ERR_SPEC_REMOTE_DEVICE_TERMINATED _POWER_OFF	Remote Device Terminated Connection due to Power Off
0916	EZS_ERR_SPEC_CONNECTION_TERMINATED _BY_LOCAL_HOST	Connection Terminated by Local Host
0917	EZS_ERR_SPEC_REPEATED_ATTEMPTS	Repeated Attempts
0918	EZS_ERR_SPEC_PAIRING_NOT_ALLOWED	Pairing Not Allowed
0919	EZS_ERR_SPEC_UNKNOWN_LMP_PDU	Unknown LMP PDU
091A	EZS_ERR_SPEC_UNSUPPORTED_REMOTE _LMP_FEATURE	Unsupported Remote Feature / Unsupported LMP Feature
091B	EZS_ERR_SPEC_SCO_OFFSET_REJECTED	SCO Offset Rejected
091C	EZS_ERR_SPEC_SCO_INTERVAL_REJECTED	SCO Interval Rejected

Code (Hex)	Name	Description
091D	EZS_ERR_SPEC_SCO_AIR_MODE_REJECTED	SCO Air Mode Rejected
091E	EZS_ERR_SPEC_INVALID_LMP_LL_PARAMETERS	Invalid LMP Parameters / Invalid LL Parameters
091F	EZS_ERR_SPEC_UNSPECIFIED_ERROR	Unspecified Error
0920	EZS_ERR_SPEC_UNSUPPORTED_LMP_LL _PARAMTER_VALUE	Unsupported LMP Parameter Value / Unsupported LL Parameter Value
0921	EZS_ERR_SPEC_ROLE_CHANGE_NOT_ALLOWED	Role Change Not Allowed
0922	EZS_ERR_SPEC_LMP_LL_RESPONSE_TIMEOUT	LMP Response Timeout / LL Response Timeout
0923	EZS_ERR_SPEC_LMP_ERROR_TRANSACTION_COLLISION	LMP Error Transaction Collision
0924	EZS_ERR_SPEC_LMP_PDU_NOT_ALLOWED	LMP PDU Not Allowed
0925	EZS_ERR_SPEC_ENCRYPTION_MODE_NOT_ACCEPTABLE	Encryption Mode Not Acceptable
0926	EZS_ERR_SPEC_LINK_KEY_CANNOT_BE_CHANGED	Link Key cannot be Changed
0927	EZS_ERR_SPEC_REQUESTED_QOS_NOT_SUPPORTED	Requested QoS Not Supported
0928	EZS_ERR_SPEC_INSTANT_PASSED	Instant Passed
0929	EZS_ERR_SPEC_PAIRING_WITH_UNIT_KEY _NOT_SUPPORTED	Pairing with Unit Key Not Supported
092A	EZS_ERR_SPEC_DIFFERENT_TRANSACTION_COLLISION	Different Transaction Collision
092B	/* 0x2B reserved */	Reserved
092C	EZS_ERR_SPEC_QOS_UNACCEPTABLE_PARAMETER = 0x092C	QoS Unacceptable Parameter
092D	EZS_ERR_SPEC_QOS_REJECTED	QoS Rejected
092E	EZS_ERR_SPEC_CHANNEL_CLASSIFICATION _NOT_SUPPORTED	Channel Classification Not Supported
092F	EZS_ERR_SPEC_INSUFFICIENT_SECURITY	Insufficient Security
0930	EZS_ERR_SPEC_PARAMETER_OUT_OF _MANDATORY_RANGE	Parameter Out Of Mandatory Range
0931	/* 0x31 reserved */	Reserved
0932	EZS_ERR_SPEC_ROLE_SWITCH_PENDING = 0x0932	Role Switch Pending
0933	/* 0x33 reserved */	Reserved
0934	EZS_ERR_SPEC_RESERVED_SLOT_VIOLATION = 0x0934	Reserved Slot Violation
0935	EZS_ERR_SPEC_ROLE_SWITCH_FAILED	Role Switch Failed
0936	EZS_ERR_SPEC_EXTENDED_INQUIRY_RSP_TOO_LARGE	Extended Inquiry Response Too Large
0937	EZS_ERR_SPEC_SSP_NOT_SUPPORTED_BY_HOST	Secure Simple Pairing Not Supported By Host
0938	EZS_ERR_SPEC_HOST_BUSY_PAIRING	Host Busy - Pairing
0939	EZS_ERR_SPEC_CONNECTION_REJECTED _NO_SUITABLE_CHANNEL	Connection Rejected due to No Suitable Channel Found
093A	EZS_ERR_SPEC_CONTROLLER_BUSY	Controller Busy
093B	EZS_ERR_SPEC_UNACCEPTABLE _CONNECTION_PARAMETERS	Unacceptable Connection Parameters
093C	EZS_ERR_SPEC_DIRECTED_ADVERTISING_TIMEOUT	Directed Advertising Timeout
093D	EZS_ERR_SPEC_CONNECTION_TERMINATED _MIC_FAILURE	Connection Terminated due to MIC Failure
093E	EZS_ERR_SPEC_CONNECTION_FAILED _TO_BE_ESTABLISHED	Connection Failed to be Established
093F	EZS_ERR_SPEC_MAC_CONNECTION_FAILED	MAC Connection Failed
0940	EZS_ERR_SPEC_COARSE_CLOCK_ADJ_REJECTED	Coarse Clock Adjustment Rejected but Will Try to Adjust Using Clock Dragging
0A00	EZS_ERR_BT	Bluetooth BR/EDR error category
0A01	EZS_ERR_BT_ERROR	Unsorted BT error code.
0A02	EZS_ERR_BT_BUSY	BR/EDR is busying, already in progress
0A03	EZS_ERR_BT_WRONG_MODE	BR/EDR device is not up
0A04	EZS_ERR_BT_ILLEGAL_VALUE	Parameter(s) are out of range

Code (Hex)	Name	Description
`0A05`	EZS_ERR_BT_SERVICE_NOT_FOUND	BT profile service not found.
`EEEE`	EZS_ERR_UNKNOWN	Unknown problem *(internal error)*

Table 7-5. EZ-Serial System Error Codes

7.4.2 EZ-Serial GATT Database Validation Error Codes

Table 7-6 lists the result/error codes generated by EZ-Serial during dynamic GATT database validation. For details, see Defining Custom Local GATT Services and Characteristics) and the documentation for the related GATT Server Group (ID=5) API command methods.

Code (Hex)	Name	Description
`0000`	GATTS_DB_VALID_OK	Validation passed with no warnings or errors
`0001`	GATTS_DB_VALID_WARNING_NOT_ENOUGH_ATTRIBUTES	Structure is valid, but more attributes are required
`0002`	GATTS_DB_VALID_ERROR_ATTRIBUTE_LIMIT_EXCEEDED	Attribute count limit exceeded
`0003`	GATTS_DB_VALID_ERROR_ATTRIBUTE_DATA_EXCEEDED	Runtime attribute value data byte limit exceeded
`0004`	GATTS_DB_VALID_ERROR_CONSTANT_DATA_EXCEEDED	Constant default data byte limit exceeded
`0005`	GATTS_DB_VALID_ERROR_CCCD_LIMIT_EXCEEDED	CCCD attribute limit exceeded
`0006`	GATTS_DB_VALID_ERROR_SVC_DECL_REQUIRED	Service declaration required
`0007`	GATTS_DB_VALID_ERROR_UNEXPECTED_SVC_DECL	Unexpected service declaration
`0008`	GATTS_DB_VALID_ERROR_CHAR_DECL_REQUIRED	Characteristic declaration required
`0009`	GATTS_DB_VALID_ERROR_UNEXPECTED_CHAR_DECL	Unexpected characteristic declaration
`000A`	GATTS_DB_VALID_ERROR_CHAR_VALUE_REQUIRED	Characteristic value attribute required
`000B`	GATTS_DB_VALID_ERROR_UNEXPECTED_DESCRIPTOR	Specified descriptor not allowed at this position
`000C`	GATTS_DB_VALID_ERROR_INVALID_ATT_PROPERTIES	Attribute properties not compatible with type
`000D`	GATTS_DB_VALID_ERROR_INVALID_ATT_LENGTH	Invalid attribute length
`000E`	GATTS_DB_VALID_ERROR_INVALID_ATT_DATA	Attribute data not compatible with type

Table 7-6. EZ-Serial GATT Validation Error Codes

7.5 Macro Definitions

Macros in EZ-Serial are simple codes which result in text substitution within the parser. Macros may be used in either text mode or binary mode. Macros always begin with the '%' character and are followed by one or more alphanumeric characters (A-Z, 0-9). Macros are not case sensitive.

Code	Description	Example Input	Example Output	Notes
`%M1`	Byte #1 of local public MAC address	`MyDevice %M1`	`MyDevice 00`	Examples assume that the local device has a public MAC address of `00:A0:50:E3:83:5F`.
`%M2`	Byte #2 of local public MAC address	`MyDevice %M2`	`MyDevice A0`	
`%M3`	Byte #3 of local public MAC address	`MyDevice %M3`	`MyDevice 50`	
`%M4`	Byte #4 of local public MAC address	`MyDevice %M4`	`MyDevice E3`	
`%M5`	Byte #5 of local public MAC address	`MyDevice %M5`	`MyDevice 83`	
`%M6`	Byte #6 of local public MAC address	`MyDevice %M6`	`MyDevice 5F`	

Macros may be used in series with or without special separators, if the entire macro code (including the '%' byte) remains intact. For example, to use the last three bytes of the MAC address in the same string, separated by the ':' byte, use the following:

```
MyDevice %M4:%M5:%M6
```

This string is particularly useful for setting a module-specific device name using the gap_set_device_name (SDN, ID=4/15) API command without needing to query or track the MAC address separately by hand.

8 GPIO Reference

This section describes various GPIO connections provided by the EZ-Serial firmware on supported modules. This section also provides details on the default boot state and the expected behavior in different operational modes.

8.1 GPIO Pin Map for Supported Modules

The EZ-Serial firmware can be run on multiple EZ-BT WICED modules, some of which have unique pin configurations. The assignment of special functions for supported modules is described in Table 8-1.

Each pin is shown with its assigned module pin and the effective pin when use the CY8CKIT-042 BLE Pioneer Kit. Some pins on the EZ-BT Module Arduino Evaluation Boards as CYBT-413034-EVAL are remapped from the module pin to the evaluation kit pin to provide more flexibility when designed with WICED Studio or ModusToolbox IDE and the BLE Pioneer Kit. Pins which have been remapped on evaluation modules are shown in **bold** in Table 8-1.

	Pin Name	**Pin Assignment**					
		CYBT-413034-02		**CYBT-413055-02**		**CYBT-413061-02**	
		CYBT-413034-EVAL	Pioneer	CYBT-413034-EVAL	Pioneer	CYBT-413034-EVAL	Pioneer
DIGITAL FUNCTIONS	**UART_RX**	P4	P0.2	P4	P0.2	P4	P0.2
	UART_TX	P33	P3.3	P33	P3.3	P33	P3.3
	UART_RTS	P6	P3.0	P6	P3.0	P6	P3.0
	UART_CTS	P7	P0.5	P7	P0.5	P7	P0.5
	ATEN_SHDN	P25	P3.2	P25	P3.2	P25	P3.2
	CONNECTION	P10	P1.0	P10	P1.0	P10	P1.0
	CP_ROLE	P0	P1.2	P0	P1.2	P0	P1.2
	CYSPP	P38	P1.3	P38	P1.3	P38	P1.3
	DATA_READY	P34	P1.1	P34	P1.1	P34	P1.1
	FACTORY_TR	P34	P1.1	P34	P1.1	P34	P1.1
	LP_MODE	P2	P2.3	P2	P2.3	P2	P2.3
	LP_STATUS	P29	P3.5/P3.7	P29	P3.5/P3.7	P29	P3.5/P3.7
PWM	**PWM0**[2]	P26[1]	P2.4/P2.6	P26[2]	P2.4/P2.6	P26[2]	P2.4/P2.6
ADC	**ADC0**[2]	P28[3]	P3.4/P3.6	P28[3]	P3.4/P3.6	P28[3]	P3.4/P3.6

Table 8-1. GPIO Pin Map on Supported Modules

[1] PWM signal channel can be routed to any usable GPIO pin through the internal supermatric, so there is no limitation on the GPIO pin usage for any PWM channel. See gpio_set_pwm_mode (SPWM, ID=9/11) for all supported PWM channels. Here, PWM0 pin configuration shown in Table 8-1 is an example. In the PWM0 example, GPIO pin was connected to P26 which is connected to the LED2 by default.

[2] See gpio_query_adc (/QADC, ID=9/2) for all supported ADC channels and mapped GPIO pins. Here, ADC0 pin configuration shown in Table 8-1 is an example.

8.2 GPIO Pin Functionality

EZ-Serial provides 12 special-function digital GPIO pins, four optional PWM output pins for generating flexible PWM signals, and one optional analog input pin for ADC reads.

8.2.1 Digital Special-Function Pins

Table 8-2 details the functionality of each digital function GPIO pin. Pins with the "Optional" column showing **Yes** may have their special functionality disabled using the gpio_set_function (SIOF, ID=9/3) API command, which will allow them to be configured as GPIOs and used for API-based input, output, or interrupts.

Pin Name	Direction	Details	Optional
UART_RX	Input	UART Communication RX signal for incoming data from external host device.	No
UART_TX	Output	UART Communication TX signal for outgoing data to external host device	No
UART_RTS	Output	UART Communication RTS signal signifying local receive permission (flow control)	Yes
UART_CTS	Input	UART Communication CTS signal detecting remote receive permission (flow control)	Yes
ATEN_SHDN	In/Out	**Description:** Open-drain active LOW bi-directional signal. If the host drives this pin LOW while it is in the idle (HIGH) state, EZ-Serial will immediately stop any activity, including the closure of any open connection, and force hibernation. Both the radio and CPU will remain completely inactive while in this state. If the module drives this pin LOW while it is in the idle (HIGH) state, it indicates an internal RX or TX serial buffer overflow depending on the context. Particularly when flow control is not used, it is impossible to avoid data loss in some high-demand cases due to limited SRAM buffering capability on the module and/or on the host device. This GPIO signal exists to give the host a way to be notified of such data loss has occurred, to handle this case as the application requires. **Status indicator logic (active-low output):** • **LOW** – depending on state: ○ While host is sending serial data, RX buffer overflow resulting in loss of data being sent from the host. ○ While host is idle or reading serial data, TX buffer overflow resulting in loss of data waiting to transmit to the host. • **HIGH** – internal buffers have not overflowed. **Control signal logic (active-low input):** • **LOW** – Forced hibernation mode, CPU and radio are inactive. • **HIGH** – CPU and radio activity allowed. **Default boot state:** • **HIGH** (idle, no buffer overflow, CPU/radio activity allowed) **Note:** The function of this pin is not supported on WICED platform currently. By default, the logic is HIGH. Pulling this pin to LOW has no effect so currently ignore the logic of this pin.	Yes
CONNECTION	Output	**Description:** BLE connection or CYSPP data pipe readiness status. When the CYSPP pin is asserted, the external host can use this pin to detect when data sent to the module will be immediately transmitted to the remote peer. **Status indicator logic (active-low):** • When CYSPP pin is de-asserted (API command mode active) ○ **LOW** – remote BLE or BR/EDR peer device is connected. ○ **HIGH** – no remote BLE or BR/EDR peer device is connected. • When CYSPP pin is asserted (CYSPP mode active) ○ **LOW** – CYSPP data stream fully available (connected and ready). ○ **HIGH** – CYSPP data stream not available (disconnected or not ready). **Default boot state:** • **HIGH** (no connection)	Yes

Pin Name	Direction	Details	Optional
CP_ROLE	In/Out	**Description:** Central or peripheral GAP role selection for CSYPP operation. The external host can use this pin to select the role that the module should use for CYSPP behavior. This pin is also internally pulled HIGH or LOW based on software-triggered GAP behavioral state. **Control signal logic (active-low):** • **LOW** – CYSPP mode will operate as a GAP central device (scan and connect) • **HIGH** – CYSPP mode will operate as a GAP peripheral device (advertise and wait) **Status indicator logic (internally pulled, may be overridden by external signals):** • **LOW** – Connected as a GAP central device if CONNECTION pin is also LOW. • **HIGH** – Connected as a GAP peripheral device if CONNECTION pin is also LOW. **Default boot state:** • Internally pulled **HIGH** (peripheral role selection for CYSPP operation)	Yes
CYSPP	In/Out	**Description:** CYSPP mode control. The external host can use this pin to begin automatic CYSPP operation without the need for any API commands. This pin is also internally pulled HIGH or LOW based on software-triggered entry or exit to and from CYSPP data mode. **Control signal logic (active-low):** • **LOW** – module enters CYSPP data mode. • **HIGH** – module exits CYSPP data mode and returns to API command mode. **Status indicator logic (internally pulled, may be overridden by external signals):** • **LOW** – API commands or remote BLE client GATT client transactions have entered CYSPP data mode. • **HIGH** – API commands or remote BLE peer GATT client transactions have exited CYSPP data mode. **Default boot state:** • Internally pulled **HIGH** (command mode active, CYSPP data mode inactive)	No
DATA_READY	Output	**Description:** The external host can use this as an interrupt signal, which is especially useful if the host cannot wake up on UART activity. This signal will be asserted regardless of whether the available outgoing data is an API response or event (command mode) or serial data from a remote peer (CYSPP mode). When used in combination with flow control and the module's CTS pin, a host can efficiently manage the module's data flow in tandem with its own sleep requirements. **Status indicator logic (active-low):** • **LOW** – data is ready to be sent to the external host. • **HIGH** – all data has been transmitted. **Default boot state:** • **HIGH**, but quickly goes **LOW** in command mode due to system boot event (**Note**: Will remain **HIGH** if CYSPP pin is asserted) ***Notes:** This pin is only effective as documented after the system has booted as normal application. When this pin takes effective, the FACTORY_RT pin will be reused and out of work.	**Yes**

Pin Name	Direction	Details	Optional
FACTORY_TR	Input	**Description:** Factory test or reset control. The external host can use this pin to boot into a manufacturing test mode (CYSPP pin HIGH), or to trigger a complete reset of all settings back to their factory default values (CYSPP pin LOW), similar to what happens when the "**system_factory_reset**" API command is used ("**/SFAC**" in text mode). If this pin and the CYSPP pin are used in this way to trigger a factory reset, the firmware will only reboot once and at least one of the pins is de-asserted. This is required to avoid an endless factory reset loop. To cause either of these operations, the FACTORY_TR pin must be asserted at the time the module boots. After the boot process is complete, the pin's logic state has no special impact on behavior. Due to this pin's purpose, it is not possible to disable this functionality in software. **Control signal logic (active-low):** • **LOW** – depends on CYSPP o **CYSPP LOW** – reset everything to factory defaults o **CYSPP HIGH** – enter manufacturing test mode • **HIGH** – firmware will boot normally **Default boot state:** • High-impedance input after briefly pulled **HIGH** during boot ***Notes:** This pin is only effective as documented above at boot time. Once the normal boot process finishes, it will be reused as DATA_READY pin.	**See notes***
LP_MODE	Input	**Description:** Low-power status control. The external host can use this pin to affect the sleep behavior of the module, specifically by either preventing or allowing entry into Sleep modes. **Control signal logic (active-low):** • **LOW** – CPU is kept in Active mode. • **HIGH** – CPU is allowed (but not forced) to Sleep. **Default boot state:** • Internally pulled **HIGH** (Sleep allowed)	No
LP_STATUS	Output	**Description:** Low-power status indicator. The external host can use this pin to understand the power state of the module. This is especially useful if the external MCU needs to know whether the module can communicate over UART (UART is disabled in Deep Sleep and Hibernate power states). **Status indicator logic (Active-low):** • **LOW** – CPU is in the Active state. • **HIGH** – CPU is in Deep Sleep or Hibernation mode. **Default boot state:** • **LOW** (awake) until boot process finishes, then **HIGH** unless **LP_MODE** is asserted.	**Yes**

Table 8-2. GPIO Pin Functionality Detail

8.2.2 PWM Output Pins

EZ-Serial provides four dedicated PWM output pins (PWM0, PWM1, PWM2, and PWM3). You can enable PWM output on any of the four PWM channels using the gpio_set_pwm_mode (SPWM, ID=9/11) API command. PWM channels are controlled via independent 24 MHz clocks, and each can use separate divider, prescaler, period, and compare settings for complete flexibility.

Enabling PWM on each channel means you cannot use that pin for other generic I/O. To return a PWM channel pin to standard functionality, use the gpio_set_pwm_mode (SPWM, ID=9/11) API command to disable PWM output on that pin.

Note: Enabling PWM output on one or more channels will automatically prevent the CPU from entering Deep Sleep under any circumstances. This happens because the high-frequency clock required to generate the PWM signal cannot operate while the CPU is in Deep Sleep. To allow Deep Sleep mode again, you must disable all PWM output. See Managing Sleep States for more details.

8.2.3 Analog Input Pins (ADC)

EZ-Serial provides a single dedicated ADC input pin (ADC0) for reading analog voltages. The ADC supports an input voltage range of 0 V minimum to 1.024 V maximum. To perform a single ADC conversion, use the gpio_query_adc (/QADC, ID=9/2) API command. Once the conversion completes, the module will transmit the result in the response to this command.

You can use the ADC0 pin as a normal digital GPIO, but using the gpio_query_adc (/QADC, ID=9/2) API command will reconfigure the pin back to a high-impedance analog input state.

8.3 Functional Capabilities

It is important to understand the intended use case for certain GPIO-related functions provided by the EZ-Serial firmware, especially digital interrupt detection and analog-to-digital conversion (ADC). This ensures that your expectations are met.

8.3.1 Digital Interrupt Detection

The internal chipset can detect and respond to interrupts extremely quickly. However, EZ-Serial generates an API event packet for each monitored edge change. These events are queued when they occur and transmitted out to the host as API event packets. To avoid overflowing the limited outgoing API packet queue, events which cannot fit into the queue are simply discarded. This means that if edge changes occur faster than API event packet transmissions can keep up, some interrupts will not be reported.

If your application specifically requires very fast interrupt detection, it may be necessary to develop a custom firmware application using the WICED Studio or ModusToolbox IDE.

8.3.2 Analog-to-Digital Conversion

Similar to Digital Interrupt Detection, the ADC operates very quickly but incurs significant processing overhead to transmit conversion results to an external host via API event packets. The EZ-Serial firmware platform provides a way to perform on-demand single ADC reads on individual analog channels, such as what might be involved in periodic battery voltage measurements or analog light, gas, or temperature sensor readings.

If your application requires rapid single- or multi-channel sequencing and data analysis, it may be necessary to use the WICED Studio or ModusToolbox IDE to create a custom firmware implementation.

9 EZ-Serial GAP Profile Reference

The EZ-Serial platform makes use of a few custom GATT profiles. The service UUIDs, characteristic UUIDs, special permissions, and overall structure are outlined here for quick reference. You can find detailed reference material on the Community.

9.1 WICED Over-the-Air (OTA) Upgrade Profile

The WICED OTA Upgrade Profile is used to transmit the OTA commands from a device that implements a device that exposes the WICED OTA Upgrade Service. It is also responsible for receiving the command responses coming from the WICED OTA Upgrade Service Device via notifications. The WICED OTA Upgrade Profile used in the EZ-Serial is non-secure.

The profile contains a single service ("WICED OTA Upgrade Service"), which contains a Control Point characteristic ("Command") and Data characteristic ("Data"). The Control Point characteristic shall also contain a standard Client Characteristic Configuration descriptor with mandatory properties. The structural outline of this profile is as follows:

- **OTA Upgrade** Service: UUID **1F38A138AD82-3586-A043-135C-471E5DAE**

  The WICED OTA Upgrade Service allows a Bootloader Component to update the existing firmware on the Cypress BLE device using the BLE interface as a communication interface. The Bootloader Service does not execute any bootloader commands, but it is designed to pass commands to the Bootloader component and send the response from the Bootloader Component to the Client.

  - **Control Point** Characteristic: UUID **1B666C080A57-8E83-994E-A7F7-BF50DDA3**
    (Write, Indicate, Notify)

    The Control Point Characteristic is used to receive the OTA Upgrade commands from the Client to the Server via the Write requests and send the response from the Server via Indication/notifications. The characteristic has a variable length and should be set for its maximum length of 512 bytes.

    - Configuration Descriptor: UUID **0x2902**

  - **Data** Characteristic: UUID **26FE2EE70924-4FB7-9140-61D9-7A6CE8A2**
    (Write)

    The Data Characteristic is used to receive the image chunk data from the Client to the Server via Write requests. The characteristic has a variable length and should be set for its maximum length of 512 bytes.

You can find additional information in the following documents:

- 002-19289 – WICED Firmware Upgrade Library

9.2 CYSPP Profile

The Cypress Serial Port Profile (CYSPP) provides bi-directional serial data transfer between two remote devices, each of which passes data through a single local hardware serial interface. It supports both acknowledged transfers and unacknowledged transfers, and provides a mechanism for virtual flow control in both the RX and TX direction.

The profile contains a single service ("CYSPP"), which contains three characteristics for data transfer and flow control ("Acknowledged Data", "Unacknowledged Data", and "RX Flow"). The structural outline of this profile is as follows:

- CYSPP Service: UUID 65333333-A115-11E2-9E9A-0800200CA100

  - **Acknowledged Data** Characteristic: UUID **65333333-A115-11E2-9E9A-0800200CA101**
    (Write, Indicate)

    The Acknowledged Data Characteristic is used to send and receive data in an acknowledged fashion. The EZ-Serial firmware can fully track every transfer in both directions. This characteristic has a variable length, supporting transfers in each direction of up to 20 bytes per packet.

    - Configuration Descriptor: UUID **0x2902**

- ☐ **Unacknowledged Data** Characteristic: (Write without response, Notify) UUID **65333333-A115-11E2-9E9A-0800200CA102**

  The Unacknowledged Data Characteristic is used to send and receive data in an unacknowledged fashion. The EZ-Serial firmware cannot track transfers using this mode once they have been accepted by the BLE stack. This provides less control, but the lack of acknowledgements also allows for much greater maximum throughput. This characteristic has a variable length, supporting transfers in each direction of up to 20 bytes per packet.

  - o Configuration Descriptor: UUID **0x2902**

- ☐ **RX Flow** Characteristic: (Indicate) UUID **65333333-A115-11E2-9E9A-0800200CA103**

  The RX Flow Characteristic is used to indicate to the client that the server can no longer safely receive new data. If the client subscribes to indications from this characteristic, the server will assume that the client will obey flow control signals. This characteristic is one byte in length. An indicated value of “0” means that it is safe for the client to send data, while a value of “1” means that the client must refrain from sending data.

  - o Configuration Descriptor: UUID **0x2902**

9.3 CYCommand Profile

The Cypress Command Profile (CYCommand) provides remote access to EZ-Serial’s API protocol. With this profile, you can send and receive API commands, responses, and events from a remote device without having wired access to the target module.

The profile contains a single service (“CYCommand”), which contains two characteristics for data transfer and optional challenge-response security. The structural outline of this profile is as follows:

- ■ CYCommand Service: UUID 65333333-A115-11E2-9E9A-0800200CA200
  - ☐ **Challenge** Characteristic: (Write, Indicate) UUID **65333333-A115-11E2-9E9A-0800200CA201**

    The Challenge Characteristic is used for simple application-level authentication which is optionally required before the remote device may use the Data characteristic (GATT-API bridge). The client sends data to the server using acknowledged writes, and the server sends data to the client using indications.

      - ▪ Configuration Descriptor: UUID **0x2902**

  - ☐ **Data** Characteristic: (Write, Indicate) UUID **65333333-A115-11E2-9E9A-0800200CA202**

    The Data Characteristic is used to send and receive API protocol data. This characteristic has a variable length, supporting transfers in each direction of up to 20 bytes per packet. The client sends command data to the server using acknowledged writes, and the server sends response and event data to the client using indications.

    - o Configuration Descriptor: UUID **0x2902**

10 Configuration Example References

The configuration examples provided in this section are each designed to work independently, assuming in each case that the platform is initially configured using factory default settings. Applying the commands in one example and then immediately following this with the commands from another example may result in changes to the first set of behavior that is no longer in line with the expected results.

You can return a module to factory defaults as a baseline configuration at any time by using the system_factory_reset (/RFAC, ID=2/5) API command. This reset command is not explicitly included in any of the configuration snippets within this section.

10.1 Factory Default Settings

While you can return to the factory default settings on the module by performing a factory reset, it is also helpful to know what those settings are for comparison or to explicitly change one or more individual settings back to the default value without reverting all customizations at once. The following is a comprehensive list of commands that will return the EZ-Serial module to default behavior:

```
SPPM,M=00
SPEM,M=01
SSLP,L=01
STXP,P=07          (lower values on some modules for regulatory compliance)
ST,I=01
STU,B=0001C200,A=00,C=00,F=00,D=08,P=00,S=01
SDN,N=EZ-Serial %M4:%M5:%M6
SDA,A=0000
SAD,D=
SSRD,D=
SAP,M=02,T=00,I=0030,C=07,L=00,O=0000,F=00
SSP,M=02,I=0100,W=0100,A=00,F=00,D=00,O=0000
SCP,I=0006,L=0000,O=0064,V=0100,W=0100,M=0000
SGSP,F=01
SGCP,F=01
SPRV,M=00,I=012C
SSBP,M=11,B=01,K=10,P=00,I=03,F=01
.CYSPPSP,E=02,G=00,C=0131,L=00000000,R=00000000,M=00000000,P=02,S=00,F=02
.CYSPPSH,A=0000,B=0000,C=0000,D=0000
.CYSPPSK,M=01,W=05,L=14,E=0D
.CYCOMSP,E=01,H=03,T=0000,F=00,C=00,S=00,R=
.IBSP,E=00,I=00A0,C=0131,J=0001,N=0001,U=E2C56DB5DFFB48D2B060D0F5A71096E0
.EDDYSP,E=00,I=00A0,T=10,D=006379707265737300
```

Remember that the above commands affect only RAM. To make them permanent, apply all settings to flash using the system_store_config (/SCFG, ID=2/4) API command.

10.2 Adopted Bluetooth SIG GATT Profile Structure Snippets

The snippets in this section demonstrate how to add various GATT service and characteristic structural elements to support official profiles defined by the Bluetooth SIG, and some other common services.

Note:

- These database structures concern only the **GATT server** side of the profiles in question. GATT client operations depend on the client device.

- The information provided in this section only covers the basic GATT structure, but does not include any specific values which may be necessary or helpful for specific functionality. Many characteristics also have flexible **length** values which depend on application design, such as those inside the Device Information Service (0x180A) or Human Interface Device Service (0x1812). See the official Bluetooth SIG documentation or other related resources linked under each service for more details.

- Additions to and removals from the GATT structure are always stored in flash. If the “result” value in the response indicates success, the change will be effective immediately and will persist through power cycles and resets. The internal CPU is occupied for approximately 15 ms during each flash write operation, and during this time no other activity will be processed (UART or BLE communication). Any UART data sent during this brief window will be lost. Therefore, you should only modify the GATT structure while disconnected, and you should allow a gap of at least 20 ms between the end of one API command and the beginning of a new command. If you have enabled hardware flow control using the system_set_uart_parameters (STU, ID=2/25) API command, EZ-Serial will block incoming data flow during flash writes to prevent serial data corruption or loss.

10.2.1 Generic Access Service (0x1800)

You can find the official documentation for this service on the Bluetooth SIG website.

Note: This service is included in the EZ-Serial application. It is always present in the fixed, non-removable part of the GATT structure. Do not add another instance of this service to the EZ-Serial application.

EZ-Serial assumes that the attribute handle is starting from 1. Data item of characteristic attribute include the attribute handle (0x0003, 0x0005, 0x0007, and 0x0009 respectively in this example) which correspond to the characteristic value attribute.

```
/CAC,T=0,P=02,L=04,D=00280018
/CAC,T=0,P=02,L=07,D=032802 0300002A
/CAC,T=1,P=0B,L=40,D=
/CAC,T=0,P=02,L=07,D=0328020500012A
/CAC,T=1,P=02,L=02,D=
/CAC,T=0,P=02,L=07,D=0328020700042A
/CAC,T=1,P=02,L=08,D=
/CAC,T=0,P=02,L=07,D=0328020900A62A
/CAC,T=1,P=02,L=01,D=
```

10.2.2 Generic Attribute Service (0x1801)

You can find the official documentation for this service on the Bluetooth SIG website.

Note: This service is included in the EZ-Serial application. It is always present in the fixed, non-removable part of the GATT structure. Do not add another instance of this service to the EZ-Serial application.

EZ-Serial assumes that the attribute is handled starting from 1. Attribute handle (0x0003) corresponds to the value attribute.

```
/CAC,T=0,P=02,L=04,D=00280018
/CAC,T=0,P=02,L=07,D=0328200300052A
/CAC,T=1,P=02,L=04,D=
/CAC,T=0,P=0A,L=04,D=0229
```

10.2.3 Immediate Alert Service (0x1802)

You can find the official documentation for this service on the Bluetooth SIG website.

```
/CAC,T=0,P=02,L=04,D=00280218
/CAC,T=0,P=02,L=07,D=032804180062A
/CAC,T=1,P=0A,L=01,D=
```

10.2.4 Link Loss Service (0x1803)

You can find the official documentation for this service on the Bluetooth SIG website.

```
/CAC,T=0,P=02,L=04,D=00280318
/CAC,T=0,P=02,L=07,D=03280A1800062A
/CAC,T=1,P=0A,L=01,D=
```

10.2.5 TX Power Service (0x1804)

You can find the official documentation for this service on the Bluetooth SIG website.

```
/CAC,T=0,P=02,L=04,D=00280418
/CAC,T=0,P=02,L=07,D=0328021800072A
/CAC,T=1,P=02,L=01,D=
/CAC,T=0,P=0A,L=04,D=0229
```

10.2.6 Current Time Service (0x1805)

You can find the official documentation for this service on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0518
/CAC,T=2803,R=01,W=00,C=12,L=0000,D=2B2A
/CAC,T=0000,R=01,W=00,C=12,L=000A,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=0F2A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=142A
/CAC,T=0000,R=01,W=00,C=02,L=0004,D=
```

10.2.7 Reference Time Update Service (0x1806)

You can find the official documentation for this service on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0618
/CAC,T=2803,R=01,W=00,C=04,L=0000,D=162A
/CAC,T=0000,R=02,W=02,C=04,L=0001,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=172A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
```

10.2.8 Next DST Change Service (0x1807)

You can find the official documentation for this service on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0718
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=112A
/CAC,T=0000,R=01,W=00,C=02,L=0008,D=
```

10.2.9 Glucose Service (0x1808)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0818
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=182A
/CAC,T=0000,R=00,W=00,C=10,L=000A,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=342A
/CAC,T=0000,R=00,W=00,C=10,L=0003,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=512A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=28,L=0000,D=522A
/CAC,T=0000,R=02,W=02,C=28,L=0003,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.10 Health Thermometer Service (0x1809)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0918
/CAC,T=2803,R=01,W=00,C=20,L=0000,D=1C2A
/CAC,T=0000,R=00,W=00,C=20,L=0005,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=1D2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=1E2A
/CAC,T=0000,R=00,W=00,C=10,L=0005,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=212A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
```

10.2.11 Device Information Service (0x180A)

In the following commands, most identification data attributes are given 16-byte lengths (`L=0010`). You will most likely need to modify these lengths according to the data you intend to write into the characteristics.

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0A18
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=292A
/CAC,T=0000,R=01,W=00,C=02,L=0010,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=242A
/CAC,T=0000,R=01,W=00,C=02,L=0010,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=252A
/CAC,T=0000,R=01,W=00,C=02,L=0010,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=272A
/CAC,T=0000,R=01,W=00,C=02,L=0010,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=262A
/CAC,T=0000,R=01,W=00,C=02,L=0010,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=282A
/CAC,T=0000,R=01,W=00,C=02,L=0010,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=232A
/CAC,T=0000,R=01,W=00,C=02,L=0008,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=2A2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=502A
/CAC,T=0000,R=01,W=00,C=02,L=0007,D=
```

10.2.12 Heart Rate Service (0x180D)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0D18
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=372A
/CAC,T=0000,R=00,W=00,C=10,L=0002,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=382A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2803,R=01,W=00,C=08,L=0000,D=392A
/CAC,T=0000,R=02,W=02,C=08,L=0001,D=
```

10.2.13 Phone Alert Status Service (0x180E)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0E18
/CAC,T=2803,R=01,W=00,C=12,L=0000,D=3F2A
/CAC,T=0000,R=01,W=00,C=12,L=0001,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=12,L=0000,D=412A
/CAC,T=0000,R=01,W=00,C=12,L=0001,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=04,L=0000,D=402A
/CAC,T=0000,R=02,W=02,C=04,L=0001,D=
```

10.2.14 Battery Service (0x180F)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0F18
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=192A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2904,R=01,W=00,C=02,L=0007,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.15 Blood Pressure Service (0x1810)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1018
/CAC,T=2803,R=01,W=00,C=20,L=0000,D=352A
/CAC,T=0000,R=00,W=00,C=20,L=0007,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=362A
/CAC,T=0000,R=00,W=00,C=10,L=0007,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=492A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
```

10.2.16 Alert Notification Service (0x1811)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1118
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=472A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=462A
/CAC,T=0000,R=00,W=00,C=10,L=0002,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=482A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=452A
/CAC,T=0000,R=00,W=00,C=10,L=0002,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=08,L=0000,D=442A
/CAC,T=0000,R=02,W=02,C=08,L=0002,D=
```

10.2.17 Human Interface Device Service (0x1812)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1218
/CAC,T=2803,R=01,W=00,C=06,L=0000,D=4E2A
/CAC,T=0000,R=01,W=01,C=06,L=0001,D=
/CAC,T=2803,R=01,W=00,C=12,L=0000,D=4D2A
/CAC,T=0000,R=01,W=00,C=12,L=0000,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2908,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=4B2A
/CAC,T=0000,R=01,W=00,C=02,L=0000,D=
/CAC,T=2907,R=01,W=00,C=02,L=0000,D=
/CAC,T=2803,R=01,W=00,C=12,L=0000,D=222A
/CAC,T=0000,R=01,W=00,C=12,L=0008,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=0E,L=0000,D=322A
/CAC,T=0000,R=01,W=01,C=0E,L=0008,D=
/CAC,T=2803,R=01,W=00,C=12,L=0000,D=332A
/CAC,T=0000,R=01,W=00,C=12,L=0003,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=4A2A
/CAC,T=0000,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=04,L=0000,D=4C2A
/CAC,T=0000,R=02,W=02,C=04,L=0001,D=
```

10.2.18 Scan Parameters Service (0x1813)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1318
/CAC,T=2803,R=01,W=00,C=04,L=0000,D=4F2A
/CAC,T=0000,R=02,W=02,C=04,L=0004,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=312A
/CAC,T=0000,R=00,W=00,C=10,L=0001,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.19 Running Speed and Cadence Service (0x1814)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1418
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=532A
/CAC,T=0000,R=00,W=00,C=10,L=0004,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=542A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=5D2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2803,R=01,W=00,C=28,L=0000,D=552A
/CAC,T=0000,R=02,W=02,C=28,L=0006,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.20 Cycling Speed and Cadence Service (0x1816)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1618
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=5B2A
/CAC,T=0000,R=00,W=00,C=10,L=0001,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=5C2A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=5D2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2803,R=01,W=00,C=28,L=0000,D=552A
/CAC,T=0000,R=02,W=02,C=28,L=0006,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.21 Cycling Power Service (0x1818)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1818
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=632A
/CAC,T=0000,R=00,W=00,C=10,L=0004,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2903,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=652A
/CAC,T=0000,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=5D2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=642A
/CAC,T=0000,R=00,W=00,C=10,L=0001,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=28,L=0000,D=662A
/CAC,T=0000,R=02,W=02,C=28,L=0005,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.22 Location and Navigation Service (0x1819)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1918
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=6A2A
/CAC,T=0000,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=672A
/CAC,T=0000,R=00,W=00,C=10,L=0002,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=692A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=28,L=0000,D=6B2A
/CAC,T=0000,R=02,W=02,C=28,L=0005,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=682A
/CAC,T=0000,R=00,W=00,C=10,L=0006,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.23 Body Composition Service (0x181B)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1B18
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=9B2A
/CAC,T=0000,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=20,L=0000,D=9C2A
/CAC,T=0000,R=00,W=00,C=20,L=002A,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.24 User Data Service (0x181C)

You will need to modify the lengths of the first three characteristics according to the data you intend to use with them. Also, the reference code lists 65 attribute definitions, but your application may not need to use all definitions. See the official specification for this service on the Bluetooth SIG website for details.

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1C18
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=8A2A
/CAC,T=0000,R=01,W=01,C=0A,L=0000,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=902A
/CAC,T=0000,R=01,W=01,C=0A,L=0000,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=872A
/CAC,T=0000,R=01,W=01,C=0A,L=0000,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=802A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=852A
/CAC,T=0000,R=01,W=01,C=0A,L=0004,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=8C2A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=982A
/CAC,T=0000,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=8E2A
/CAC,T=0000,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=962A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=8D2A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=922A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=912A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=7F2A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=832A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=932A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=862A
/CAC,T=0000,R=01,W=01,C=0A,L=0004,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=972A
/CAC,T=0000,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=8F2A
/CAC,T=0000,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=882A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=892A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=7E2A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=842A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=812A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=822A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=8B2A
/CAC,T=0000,R=01,W=01,C=0A,L=0004,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=942A
/CAC,T=0000,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=952A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=992A
/CAC,T=0000,R=01,W=01,C=0A,L=0004,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=9A2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2803,R=01,W=00,C=28,L=0000,D=9F2A
/CAC,T=0000,R=02,W=02,C=28,L=0002,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=A22A
/CAC,T=0000,R=01,W=01,C=0A,L=0000,D=
```

10.2.25 Weight Scale Service (0x181D)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1D18
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=9E2A
/CAC,T=0000,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=20,L=0000,D=9D2A
/CAC,T=0000,R=00,W=00,C=20,L=0013,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.26 Bond Management Service (0x181E)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1E18
/CAC,T=2803,R=01,W=00,C=08,L=0000,D=A42A
/CAC,T=0000,R=02,W=02,C=08,L=0001,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=A52A
/CAC,T=0000,R=01,W=00,C=02,L=0003,D=
```

10.2.27 Continuous Glucose Monitoring Service (0x181F)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1F18
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=A72A
/CAC,T=0000,R=00,W=00,C=10,L=0006,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=A82A
/CAC,T=0000,R=01,W=00,C=02,L=0006,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=A92A
/CAC,T=0000,R=01,W=00,C=02,L=0005,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=AA2A
/CAC,T=0000,R=01,W=01,C=0A,L=0009,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=AB2A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=28,L=0000,D=522A
/CAC,T=0000,R=02,W=02,C=28,L=0003,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=28,L=0000,D=AC2A
/CAC,T=0000,R=02,W=02,C=28,L=000F,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

10.2.28 Environmental Sensing Service (0x181A)

The complete implementation of every supported sensor data characteristic within this service will not fit within EZ-Serial's dynamic GATT implementation due to flash limits. The reference code lists 124 attribute definitions, but only 102 can fit (38 on devices with 128K of flash memory) as described in Defining Custom Local GATT Services and Characteristics). Therefore, you must choose a subset of the functionality listed here according to the sensors that your application requires.

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=1A18
/CAC,T=2803,R=01,W=00,C=20,L=0000,D=7D2A
/CAC,T=0000,R=00,W=00,C=20,L=0002,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=732A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=722A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=7B2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=6C2A
/CAC,T=0000,R=01,W=00,C=02,L=0003,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0006,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=742A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=7A2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=6F2A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=772A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=752A
/CAC,T=0000,R=01,W=00,C=02,L=0003,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0006,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=782A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=6D2A
/CAC,T=0000,R=01,W=00,C=02,L=0004,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0008,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=6E2A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=712A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=702A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=762A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=792A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=A32A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=2C2A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=A02A
/CAC,T=0000,R=01,W=00,C=02,L=0004,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=A12A
/CAC,T=0000,R=01,W=00,C=02,L=0006,D=
/CAC,T=290C,R=01,W=00,C=02,L=000B,D=
/CAC,T=290D,R=01,W=00,C=02,L=0002,D=
/CAC,T=2901,R=01,W=00,C=02,L=0000,D=
/CAC,T=2906,R=01,W=00,C=02,L=0004,D=
```

10.2.29 HTTP Proxy Service (0x1823)

You can find the official documentation for this service can be found on the Bluetooth SIG website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=2318
/CAC,T=2803,R=01,W=00,C=08,L=0000,D=B62A
/CAC,T=0000,R=02,W=02,C=08,L=0000,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=B72A
/CAC,T=0000,R=01,W=01,C=0A,L=0000,D=
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=B92A
/CAC,T=0000,R=01,W=01,C=0A,L=0000,D=
/CAC,T=2803,R=01,W=00,C=08,L=0000,D=BA2A
/CAC,T=0000,R=02,W=02,C=08,L=0001,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=B82A
/CAC,T=0000,R=00,W=00,C=10,L=0003,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=BB2A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
```

10.2.30 Apple Notification Center Service (7905F431-B5CE-4E99-A40F-4B1E122D00D0)

You can find the official documentation for this service can be found on the Apple Developer Website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=D0002D121E4B0FA4994ECEB531F40579
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=BD1DA299E625588CD94201630D12BF9F
/CAC,T=0000,R=00,W=00,C=10,L=0008,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/CAC,T=2803,R=01,W=00,C=08,L=0000,D=D9D9AAFDBD9B2198A849E145F3D8D169
/CAC,T=0000,R=02,W=02,C=08,L=0006,D=
/CAC,T=2803,R=01,W=00,C=10,L=0000,D=FB7B7CCE6AB344BEB54BD624E9C6EA22
/CAC,T=0000,R=00,W=00,C=10,L=0000,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
```

Document Revision History

Document Title: EZ-Serial WICED Firmware Platform User Guide for EZ-BT Modules

Document Number: 002-30728

Revision	ECN	Issue Date	Description of Change
**	6959751	09/03/2020	Initial release

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

WICED® Solutions

CYW20706 | CYW20719 | CYW20735 | CYW20721 | CYW20819

Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training| Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709