

EZ-Serial BLE Firmware Platform User Guide

Doc. No. 002-11259 Rev. *E

Infineon Technologies AG198 Champion Court
San Jose, CA 95134-1709
www. infineon.com

© Infineon Technologies AG, 1999-2023. This document is the property of Infineon Technologies AG and its subsidiaries, including Cypress Semiconductor Corporation and Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Infineon under the intellectual property laws and treaties of the United States and other countries worldwide. Infineon reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Infineon hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Infineon hardware product units, and (2) under those claims of Infineon’s patents that are infringed by the Software (as provided by Infineon, unmodified) to make, use, distribute, and import the Software solely for use with Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Infineon reserves the right to make changes to this document without further notice. Infineon does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Infineon products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Infineon is not liable, in whole or in part, and you shall and hereby do release Infineon from any claim, damage, or other liability arising from or related to all Unintended Uses of Infineon products. You shall indemnify and hold Infineon harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Infineon products.

Infineon, the Infineon logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Infineon in the United States and other countries. For a more complete list of Infineon trademarks, visit Infineon.com. Other names and brands may be claimed as property of their respective owners.

Contents

1. Introduction 5
   - 1.1 How to Use This Guide 5
   - 1.2 Block Diagram 6
   - 1.3 Functional Overview 7
   - 1.4 Infineon BLE Device Support 8
2. Getting Started 9
   - 2.1 Prerequisites 9
   - 2.2 Factory Default Behavior 9
   - 2.3 Connecting a Host Device 10
   - 2.4 Communicating with a Host Device 13
   - 2.5 Configuration Settings, Storage, and Protection 25
   - 2.6 Where to Find Related Material 27
3. Operational Examples 28
   - 3.1 System Setup Examples 28
   - 3.2 Cable Replacement Examples with CYSPP 37
   - 3.3 Remote Control Examples with CYCommand 41
   - 3.4 GAP Peripheral Examples 43
   - 3.5 GAP Central Examples 47
   - 3.6 GATT Server Examples 50
   - 3.7 GATT Client Examples 57
   - 3.8 Security and Encryption Examples 60
   - 3.9 Beacon Examples 66
   - 3.10 Performance Testing Examples 67
   - 3.11 Device Firmware Update Examples 71
4. Application Design Examples 73
   - 4.1 Smart MCU Host with 4-Wire UART and Full GPIO Connections 73
   - 4.2 Dumb Terminal Host with CYSPP and Simple GPIO State Indication 73
   - 4.3 Module-Only Application with Beacon Functionality 75
5. Host API Library 76
   - 5.1 Host API Library Overview 76
   - 5.2 Implementing a Project Using the Host API Library 77

5.3 Porting the Host API Library to Different Platforms 80
5.4 Using the API Definition JSON File to Create a Custom Library 81
6. Troubleshooting 82
6.1 UART Communication Issues 82
6.2 BLE Connection Issues 83
6.3 GPIO Signal Issues 84
7. API Protocol Reference 85
7.1 Protocol Structure and Communication Flow 85
7.2 API Commands and Responses 89
7.3 API Events 205
7.4 Error Codes 234
7.5 Macro Definitions 238
8. GPIO Reference 239
8.1 GPIO Pin Map for Supported Modules 239
8.2 GPIO Pin Functionality 241
8.3 Functional Capabilities 242
9. Infineon GATT Profile Reference 243
9.1 Bootloader Profile 243
9.2 CYSPP Profile 243
9.3 CYCommand Profile 244
10. Configuration Example Reference 245
10.1 Factory Default Settings 245
10.2 Adopted Bluetooth SIG GATT Profile Structure Snippets 246
Revision History 258

1. Introduction

This document provides a complete guide to the EZ-Serial platform on EZ-BLE modules. The guide covers the following:

- Infineon 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 equal or more than 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 all of these behaviors from an external host

1.1 How to Use This Guide

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

- System description and functional overview (Chapter 1, Introduction and Chapter 2, Getting Started)
- Firmware configuration examples (Chapter 3, Operational Examples)
- Complete design examples (Chapter 4, Application Design Examples)
- API protocol implementations for external MCU (Chapter 5, Host API Library)
- Troubleshooting guides (Chapter 6, Troubleshooting)
- Reference material (Chapter 7, API Protocol Reference through 10, Configuration Example Reference)

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

Read through Chapter 1 (Introduction) and Chapter 2 (Getting Started) for a functional overview.

Find at least one example from Chapter 3 (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:

- How to Get Started in CYSPP Mode with Zero Custom Configuration
- How to Define Custom Local GATT Services and Characteristics
- How to Detect and Process Written Data from a Remote Client
- How to Bond With or Without MITM Protection
- How to Configure iBeacon Transmissions
- How to Update Firmware Using the DFU Bootloader

Find at least one design example from Chapter 4 (Application Design Examples) that is similar to the type of system you intend to use an EZ-Serial-based EZ-BLE module 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 Chapter 5 (Host API Library) to understand how the external MCU will need to communicate with the module.

Spend a few minutes reading through the guides in Chapter 6 (Troubleshooting) to avoid unnecessary frustration later on in the event that something doesn't behave in the way you expect.

Note the reference material available in this document to allow fast access to additional information and resources available from Infineon. 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:

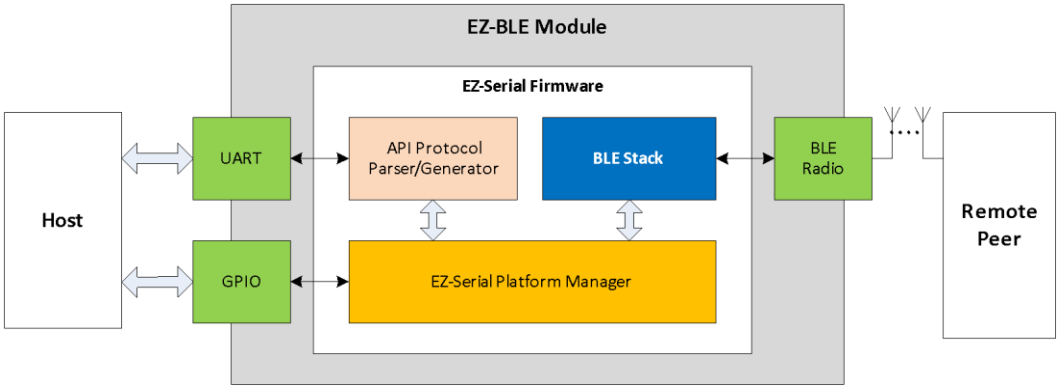
- Proper descriptive name (e.g. "**gpio_set_direction**"), unique among all other methods.
- Text-mode name (e.g. "**SIOD**"), applicable when using the API protocol in text mode (see Section 2.4.1, Using the API Protocol in Text Mode).
- Group/method ID values (e.g. "**9/5**"), present in the 4-byte header when using the API protocol in binary mode (see Section 2.4.2, Using the API Protocol in Binary Mode).

Click on any linked API method for detailed reference material in Chapter 7 (API Protocol Reference).

1.2 Block Diagram

The EZ-Serial platform is built on top of EZ-BLE modules from Infineon. 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. EZ-BLE modules communicate with a remote device using the Bluetooth Low Energy (BLE) 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 BLE-based applications. To accomplish this, the firmware implements an intuitive API protocol over the UART interface and exposes a number of 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 (requires a device with equal or more than 256K flash memory)
- Encryption, bonding, and protection from man-in-the-middle (MITM) threats
- CYSPP mode for bidirectional serial data transmission
- UART and over-the-air (OTA) bootloader for firmware updates (requires a device with 256K flash memory)
- iBeacon and Eddystone beaconing
- Remote firmware configuration
- Efficient low-power operation

1.3.2 Hardware and Communication Features

The EZ-Serial platform also implements a number of 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
- Access to internal AES encryption and decryption engine
- Access to internal pseudo-random number generator
- UART wake-on-RX support

1.3.3 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 PSoC Creator 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 EZ-Serial later, simply download the latest image from the Infineon website and flash it using the same mechanism.

For details on where to find these images, see Section 2.6.1 (Latest EZ-Serial Firmware Image).

1.4 Infineon BLE Device Support

As of the current release (v1.1.1 build 26), EZ-Serial firmware images exist for the following devices:

Table 1-1. Supported Devices

Devices with 128k flash memory	Devices with 256k flash memory
CYBLE-01201X-X0 CYBLE-014008-00 CYBLE-022001-00	CYBLE-2X20XX-X1 CYBLE-2120XX-X0 CYBLE-212020-01 CYBLE-214009-00 CYBLE-214015-01 CYBLE-222005-00 CYBLE-222014-01 CYBLE-224110-00 CYBLE-224116-01

While all images are based on the same design, those compiled for devices with 128K flash memory have a few limitations due to the reduced flash and SRAM available on those platforms. Here is a complete list of limitations on 128K parts:

- **128K modules do not support DFU for either UART or over-the-air (OTA) image updates**
- 128K modules do not support direct L2CAP connectivity
- 128K modules support a GATT MTU of 384 bytes instead of 512 bytes
- 128K modules support half as many dynamic GATT entries (see Table 3-9 in Section 3.6.1, How to Define Custom Local GATT Services and Characteristics)

All other internal features are identical on all platforms.

Because many devices have unique footprints, the physical pin assignment for functional pins such as **DATA_READY** or **LP_MODE** vary between devices. For details on which pins support which functions, see Table 8-1 in Section 8.1 (GPIO Pin Map for Supported Modules).

2. Getting Started

EZ-Serial allows for rapid integration of BLE 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, we recommend that you have the following parts available:

- CY8CKIT-042-BLE-A Bluetooth® Low Energy 4.2 Compliant Pioneer Kit
- CYBLE-212019-00 EZ-BLE PRoC Module Evaluation Board or other EZ-Serial-compatible modules
- Computer with serial terminal software such as Tera Term, Realterm, or PuTTY
- *Optional:* CYUSBS232 USB-UART LP Reference Design Kit for maximizing throughput with flow control
    - *Optional:* BLE-capable mobile device such as an iPad, iPhone, or Android phone or tablet

The BLE Pioneer Kit contains an evaluation board with a USB-to-UART bridge built in, as well as the CySmart BLE dongle that you can use with the matching CySmart software for various client-side functions such as connection establishment, GATT exploration, and firmware updates.

> NOTE: The BLE Pioneer Kit's internal USB-to-UART bridge does not support flow control and may exhibit some data loss at very high throughput. It also does not support baud rates above 115200. For fast throughput tests, you should connect an external adapter that supports flow control and higher baud rates, such as the CYUSBS232 kit.

You can control EZ-Serial over a UART interface without additional GPIOs; refer to Chapter 4 (Application Design Examples) for detail. However, we recommend using the BLE Pioneer kit for the best experience learning and prototyping due to its comprehensive design and peripheral support.

2.2 Factory Default Behavior

The default configuration of EZ-Serial firmware is shown below:

- UART interface configured for 115200 baud, 8 data bits, no parity, 1 stop bit
- UART flow control disabled (signals from the module are not generated, signals from the host are ignored)
- 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 **enabled** in pull up/down mode (not strong drive)

When the module is powered on or reset, it will generate the system_boot (BOOT, ID=2/1) API event. This is only one example of one API method used by the platform; refer to Chapter 7 (API Protocol Reference) for details on the structure and behavior of the API protocol.

The boot event will appear similar to this, if the protocol generator is in the default text mode:

```
@E,003B,BOOT,E=0101011A,S=030300FA,P=0103,H=05,C=01,A=00A050421A63
```

This text-mode string of data indicates:

- `@E` – an event has occurred
- `003B` – there are 59 bytes (0x3B) of content to follow
- `BOOT` – the event which occurred is the **BOOT** event
- `E=0101011A` – the EZ-Serial application version is 1.1.1 build 26 (0x1A)
- `S=03030035` – the BLE stack component version is 3.3.0 build 53 (0x35)
- `P=0103` – the protocol version is 1.3
- `H=05` – the hardware platform is CYBLE-2120XX-X0 (e.g. CYBLE-212019-00)
- `C=01` – the cause for this boot/reset is standard power-cycle or XRES hardware signal
  - `A=00A050421A63` – the public Bluetooth MAC address of this module is 00:A0:50:42:1A:63

> **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 the 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 gap_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 else keep the module in the hibernate state by asserting (LOW) the **ATEN_SHDN** pin.

Refer to Section 2.4.5 (Using CYSPP) and Section 3.2 (Cable Replacement Examples with CYSPP) for details concerning CYSPP configuration and behavior. A full GPIO reference is available in Chapter 8 (GPIO Reference).

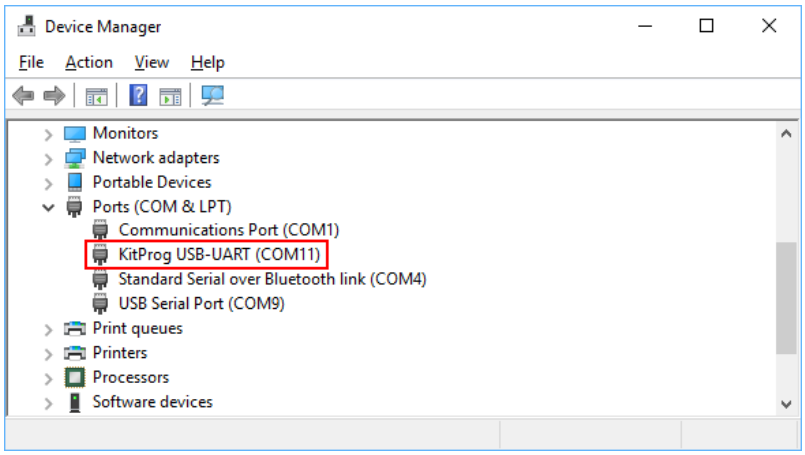
2.3 Connecting a Host Device

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. Chapter 4 (Application Design Examples) describes each of these use cases.

2.3.1 Connecting the CY8CKIT-042-BLE Pioneer Kit

When using the recommended evaluation kit for prototyping, simply connect the mini-USB cable between your PC and the main board and ensure that the EZ-Serial-compatible evaluation module is securely plugged into the receptacle. This provides power to the module and a communication interface (UART) via the kit's onboard PSoC 5LP microcontroller. Once you have connected the cable and allowed any necessary drivers to install, a new virtual COM port will become available, as shown in Figure 2-1:

Figure 2-1. Virtual Serial Port from BLE Pioneer Kit



Note: COM11 is shown here, but your port number may be different.

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

> **NOTE:** The PSoC 5LP microcontroller on the BLE Pioneer Kit board will only provide the expected USB-to-UART bridge functionality if it is running the default KitProg firmware that Infineon ships on the evaluation board. If you have changed this firmware using a debugger or bootloader, please refer to KBA87474 - PSoC® 4 Pioneer Kit (CY8CKIT-042) Factory Restore Instructions for Programmer and Debugger Functionality for instructions on restoring the default firmware.

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 (0V) level and logic HIGH at the VDD level (typically 3.3V or 5V depending on the chosen module power supply). This is necessary for high-throughput tests, which require flow control.

> **WARNING:** Do not connect the module directly to RS-232 signals. This will damage the device.

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)
    - Optional: **CTS** – Host-side flow control (input), connect to host RTS (output)

Refer to Section 8.1 (GPIO Pin Map for Supported Modules) for pin-to-function correlations.

> **NOTE:** If you connect an external UART device or adapter to the CY8CKIT-042-BLE Pioneer Kit headers for module UART access, the built-in USB-to-UART bridge interface provided by the kit's onboard PSoC 5LP will compete with it as both devices attempt to drive the module's **P1.4 UART_RX** pin. The CYUSBS232 kit is known to override the PSoC 5LP signal and successfully communicate, but some other devices or adapters may not drive or pull with the same resistive strength and will be unable to send UART data to the module. To work around this, you can either (1) erase/modify the firmware on the kit's PSoC 5LP module using PSoC Programmer, or (2) de-solder R53 to disconnect the PSoC 5LP's TX pin.

The default port settings are 115200 baud, 8 data bits, no parity, and one stop bit. Flow control is supported, but must be specifically enabled 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 they cannot be written to flash until they have first been applied to RAM. This prevents unintentional communication lockouts. Refer to Section 2.5.3 (Protected Configuration Settings) for details concerning protected settings.

If you experience any problems communicating over the serial interface, refer to Chapter 6 (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 two or four pins used for UART communication. All of these pin functions are enabled by default, but many 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 below summarizes the functions provided by these pins. For additional information including module-specific pin assignments, operational side-effects, and default logic states, refer to Chapter 8 (GPIO Reference).

Table 2-1. GPIO Function Summary

Pin name	Direction	Optional*	Functional Description
LP_MODE	Input	No	Low-power mode control. Assert (LOW) to prevent sleep, de-assert (HIGH) to allow sleep.
ATEN_SHDN	In/Out	Yes	Bidirectional signal. Host can assert (LOW) to stop all activity and force immediate hibernation. 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.
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 Section 2.4.5 (Using CYSPP Mode) for detail.
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 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 Infineon and is not intended for general use.

**Optional pin functions can be disabled to allow standard GPIO behavior*

By default, the pins noted as **output** are not strongly driven, but instead are internally pulled to the indicated states with approximately 5.6 kOhms. This prevents unintentional damage in cases where the initial power-on state of an externally connected device's pins could otherwise result in a direct short between opposite supply lines. 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, please refer to Chapter 8 (GPIO Reference).

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 sections below describe these modes in detail.

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

- **CYSPP** pin externally de-asserted (HIGH): **command mode** (text or binary)
    - **CYSPP** pin externally asserted (LOW): **CYSPP mode**
    - **CYSPP** pin left floating: **command mode** until activating CYSPP data pipe, then **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 what appears to be a 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 in order to provide the fastest possible path to rapid prototyping. Commands are typed using short codes, and responses and events come back 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.
- 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 Section 7.4).

- 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 (e.g. -1 = `FF`, -16 = `F0`, -128 = `80`).
- All multi-byte numeric data is entered and expressed in big-endian byte order (e.g. 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. This requirement allows a wider range of "dummy" byte options when using wake-on-RX. See Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition) for detail.
- 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. These all use the same basic syntax, but execute different types of behavior.

Table 2-2. Text Mode Command Categories

Category	Features
ACTION	ACTION commands trigger operations that cannot persist across resets or power-cycles, with very few exceptions. They accomplish things such as connection establishment, querying of GPIO logic states, entry into advertisement mode, and remote GATT discovery and data transfer. The exceptions to the "current session only" rule are these: • system_store_config (/SCFG, ID=2/4), used to write all modified settings to flash immediately • system_factory_reset (/RFAC, ID=2/5), used to clear all modified settings and reset the module • system_write_user_data (/WUD, ID=2/11), used to write arbitrary user data to a dedicated section of flash • gatts_create_attr (/CAC, ID=5/1), used to add custom GATT database attributes • gatts_delete_attr (/CAD, ID=5/2), used to remove custom GATT database attributes • smp_pair (/P, ID=7/3), used to initiate pairing, resulting in new bonding data stored in flash • smp_delete_bond (/BD, ID=7/2), used to delete 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 non-volatile (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 them to flash. In text mode, you can also invoke a SET command with a '$' after the text code (e.g. "`SDN$,N=...`") to cause that change to be written to both RAM and flash immediately. A small number of SET commands also manage **protected settings**, which are those that can affect core chipset operation and communication. For these settings, you cannot write changed values directly to flash without first performing a *separate* write to RAM only. This prevents accidental changes that are difficult to undo. Section 2.5.3 (Protected Configuration Settings) has more detail on this behavior.
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 refer to 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.

For more information on these command categories and behaviors, refer to the configuration hierarchy in Section 2.5.1 (Factory, Boot, Runtime, and Automatic Settings) and the material in Chapter 7 (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, as long as it works with standard serial ports and can be configured to open the port with the proper baud rate, flow control, and other settings. The figure below 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.

Figure 2-2. Text Command Mode Session with PuTTY

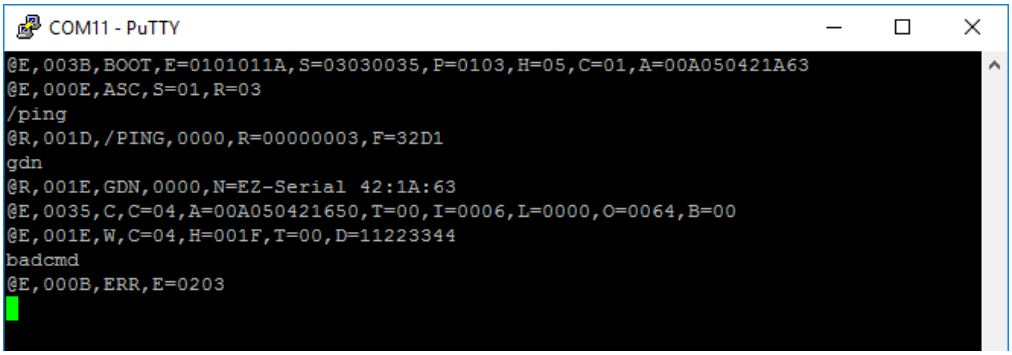


Table 2-3 describes the various protocol methods shown in the figure above.

Table 2-3. Text Mode Communication Example

Direction	Content	Detail
←RX	`@E,003B,BOOT,E=0101011A,S=03030035, P=0103,H=05,C=01,A=00A050421A63`	system_boot (BOOT, ID=2/1) API event received: app = 1.1.1 build 26 stack = 3.3.0 build 53 protocol = 1.3 hardware = CYBLE-2120XX-X0 module boot cause = power-on/XRES MAC address = 00:A0:50:42:1A:63
←RX	`@E,000E,ASC,S=01,R=03`	gap_adv_state_changed (ASC, ID=4/2) API event received: **state** = 1 (active) **reason** = 3 (CYSPP operation)
TX→	`/ping`	system_ping (/PING, ID=2/1) API command sent to ping the local module to verify proper communication
←RX	`@R,001D,/PING,00000000,R=00000003,F=32 D1`	system_ping (/PING, ID=2/1) API response received: **result** = 0 (success) **runtime** = 3 seconds **fraction** = 13009/32768 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 42:1A:63`	gap_get_device_name (GDN, ID=4/16) API response received: **result** = 0 (success) **name** = “EZ-Serial 42:1A:63”
←RX	`@E,0035,C,C=01,A=00A050421650, T=00,I=0006,L=0000,O=0064,B=00`	gap_connected (C, ID=4/5) API event received: **conn_handle** = 1 **peer** = 00:A0:50:42:16:50 **addr_type** = 0 (public) **interval** = 6 (7.5ms) **slave_latency** = 0 **supervision_timeout** = 0x64 (100 = 1 second) **bond** = 0 (not bonded)
←RX	`@E,001E,W,C=01,H=001B,T=00, D=0200`	gatts_data_written (W, ID=5/2) API event received: **conn_handle** = 1 **attr_handle** = 0x1B (27) **type** = 0 (simple write) **data** = 2 bytes `[02 00]`
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)

Refer to the reference material in Chapter 7 (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 much 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.

Table 2-4. Binary Packet Structure

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

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 structure above 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 Section 7.1 (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.

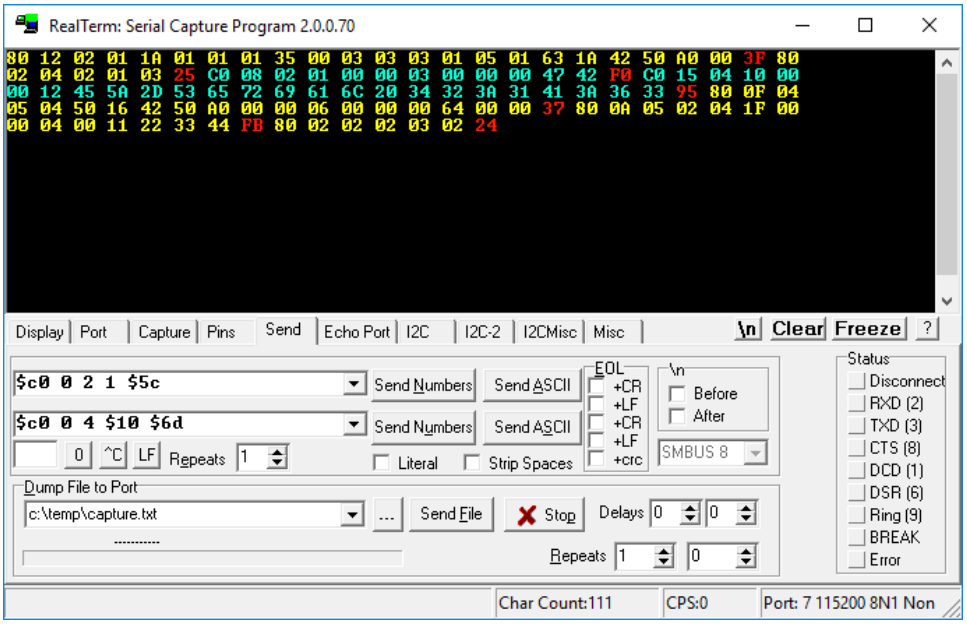
- The binary command header includes a single bit in the first byte which performs the same duty as the '$' character in text mode, to cause changed settings to be written to flash immediately instead of just RAM.

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 Infineon and discussed in Chapter 5 (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 this type of test using Realterm, including hexadecimal representation of data. There is no local echo when binary mode is used, so the screenshot 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**.

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.

Table 2-5. Binary Mode Communication Example

Direction	Content	Detail
←RX	`80 12 02 01 1A 01 01 01 35 00 03 03 03 01 05 01 63 1A 42 50 A0 00 3F`	system_boot (BOOT, ID=2/1) API event received: **app** = 1.1.1 build 26 **stack** = 3.3.0 build 53 **protocol** = 1.3 **hardware** = CYBLE-2120XX-X0 module **boot cause =** power-on/XRES **MAC address** = 00:A0:50:42:1A:63
←RX	`80 02 04 02 01 03 25`	gap_adv_state_changed (ASC, ID=4/2) API event received: **state** = 1 (active) **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

Direction	Content	Detail
←RX	`C0 08 02 01 00 00 03 00 00 00 47 42 F0`	system_ping (/PING, ID=2/1) API response received: **result** = 0 (success) **runtime** = 3 seconds **fraction** = 16967/32768
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 34 32 3A 31 41 3A 36 33 95`	gap_get_device_name (GDN, ID=4/16) API response received: **result** = 0 (success) **name** = "EZ-Serial 42:1A:63"
←RX	`80 0F 04 05 04 50 16 42 50 A0 00 00 06 00 00 00 64 00 00 37`	gap_connected (C, ID=4/5) API event received: **handle** = 4 **peer** = 00:A0:50:42:16:50 **addr_type** = 0 (public) **interval** = 6 (7.5ms) **slave_latency** = 0 **supervision_timeout** = 0x64 (100 = 1 second) **bond** = 0 (not bonded)
←RX	`80 0A 05 02 04 1F 00 00 04 00 11 22 33 44 FB`	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)

Refer to the reference material in Chapter 7 (API Protocol Reference) for details concerning 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 each 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 lists contain important factors to consider when choosing which mode to use.

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 (e.g. another Infineon 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, etc.) 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 (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 detail on this type of temporary configuration, see Section 2.5.1 (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.

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

For configuration examples in each mode, refer to Section 3.2 (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 communicate whether it is safe to write new data or not.
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 vs. 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 has ended, 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 which options can be changed, and how to do so:

- 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 starting in EZ-Serial v1.1. 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 the 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 following structure:

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

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

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 for detail). The factory default value is the Infineon 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 enable user-defined advertisement content. If you have configured user-defined advertisement content instead as described in Section 3.4.3 (How to Customize 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 in order 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 for comparison 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

In order for EZ-Serial to initiate a connection to a CYSPP peripheral device, the "**remote_key**" value must match with 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 below provides examples of each case.

Table 2-7. Connection Key and Mask Examples

Remote Key	Remote Mask	Key & 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

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 also have a Manufacturer Data field containing a connection key in the same advertisement packet, the value "0" will be substituted for an actual key for the purpose of filtering on 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 below 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 these two key behaviors concerning hardware control vs. 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.

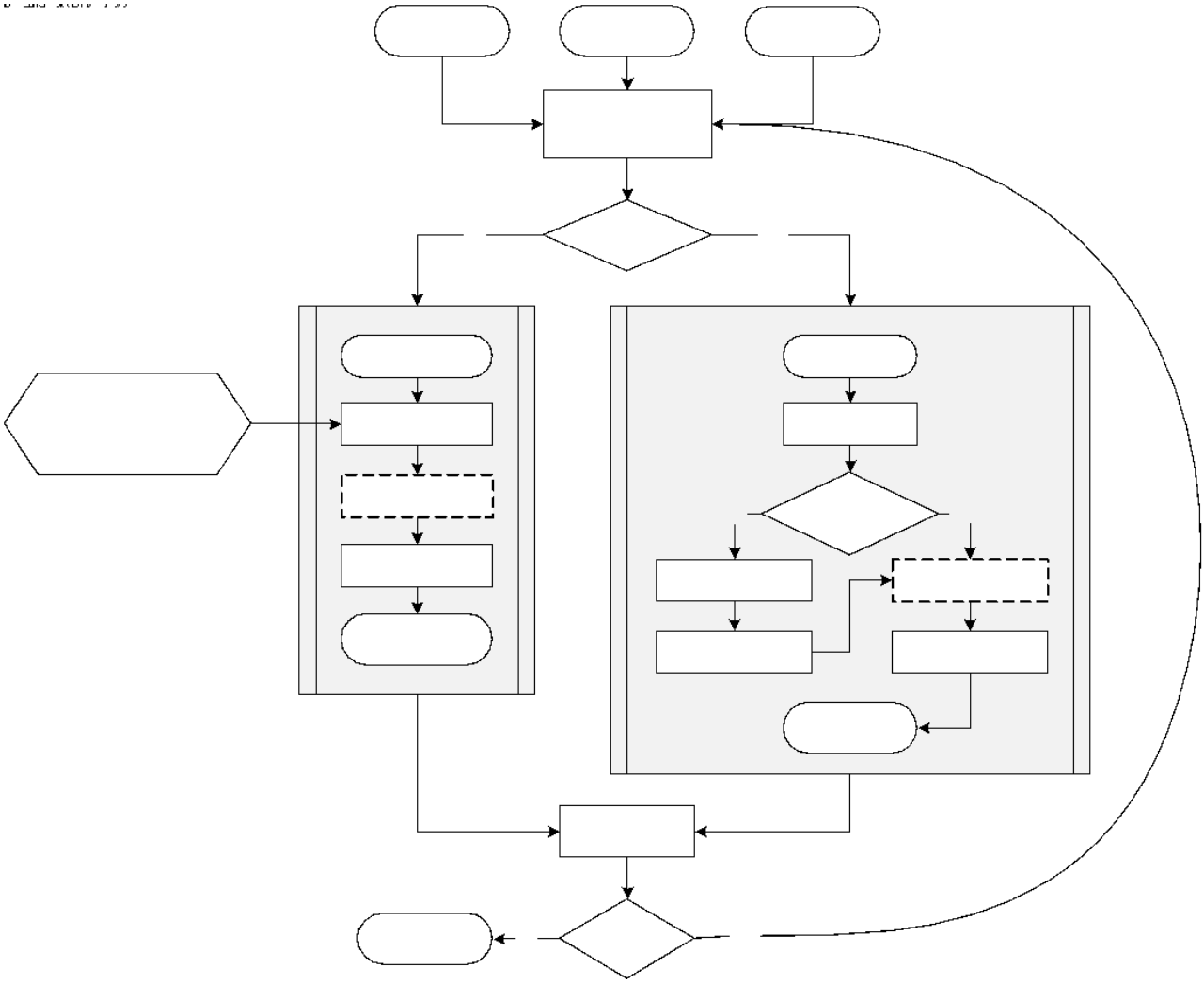
Table 2-8. CYSPP Configuration and Pin Relationship

CYSPP pin state	CYSPP "enable" value in configuration	CYSPP Operation
Floating **(assumed default)**	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.** 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 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 configured role and **CP_ROLE** pin state. API events (boot, state changes, connections, etc.) will **not be visible** over UART, because API communication is always suppressed when **CYSPP** pin is asserted.

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.

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.

2.5 Configuration Settings, Storage, and Protection

The EZ-Serial platform provides methods to customize its many built-in functions. It's 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 2-9 below describes each type of configuration storage in detail.

Table 2-9. Configuration Setting Storage Layers

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 at all times, 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.

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. It is simpler than the alternative if you are unsure which settings 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. This is simpler than the alternative if you know exactly which settings need to be changed, since 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 Chapter 7 (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, the internal chipsets used in the EZ-BLE 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 Infineon's website:

https://www.infineon.com/cms/en/design-support/software/device-driver-libraries/airoc-wi-fi-bluetooth-ez-serial-module-firmware-platform/

These images are suitable for both SWD interface re-flashing through PSoC Programmer and for bootloader updates over BLE in the case of target devices with 256K of flash memory or more. Please refer to Section 3.11 (Device Firmware Update Examples) for details about 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 Infineon's website:

https://www.infineon.com/cms/en/design-support/software/device-driver-libraries/airoc-wi-fi-bluetooth-ez-serial-module-firmware-platform/

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. Refer to Chapter 7 (API Protocol Reference) of this document for detailed material concerning the API structure and protocol.

3. Operational Examples

EZ-Serial provides a great platform on which to build a wide variety of BLE applications. The sections below describe many common operations that you can experiment with or combine together 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 detail on the API methods used in each case and the binary equivalents of each command, response, and event, refer to the material in Chapter 7 (API Protocol Reference).

3.1.1 How to Identify the Running Firmware and BLE Stack Version

The EZ-Serial firmware, BLE 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 this:

```
@E,003B,BOOT,E=0101011A,S=03030035,P=0103,H=05,C=01,A=00A050421A63
```

If the protocol parser is in **binary mode**, this event will be similar to that shown below, expressed in hexadecimal notation:

Header	Payload	Checksum
80 12 02 01	1A 01 01 01 35 00 03 03 03 01 05 01 63 1A 42 50 A0 00	3F

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 3-1.

Table 3-1. Payload Detail for Boot Event

Text Code	Text Data	Binary Data	Details	Interpretation
E	“0101011A”	1A 01 01 01	EZ-Serial application version	Version 1.1.1 build 26 (0x1A)
S	“03030035”	35 00 03 03	BLE stack version	Version 3.3.0 build 53 (0xFA)
P	“0103”	03 01	API protocol version	Version 1.3
H	“40”	05	Hardware ID	CYW20822
C	“01”	01	Cause for boot event	Power-cycle/XRES
A	“00A050421A63”	63 1A 42 50 A0 00	MAC address	00:A0:50:42:1A:63

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=0101011A,S=03030035,P=0103,H=05
```

The binary-mode response packet is as shown below:

Header	Payload	Checksum
C0 0D 02 06	00 00 1A 01 01 01 35 00 03 03 03 01 05	7A

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

3.1.2 How to Change the 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 3-2 lists common baud rates and their hexadecimal equivalents:

Table 3-2. Common UART Baud Rates and Hex Equivalents

Baud Rate	Hex Equivalent
300	12C
9,600	2580
19,200	4B00
115,200 (default)	1C200
230,400	38400
460,800	70800
1000000	F4240

NOTE: EZ-Serial supports non-standard baud rates not listed in the table above, 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: The USB-to-UART bridge provided by the BLE Pioneer Kit's PSoC 5LP microcontroller supports configurable baud rates and parity/stop bits, but does not support flow control. It is also limited to **115200** baud to remain within typical clock tolerances. You must connect an external UART device or MCU to the module's UART data and flow control pins if you wish to use flow control or faster baud rates. Refer to Section 2.3.2 (Connecting the Serial Interface) for detailed instructions and specific requirements for proper functionality when connecting an external UART device to the BLE Pioneer Kit.

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 960000 baud, no parity, flow control enabled, and store in flash

Direction	Content	Effect
TX→	`STU,B=2580,F=1`	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`	Apply new UART parameters
←RX	`@R,0009,STU,0000`	Response indicates success

3.1.3 How to Change the 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 Section 7.5 (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 the first example below.

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 can be found 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

n		
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 How to Change the 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 **-18 dBm** to **+3 dBm**. Table 3-3 lists each allowed value.

Table 3-3. Supported TX Power Output Options

Argument	Power Level
1	-20 dBm
2	-10 dBm
3	-6 dBm
4	-4 dBm
5	-2 dBm
6(default)	0 dBm
7	2 dBm
8	4 dBm

of -12 dBm for the same reason. See each module's datasheet for details about these restrictions.

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 How to Manage 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 3-4 provides a high-level summary of the four power states used by the platform.

Table 3-4. EZ-Serial Power States

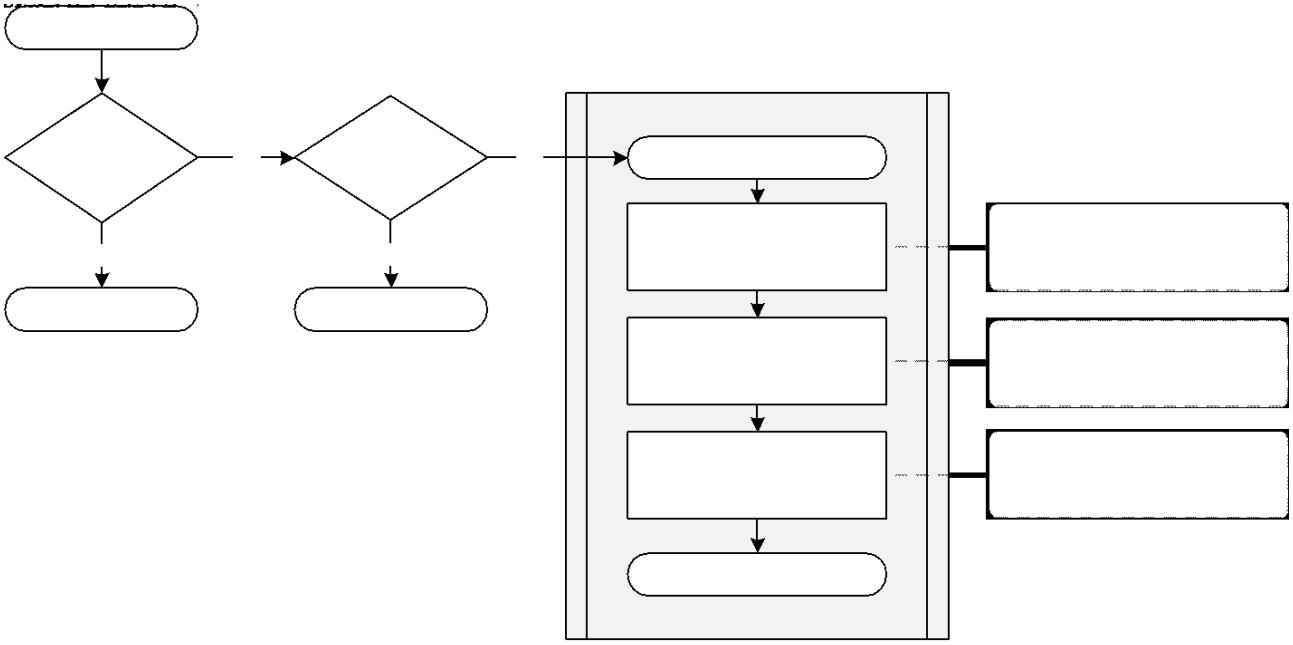
Power Mode	Current Range (typical), Vdd = 3.3 V to 5.0 V	Wakeup Time	Description
Active	235 uA	n/a	CPU and all peripherals are active. All functionality possible with no delay.
Sleep	222 uA	0	CPU is asleep, BLE subsystem is asleep, all peripherals are active. This state is only used when you have limited sleep with the system_set_sleep_parameters (SSLP, ID=2/19) API command or enabled high-resolution PWM output using the gpio_set_pwm_mode (SPWM, ID=9/11) API command. In these cases, standard sleep is used at any time when deep sleep would have been used.

Deep sleep	6.5 uA	12 ms	CPU, BLE subsystem, and UART are asleep. High-speed clocks are off. Wake-up is possible via UART RX, GPIO interrupts, BLE stack events, or scheduled tasks. Refer to Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition) for safe wake-on-RX procedures.
Hibernate	4 uA	180 ms	CPU, BLE subsystem, all peripherals, and all clocks are disabled. Wake-up is possible only by trigger interrupt pin.

The “Stop” state supported by the chipset is not currently used by the EZ-Serial platform. For a detailed discussion of low-power states in general, refer to the Infineon application note AN86233 - PSoC® 4 Low-Power Modes and Power Reduction Techniques.

EZ-Serial uses the maximum allowed sleep level based 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. Figure 3-1 below describes the sleep level determination logic.

Figure 3-1. EZ-Serial Sleep State Behavior



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

1.
2. If the **LP_MODE** pin is asserted, remain in **active** mode. Otherwise:
3. Select the *lowest* value (**0** = no sleep, **1** = normal sleep, **2** = deep sleep) among the following:
    a. The system sleep level configured with system_set_sleep_parameters (SSLP, ID=2/19) API command.
    b. 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).
    c. **Normal** sleep if high-resolution PWM output is enabled with the gpio_set_pwm_mode (SPWM, ID=9/11) API command.

NOTE: EZ-Serial does not allow changes to the sleep level calculation hierarchy order. For example, if CYSPP sleep level is “2” (deep sleep) but system-wide sleep is level “1”, then the system-wide setting will override the CYSPP setting because it is a lower value. EZ-Serial will always select the lowest applicable value for the current operational state.

This fine-grained level of control over sleep mode selection in various operational states allows you to achieve the most efficient power consumption supported by your application design. For example, you may allow deep sleep at all times

except when the CYSPP data pipe is open, in order to easily avoid potential initial-byte data corruption at high baud rates. For more detail, see Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition).

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 which sleep mode to use.

Active PWM output will limit the effective maximum sleep level in any state to **normal sleep** (value = 1) if another setting is net even lower than this. If the CYSPP data pipe is open (connected and in CYSPP data mode), then the CYSPP-specific sleep level may further limit the effective maximum sleep level. Refer to Figure 3-1 for a diagram showing how EZ-Serial determines which sleep level to use.

EZ-Serial allows only **normal sleep** (value = 1) as the factory default system-wide sleep level, for a simpler out-of-the-box experience concerning UART communication. However, you can change this to allow **deep sleep** to significantly improve average current consumption. Ensure that your application can properly work within this mode before applying it; refer to Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition) for details.

Example 1: Change system-wide sleep level to deep sleep

Direction	Content	Effect
TX→	`SSLP,L=2`	Set new system sleep level to "deep sleep"
←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 Section 3.1.5.3 (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. When sleep is not disabled using the **LP_MODE** pin, this value is the second limit applied when calculating which sleep mode to use. The system-wide sleep level takes precedence over the CYSPP sleep level. Further, PWM output will limit the effective maximum sleep level in any state to **normal sleep** (value = 1), regardless of other settings. Refer to Figure 3-1 for a diagram showing how EZ-Serial determines the sleep level to use.

Setting the CYSPP data mode sleep level to **normal sleep** (value = 1) or **no sleep** (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 **deep sleep** (value = 2), but keep in mind that factory defaults also set the system-wide sleep level limit to **normal sleep** (value = 1), which prevents deep sleep at all times unless you reconfigure it.

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 **deep sleep** and the CYSPP data mode sleep level to **normal sleep**. The CPU will sleep aggressively until a remote peer opens the CYSPP data pipe, at which point the CPU will use only **normal sleep** so that the wired external host does not need any special sleep/wake transition control.

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

Direction	Content	Effect
TX→	`.CYSPPSP,S=1`	Set new CYSPP sleep level to "normal sleep"
←RX	`@R,000E,.CYSPPSP,0111,S=01`	`Reboot reuired`

3.1.5.3 Preventing Sleep with the LP_MODE Pin

Assert (LOW) the **LP_MODE** control pin to prevent the module from sleeping under any circumstances other than a forced shutdown via the **ATEN_SHDN** pin. 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. Refer to Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition) for more detail.

> **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 Preventing Activity with the ATEN_SHDN Pin

Assert (LOW) the **ATEN_SHDN** pin to force EZ-Serial into hibernation regardless of other activity, including the state of the **LP_MODE** pin. In this state, the BLE subsystem, CPU, and peripheral interfaces are completely disabled. You must de-assert the **ATEN_SHDN** pin in order to wake up again.

> **NOTE:** Using the **ATEN_SHDN** pin to put the module into a hibernation state retains values in RAM, but otherwise behaves like a chipset reset. De-asserting (HIGH) this pin after hibernation will result in the generation of the system_boot (BOOT, ID=2/1) API event. Monitor the "**cause**" parameter of this event to detect whether it occurs due to waking from hibernation or not.

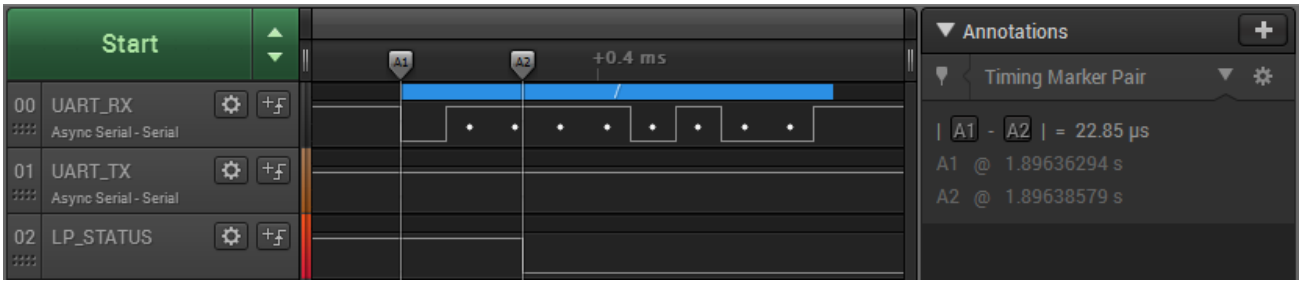
> **WARNING:** Asserting the **ATEN_SHDN** pin will immediately terminate any BLE communication without cleanly disconnecting first. If you require a clean disconnection, you should use the gap_disconnect (/DIS, ID=4/5) API command to close an active connection from an external host prior to asserting the **ATEN_SHDN** pin.

For more detail concerning the **LP_MODE**, **LP_STATUS**, and **ATEN_SHDN** pins, refer to GPIO reference material in Chapter 8 (GPIO Reference).

3.1.5.5 Avoiding UART Data Loss or Corruption due to Deep Sleep Transition

Allowing **deep sleep** provides the best average power consumption. However, because the UART peripheral cannot operate in deep sleep mode, supporting UART communication while also allowing deep sleep requires special consideration. It takes approximately 25 µs for the CPU to transition from deep sleep to fully awake, and any UART data sent during this time will be lost. The UART peripheral will begin processing data on the first *falling* edge detected after waking, which can result in persistent bit misalignment and incorrect data reported to the API parser, illustrated in the figure below. The time between the A1 and A2 markers represents the CPU wake-up delay.

Figure 3-2. Deep Sleep Wake-on-RX Without Dummy Byte at 115200 Baud, 8/N/1



In the example shown in Figure 3-2, the forward slash character ('/', 0x2F) contains three falling edges. The first one is the actual start bit, but the module only begins processing UART data after the second falling edge occurs, resulting in the following scenario:

Table 3-5. Wake-on-RX Bit Misalignment from Deep Sleep at 115200 Baud

Host transmits **0x2F** (0b00101111 in LSB order)										*Host idle (HIGH)*				
Start	1	1	1	1	0	1	0	0	**Stop**	5-bit misalignment				
5-bit misalignment					**Start**	1	0	0	1	1	1	1	1	**Stop**
CPU wakes		*Wait for start*			Module receives **0xF9** (0b11111001 in LSB order)									

While the above case describes one possible outcome, the exact nature of received data corruption depends on the transmitted bytes, baud rate, and other UART parameters. Therefore, to avoid potential data loss or corruption during the 25 µs transition between Deep Sleep mode and Active mode, you must implement one of the methods described in Table 3-6 below.

Table 3-6. Deep Sleep UART Corruption Avoidance Methods

Method	Advantages	Disadvantages
Disable deep sleep in software by changing system sleep parameters using the system_set_sleep_parameters	Simple and effective, requires no special consideration or behavior	Higher power consumption caused by only using normal sleep mode instead

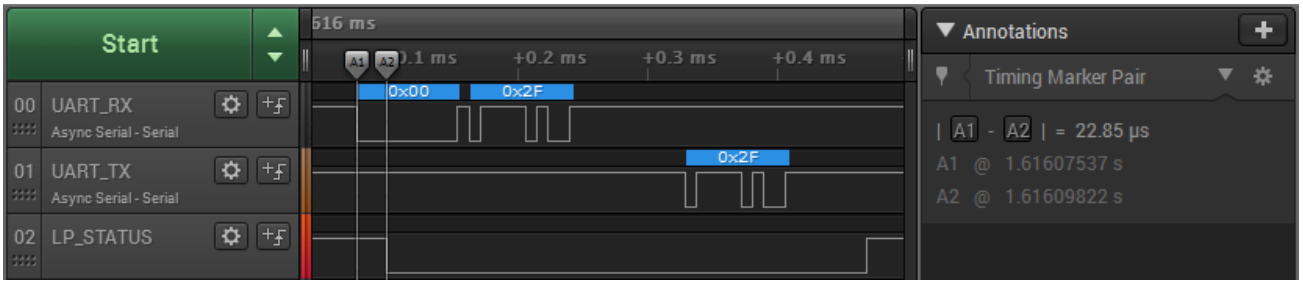
(SSLP, ID=2/19) API command, or (if using CSYPP) by changing the CYSPP-specific sleep parameters using the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command.	on the external host.	of deep sleep mode in certain operational states.
Disable all sleep in hardware by externally asserting (LOW) the **LP_MODE** pin.	Simple and effective, requires no special consideration or behavior on the external host or configuration within EZ-Serial.	High power consumption caused by constantly active CPU.
Externally assert (LOW) the **LP_MODE** pin at least 25 µs prior to sending data, and only de-assert after the last transmitted byte is fully clocked into the module.	Allows most efficient sleep state usage with minimal overhead, works in CYSPP data mode as well as command mode.	Requires additional GPIO connection between host and module, and special application logic on the host.
Begin each transmission with one or more dummy bytes to allow at least 25 µs for power state transition after the first UART start bit.	Allows efficient sleep state usage with minimal overhead, requires no additional GPIO connection.	Requires special logic on the host, only works reliably in command mode where known start-of-packet bytes allow mutually exclusive “dummy” byte selection. Usage in CYSPP mode requires application tolerance of dummy bytes randomly placed within raw data stream if the host transmits them while the CPU is already awake.

For the “dummy byte” method, the UART RX wake signal begins at the start bit’s falling edge, and any data sent before the 25 µs sleep state transition time interval will not be processed. Skipping over the calculations involved, this means you must send:

- At least **1** dummy byte if the data rate is below 416 kbaud
- At least **2** dummy bytes if the data rate is above 416 kbaud but below 833 kbaud
- At least **3** dummy bytes if the data rate is above 833 kbaud through the maximum supported 921 kbaud

Use the NULL byte (0x00) as the dummy byte, since the API parser will ignore it as a start-of-packet byte whether you are using text mode or binary mode. Also, 0x00 always contains exactly one falling edge (the start bit) regardless of UART parity settings.

Figure 3-3. Deep Sleep Wake-on-RX With Dummy Byte at 115200 Baud, 8/N/1

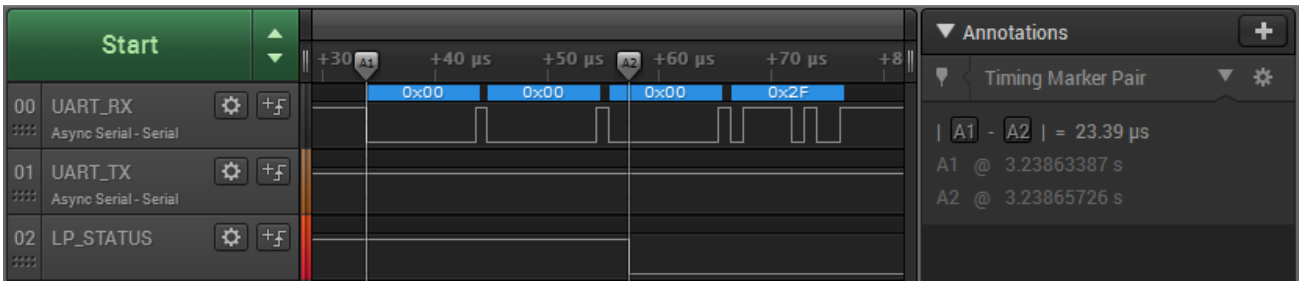


In the example shown in Figure 3-3, the CPU has fully transitioned to active mode before the host begins sending the forward slash (‘/’) character, allowing correct data reception into the module.

> **WARNING:** Although the API parser will safely ignore as many 0x00 dummy bytes as the host transmits even if the CPU is already awake, CYSPP data mode does not have this same guarantee. Since the module may have woken already for BLE connection management purposes, a dummy byte sent in CYSPP mode could be fully received and transmitted to the remote peer. For this reason, you should either (1) choose a different workaround for CYSPP mode, or (2) design your application protocol to tolerate spurious dummy bytes appearing in the data stream in case this occurs.

For reference, the following diagram shows the wake timing at 921600 baud with three consecutive dummy bytes. The time between the A1 and A2 markers represents the CPU wake-up delay, which extends into the middle of the third dummy byte.

Figure 3-4. Deep Sleep Wake-on-RX With Dummy Byte at 921600 Baud, 8/N/1



3.1.5.6 Managing Host and Module Sleep Simultaneously

In applications that include both an external host MCU and a BLE module, usually both components need to sleep in order 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 deasserted (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 How to Perform 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,003B,BOOT,E=0101011A,S=03030035,P=0103,H=05, C=05,A=00A050421A63`	Event indicates system has rebooted, cause is set to 0x05 (factory reset)

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 Section 2.4.5 (Using CYSPP Mode) for a description of how CYSPP mode behaves generally and how it affects API communication.

3.2.1 How to Get 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 **BLE Pioneer Kit** for evaluation, perform the following steps:

1. Open the kit-provided COM port in your terminal software of choice, being sure to use the 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 no flow control.
2. To use CYSPP in central/client mode, hold down the SW2 button and press and release SW1 to reboot in central mode. You can release SW2 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 Section 2.4.5 (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 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=05` (running in peripheral role)

   `@E,000C,.CYSPP,S=15` (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. If left floating, EZ-Serial will use the role configured in firmware using the p_cyspp_set_parameters (.CYSPPSP, ID=10/3) API command. 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. If the CYSPP pin is left floating:
   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=05` (running in peripheral role)

      `@E,000C,.CYSPP,S=15` (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 How to Start 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 Section 2.4.5.2 (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,003B,BOOT,E=0101011A,S=03030035,P=0103,H=05,C=01,A=00A050421A63`	Boot event
←**RX**	`@E,000E,ASC,S=01,R=03`	CYSPP-triggered advertisement started
←**RX**	`@E,0035,C,C=01,A=00A050421650,T=00,I=0006,L=0000,O=0064,B=00`	Connection established with remote device
←**RX**	`@E,001A,W,C=01,H=001B,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=0018,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=05`	CYSPP status update (0x05): • 0x04: Subscribed to RX flow control • 0x01: Subscribed to unacknowledged data
Host may now send data to the module for delivery to the remote peer, received data comes from peer		

3.2.1.2 How to Start 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. If you are using the BLE Pioneer Kit, simply hold down the **SW2** button while momentarily pressing the **SW1** button to reset the module.

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

Direction	Content	Effect
←**RX**	`@E,003B,BOOT,E=0101011A,S=03030035,P=0103,H=05,C=01,A=00A050E3835F`	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=00,A=00A050421650,T=00,I=0006,L=0000,O=0064,B=00`	Connection established with remote device
←**RX**	`@E,0029,DR,C=00,H=0001,R=0009,T=2800,P=00,U=0018`	GATT discovery result (0x1800)
←**RX**	`@E,0029,DR,C=00,H=000A,R=000A,T=2800,P=00,U=0118`	GATT discovery result (0x1801)
←**RX**	`@E,0045,DR,C=00,H=0012,R=001B,T=2800,P=00,U=00A10C2000089A9EE21115A133333365`	GATT discovery result (CYSPP service)
←**RX**	`@E,0045,DR,C=00,H=001C,R=0022,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 ack'd data)
←**RX**	`@E,0029,DR,C=04,H=000F,R=0000,T=2902,P=00,U=0229`	GATT discovery result (configuration descriptor)

Direction	Content	Effect
←**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 unack'd 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=00,H=001B,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=00,H=001A,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=00,H=0018,R=0000`	Remote server acknowledged write operation which enabled notifications on unacknowledged data characteristic
←**RX**	`@E,000C,.CYSPP,S=15`	CYSPP status update (0x15): ● 0x10: CYSPP peer support verified ● 0x04: Subscribed to RX flow control ● 0x01: Subscribed to unacknowledged data
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 and/or a specific level of encryption and bonding 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 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.

EZ-Serial implements the GATT server side of CYCommand behavior using the GATT structure detailed in Section 9.3 (CYCommand Profile).

> **NOTE:** CYCommand access requires the module to be connectable in order 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:
>
> 1. Use gap_start_adv (/A, ID=4/8) sent from a host to advertise on demand
> 2. Use gap_set_adv_parameters (SAP, ID=4/23) to auto-start advertising on boot
> 3. Use p_cyspp_set_parameters (.CYSPPSP, ID=10/3) to auto-start peripheral role CYSPP operation

3.3.1 How to Secure 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 but require 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 type 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

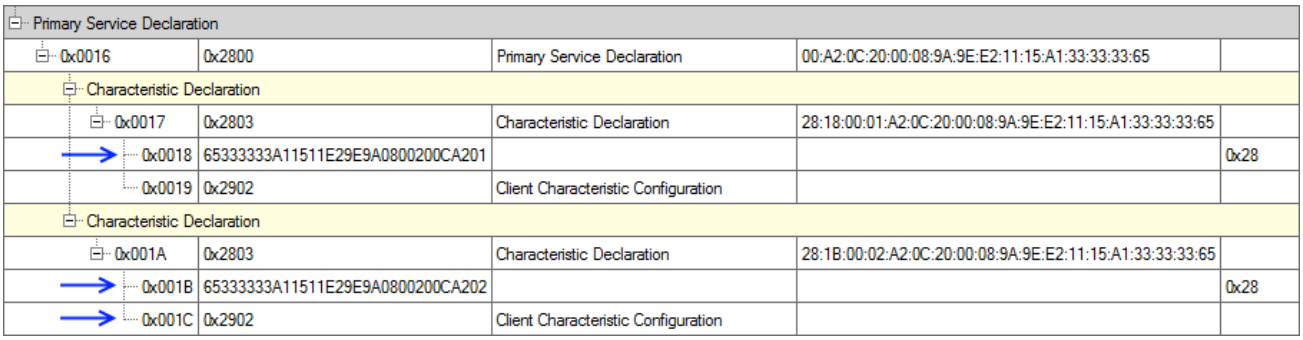
Example 2: Require CYCommand password “Infineon”, store in flash

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

3.3.2 How to Send and Receive API Commands over GATT

EZ-Serial implements the GATT server side of CYCommand behavior using the GATT structure detailed in Section 9.3 (CYCommand Profile). Figure 3-5 shows the CYCommand service structure as discovered and organized in the CySmart application, with the three most relevant attributes highlighted.

Figure 3-5. 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 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.

   Example commands to try:

   a. `2F50494E470A` – system_ping (/PING, ID=2/1) in text mode

   b. `C00002015C` – system_ping (/PING, ID=2/1) in binary mode

   c. `47444E0A` – gap_get_device_name (GDN, ID=4/16) in text mode (response comes in multiple packets)

   d. `C006090502FF000000006E` – gpio_set_drive (SIOD, ID=9/5) in binary mode, set all Port 2 pin drive modes to high-Z digital input

   e. `C00109010266` – gpio_query_logic (/QIOL, ID=9/1) in binary mode, query Port 2 logic state

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), then 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 one final time 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 at the moment a connection is initiated based on which side initiates the connection.

The GAP layer provides four different 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 How to Advertise as Peripheral Device

Advertising is the BLE activity which allows scanning devices to observe and connect to peripherals. It is required in order 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 by 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 as a result 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
←RX	`@E,000E,ASC,S=01,R=00`	Event indicates advertising state changed to “active”

Example 2: Start advertising with custom parameters

Direction	Content	Effect
TX→	`/A,M=2,T=0,I=A0,C=6,F=0,O=1E`	Begin advertising with custom arguments
←RX	`@R,0008,/A,0000`	Response indicates success
←RX	`@E,000E,ASC,S=01,R=00`	Event indicates advertising state changed to “active”

3.4.2 How to Stop Advertising 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 How to Customize 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 which profiles 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.

Key features and requirements for customizing data:

- Each of the 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 things like 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 3-7 lists fields that are most commonly used:

Table 3-7. Common Advertisement Field Types

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)

EZ-Serial does not validate advertisement or scan response payload content, and the underlying BLE stack has only limited validation on the Flags field. You must ensure 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-only devices which do not support “classic” BR/EDR Bluetooth behavior, this means that the flags byte will almost always be one of these three values:

- 0x04: Non-discoverable/broadcast-only (common for beacon-only devices)
- 0x05: Limited discoverable
- 0x06: General discoverable (most common for connectable devices)

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

Table 3-8 provides examples for reference:

Table 3-8. Examples of Well-Formed Advertisement Fields

Byte content	Field Description
`02 01 06`	**Length:** 2 bytes **Type:** Flags (0x01) **Value:** LE General Discoverable Mode, BR/EDR Not 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 (Infineon Semiconductor) Data = `[AA BB CC DD EE FF]`

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 06 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 as long as 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=020106050209180D18`	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=020106110700a10c2000089a9ee21115a133333365`	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, refer to the material at the beginning of Section 3.4 (GAP Peripheral Examples).

3.5.1 How to Scan 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 ensure that known or compatible peripherals are nearby and connectable.

> **NOTE:** If you do not have any automatic scan timeout set, then 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.

> **NOTE:** 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 started as a result 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 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 vs. 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 than 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, a large number of 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,`	Event indicates scan result from 00:A0:50:E3:83:5E, normal ad packet, RSSI -47 dBm (0xB1), Flags field and 128-bit

	D=0201061107CA366D7D5BCC0288B14DE541D9FF652F	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 How to Stop 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. Please ensure 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 How to Connect 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 a 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:** At this time, the Infineon Bluetooth stack supports one active connection at a time. In order 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 Section 3.8.1 (How to Use 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. This is why a peripheral device must be in an advertising state in order 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

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`	Initiate connection
←RX	`@R,000D,/C,0000,H=00`	Response indicates success
←RX	`@E,0030,C,H=04,A=00A050E3835E,T=00,I=0010,L=0000,O=0064`	Event indicates connection opened

3.5.4 How to Cancel a 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,A=00A050E3835E`	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 How to Disconnect 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`	Disconnect from peer
←RX	`@R,000A,/DIS,0000`	Response indicates success
←RX	`@E,0010,DIS,H=04,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 How to Define 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, and do not persist across power-cycles or resets.

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

A GATT structure is fundamentally made up of individual attributes, each of which has 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.

> **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. The space left for custom entries depends on whether the device running EZ-Serial has 128K or 256K of flash memory. The table below lists each relevant value on both platforms:

Table 3-9. Dynamic GATT Structural Limitations

Category	Built-in	128K Flash		256K Flash	
		Total	Avail.	Total	Avail.
Attribute definitions (flash)	26	64	38	128	102
Attribute value containers (SRAM)	2	64	62	128	126
Client Characteristic Configuration descriptors (flash and SRAM)	6	32	26	64	58
Storage pool for UUIDs and default attribute values (flash)	84 bytes	256 bytes	172 bytes	512 bytes	428 bytes
Storage pool for runtime attribute values (SRAM)	12 bytes	512 bytes	500 bytes	1024 bytes	1012 bytes

EZ-serial allocates 4096 bytes for storing GATT attributes data. The Generic Access, Generic Attribute, CYSPP, and CYCMD are built-in services, their handle from 1 to 34. And 4096 bytes only for storing custom attributes. Now the maximum index for CYSPP, CYCMD and custom attribute is 255, so the custom attribute number is about 237.

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 Section 7.4 , 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 which all have the UUID 0x2800, which is the official “Primary Service Declaration” UUID defined by the Bluetooth SIG. Table 3-10 lists notable pre-defined structural definition UUIDs from the Bluetooth SIG.

Table 3-10. Bluetooth SIG Structural UUIDs

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

Further detail on these and other official identifiers can be found on 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 follows:

Table 3-11. Required Characteristic Attribute Order

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
#7	0x2904	Characteristic Format	No
#8	0x2905	Characteristic Aggregate Format	No

Any optional attributes may be omitted as long as all provided attributes are supplied in the above order.

After adding all attributes by gatts_create_attr (/CAC, ID=5/1), You need to send gatts_service_active (/SACT, ID=5/16) to activate these attributes.

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, refer to Section 10.2 (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

It is critical to use correct permissions when defining any custom GATT structural elements. Refer to Section 10.2 (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 they 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)

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 is able to discover the database structure correctly, even if additional security is required to execute read and/or write 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.

3.6.2 How to List 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 Section 3.7.1, How to Discover 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_remote_procedure_complete (RPC, 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=0002**	Response indicates success, 2 records to follow
←RX	**@E,0040,DL,H=0012,R=001B,T=2800,P=00,U=00A10C2 000089A9EE21115A133333365**	Service 65333333-A115-11E2-9E9A-0800200CA100, start=18 (0x12), end=27 (0x1B)
←RX	**@E,0040,DL,H=001C,R=0022,T=2800,P=00,U=00A20C2 000089A9EE21115A133333365**	Service 65333333-A115-11E2-9E9A-0800200CA200, start=28 (0x1C), end=34 (0x22)

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=0005**	Response indicates success, 5 records to follow
←RX	**@E,0040,DL,H=0013,R=0014,T=2803,P=28,U=01A10C20 00089A9EE21115A133333365**	Char 0x6533…A101, decl handle=0x13, value handle=0x14, perm=0x028
←RX	**@E,0040,DL,H=0016,R=0017,T=2803,P=14,U=02A10C20 00089A9EE21115A133333365**	Char 0x6533…A102, decl handle=0x16, value handle=0x17, perm=0x14
←RX	**@E,0040,DL,H=0019,R=001A,T=2803,P=20,U=03A10C20 00089A9EE21115A133333365**	Char0x6533…A103, decl handle=0x19, value handle=0x1A, perm=0x20
←RX	**@E,0040,DL,H=001D,R=001E,T=2803,P=28,U=01A20C2 000089A9EE21115A133333365**	Char 0x6533…A201, decl handle=0x1D, value handle=0x1E, perm=0x28
←RX	**@E,0040,DL,H=0020,R=0021,T=2803,P=28,U=02A20C20 00089A9EE21115A133333365**	Char 0x6533…A202, decl handle=0x20, value handle=0x21, perm=0x28

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=0011**	Response indicates success, 17 records to follow
←RX	**@E,0024,DL,H=0012,R=001B,T=2800,P=00,U=0028**	UUID 0x2800 (Primary Service), start=0x12, end=0x1B
←RX	**@E,0024,DL,H=0013,R=0014,T=2803,P=28,U=0328**	UUID 0x2803 (Characteristic), decl=x013, value handle=0x14
←RX	**@E,0040,DL,H=0014,R=0000,T=0000,P=28,U=01A10C20 00089A9EE21115A133333365**	UUID 0x6533…A101 (CYSPP), handle=0x14, perm=0x28
Additional records omitted for brevity		
←RX	**@E,0024,DL,H=001C,R=0022,T=2800,P=00,U=0028**	UUID 0x2800 (Primary Service), start=0x1C, end=0x22
←RX	**@E,0024,DL,H=001D,R=001E,T=2803,P=28,U=0328**	UUID 0x2803 (Characteristic), decl=0x1D, value handle=0x1E, perm=0x28
←RX	**@E,0040,DL,H=001E,R=0000,T=0000,P=28,U=01A20C20 00089A9EE21115A133333365**	UUID 0x6533…A201 (CYCommand Challenge), handle=0x1E, perm=0x28

Direction	Content	Effect
←RX	@E,0024,DL,H=001F,R=0000,T=2902,P=0A,U=0229	UUID 0x2902 (CCCD), handle=0x1F, perm=0x0A
←RX	@E,0024,DL,H=0020,R=0021,T=2803,P=28,U=0328	UUID 0x2803 (Characteristic), decl=0x20, value handle=0x21, perm=0x28
←RX	@E,0040,DL,H=0021,R=0000,T=0000,P=28,U=02A20C2000089A9EE21115A133333365	UUID 0x6533…A202 (CYCommand Data), handle=20x217, perm=0x28
←RX	@E,0024,DL,H=0022,R=0000,T=2902,P=0A,U=0229	UUID 0x2902 (CCCD), handle=0x22, perm=0x0A

3.6.3 How to Read and Write 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 always possible to local 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 characteristic CCCD value

Create an TX power service (10.2.5) before testing read local attribute value

Direction	Content	Effect
TX→	/CAC,/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0418	Add a new attribute (TX Power Service) to the local GATT structure
←RX	@R,0018,/CAC,0000,H=0023,V=0001	Response indicates success
TX→	/CAC,/CAC,T=2803,R=01,W=00,C=02,L=0000,D=072A	Add a new characteristic declaration to the local GATT structure
←RX	@R,0018,/CAC,0000,H=0024,V=0001	Response indicates success
TX→	/CAC,/CAC,T=0000,R=01,W=00,C=02,L=0001,D=	Add a new characteristic value to the local GATT structure
←RX	@R,0018,/CAC,0000,H=0025,V=0000	Response indicates success
TX→	/CAC,/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=	Add a CCCD to the local GATT structure
←RX	@R,0018,/CAC,0000,H=0026,V=0000	Response indicates success
TX→	/SACT	GATT service active
←RX	@R,000B,/SACT,0000	Response indicates success
TX→	/RLH,H=26	Read attribute with handle = 0x26
←RX	@R,0011,/RLH,0000,D=0000	Response indicates success, hex data is “0000”

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.

NOTE: User-managed attributes have no RAM-backed data storage, so there is no destination for storing written data. Attempting to write this type of characteristic will generate an error result in the response. Also, service and characteristic declarations (0x2800 range) are stored in flash, and cannot be changed with this command.

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 Section 0 (

How to Notify and Indicate Data to a Remote Client) for details on how to push data.

Example 1: Write "0200" to CCCD in TX power service

Followed the steps in 3.6.3.1 example 1 to create an TX power service (10.2.5) first.

Direction	Content	Effect
TX→	`/WLH,H=26,D=0200`	Write "0200" (hex) into attribute with handle = 0x26
←RX	`@R,000A,/WLH,0000`	Response indicates success
TX→	**/RLH,H=26**	Read attribute with handle = 0x26 to verify
←RX	**@R,0011,/RLH,0000,D=0200**	Response indicates success, data shows expected value

3.6.4 How to Notify and Indicate Data to a Remote Client

Notifying and indicating both 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 both 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:

Table 3-12. Service Changed GATT Characteristic Structure

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

This characteristic supports the "indicate" operation. In order 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 = 23 (0x17)
←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: a four-byte value to a client manually using the CYSPP Acknowledged Data characteristic

Direction	Content	Effect
TX→	`/IH,H=14,D=41424344`	Write “ABCD” (hex) via attribute with handle = 0x14(0x0A)
←RX	`@R,0009,/IH,0000`	Response indicates success
←RX	`@E,000F,IC,C=04,H=0014`	Event indicates client has confirmed receipt of data

3.6.5 How to Detect and Process 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, this means that 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 the sections below require an active connection to a remote peer device, and will generate an error result if attempted without one.

3.7.1 How to Discover 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, refer to Section 0 (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.

How to List Local GATT Services, Characteristics, and Descriptors).

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`	Request to discover all remote services
←RX	`@R,000A,/DRS,0000`	Response indicates success
←RX	`@E,0029,DR,C=00,H=0001,R=0009,T=2800,P=00,U=0018`	Service 0x1800, start=1, end=9
←RX	`@E,0029,DR,C=00,H=000A,R=000A,T=2800,P=00,U=0118`	Service 0x1801, start=8, end=10 (0x0A)
←RX	@E,0045,DR,C=00,H=000B,R=0011,T=2800,P=00,U=62905C36B7C4DC8F8241B575398DE7A0	Service 0xA0E7…9062, start=11 (0x0B), end=17 (0x11)
←RX	`@E,0045,DR,C=04,H=0012,R=001B,T=2800,P=00,U=00A10C2000089A9EE21115A133333365`	Service 0x6533…A100, start=18 (0x12), end=71 (0x1B)
←RX	`@E,0045,DR,C=04,H=001C,R=0022,T=2800,P=00,U=00A20C2000089A9EE21115A133333365`	Service 0x6533…A200, start=28 (0x1C), end=34 (0x22)
←RX	`@E,0010,RPC,C=04,R=060A`	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`	Request to discover all remote characteristics

Direction	Content	Effect
←**RX**	`@R,000A,/DRC,0000`	Response indicates success
←**RX**	`@E,0029,DR,C=00,H=0002,R=0003,T=2803,P=02,U=002A`	Char 0x2A00, decl handle=2, value handle=3, perm=0x02
←**RX**	`@E,0029,DR,C=00,H=0004,R=0005,T=2803,P=02,U=012A`	Char 0x2A01, decl handle=4, value handle=5, perm=0x02
←**RX**	`@E,0029,DR,C=00,H=0006,R=0007,T=2803,P=02,U=042A`	Char 0x2A04, decl handle=6, value handle=7, perm=0x02
←RX	@E,0029,DR,C=000,H=0008,R=0009,T=2803,P=02,U=C92A	Char 0x2AC9, decl handle=8, value handle=9, perm=0x02
←**RX**	`@E,0029,DR,C=00,H=000C,R=000D,T=2803,P=14,U=300225EE8263EAABC2F78AF007650C04`	Char 0x040C…3002, decl handle=12, value handle=13, perm=0x14
←RX	@E,0045,DR,C=00,H=000F,R=0010,T=2803,P=2A,U=2BE545B07D406B87FBDB8030D6B091CB	Char 0xCB91…2BE5, decl handle=15, value handle=16, perm=0x2A
←**RX**	`@E,0045,DR,C=00,H=0013,R=0014,T=2803,P=28,U=01A10C2000089A9EE21115A133333365`	Char 0x6533…A101, decl handle=19, value handle=20, perm=0x28
←**RX**	`@E,0045,DR,C=00,H=0016,R=0017,T=2803,P=14,U=02A10C2000089A9EE21115A133333365`	Char 0x6533…A102, decl handle=22, value handle=23, perm=0x14
←**RX**	`@E,0045,DR,C=00,H=0019,R=001A,T=2803,P=20,U=03A10C2000089A9EE21115A133333365`	Char 0x6533…A103, decl handle=25, value handle=26, perm=0x20
←**RX**	`@E,0045,DR,C=00,H=001D,R=001E,T=2803,P=28,U=01A20C2000089A9EE21115A133333365`	Char 0x6533…A201, decl handle=29, value handle=30, perm=0x28
←**RX**	`@E,0045,DR,C=00,H=0020,R=0021,T=2803,P=28,U=02A20C2000089A9EE21115A133333365`	Char 0x6533…A202, decl handle=32, value handle=33, perm=0x28
←**RX**	`@E,0010,RPC,C=04,R=060A`	Remote procedure complete, 0x060A = no attributes found in last search request

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`	Request to discover all remote descriptors
←**RX**	`@R,000A,/DRD,0000`	Response indicates success
←**RX**	`@E,0024,DR,C=00,H=0001,R=0000,T=2800,P=00,U=0028`	UUID 0x2800 (Primary Service), start=1
←**RX**	`@E,0024,DR,C=00,H=0002,R=0000,T=2803,P=00,U=0328`	UUID 0x2803 (Characteristic), decl=2
←**RX**	`@E,0024,DR,C=00,H=0003,R=0000,T=0000,P=00,U=002A`	UUID 0x2A00 (Device Name), handle=3
Additional records omitted for brevity		
←**RX**	`@E,0029,DR,C=00,H=001C,R=0000,T=2800,P=00,U=0028`	UUID 0x2800 (Primary Service), start=28
←**RX**	`@E,0029,DR,C=00,H=001D,R=0000,T=2803,P=00,U=0328`	UUID 0x2803 (Characteristic), decl=29
←**RX**	`@E,0045,DR,C=00,H=001E,R=0000,T=0000,P=00,U=01A20C2000089A9EE21115A133333365`	UUID 0x6533…A201 (CYCommand Challenge), handle=30
←**RX**	`@E,0029,DR,C=00,H=001F,R=0000,T=2902,P=00,U=0229`	UUID 0x2902 (CCCD), handle=31

Direction	Content	Effect
←**RX**	`@E,0029,DR,C=00,H=0020,R=0000,T=2803,P=00,U=0328`	UUID 0x2803 (Characteristic), decl=32
←**RX**	`@E,0045,DR,C=00,H=0021,R=0000,T=0000,P=00,U=02A20C2000089A9EE21115A133333365`	UUID 0x6533…A202 (CYCommand Data), handle=33
←**RX**	`@E,0029,DR,C=00,H=0022,R=0000,T=2902,P=00,U=0229`	UUID 0x2902 (CCCD), handle=34
←**RX**	`@E,0010,RPC,C=00,R=060A`	Long remote procedure complete, 0x060A = no attributes found in last search request

3.7.2 How to Read and Write 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 How to Detect 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.

In order 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 Client Characteristic Configuration Descriptor (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 something like 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/custom characteristics, refer to whatever documentation or reference material is made 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 How to Use Peripheral and Central Privacy

GAP privacy randomizes the Bluetooth connection address visible to remote devices in 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 How to Bond With or Without MITM Protection

Bonding between two devices requires first generating and exchanging encryption keys and then permanently storing encryption data along with information required to identify the bonded device and re-use the same keys again in the future. The mechanics of pairing depend on which side (master or slave) 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 so-called “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_set_security_parameters (SSBP, ID=7/11) API command.

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 vs. unauthenticated) and the exact nature of which events and commands must be used on each side. Table 3-13 below describes all possible I/O arrangements and the resulting behavior and authentication level.

Table 3-13. I/O Capabilities and Pairing Behavior

	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)

The information in the above table 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 follow immediately after your local pairing request command. The EZ-Serial peripheral device will then be operating in the "responder" role describe above.

The following table 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 Section 3.8.2.2 (Controlling Automatic Pairing Request Acceptance).

Table 3-14. EZ-Serial API Flow in Common I/O Capability Configurations

		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)

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_security_parameters (SSBP, ID=7/11) 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.

The example below 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→	`SSBP$,F=0`	Clear Bit 0 (auto-accept)
←RX	`@R,000B,SSPB$,0000`	Response indicates success, stored in flash
←RX	`@E,001B,P,C=01,M=01,B=01,K=10,P=00`	Event indicates incoming pairing request
TX→	`/PR,R=0`	Send pairing request response with "0" result (accept)
←RX	`@R,0009,/PR,0000`	Response indicates success
←RX	`@E,001B,B,B=03,A=00A050E3835F,T=00`	Event indicates new bond entry created
←RX	`@E,000F,PR,C=01,R=0000`	Event indicates pairing process 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.

The example below 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,001B,B,B=03,A=00A050421C63,T=00`	Event indicates new bond entry created
←RX	`@E,000F,PR,C=00,R=0000`	Event indicates pairing process 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 which side initiates pairing and which side responds. See Section 3.8.2.1 (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.

The example below 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=00,M=02,B=01,K=10,P=00`	Event indicates incoming pairing request
←RX	`@E,0014,PKD,C=00,P=00017266`	Event indicates passkey display (17266 hex = **094822** dec)
←RX	`@E,001B,B,B=03,A=00A050421C63,T=00`	Event indicates new bond entry created
←RX	`@E,000F,PR,C=00,R=0000`	Event indicates pairing process 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_security_parameters (SSBP, ID=7/11) API command. 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 in order 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_security_parameters (SSBP, ID=7/11) API command. If both of these conditions are not met, then the stack will revert to the default behavior of using a random passkey.

The example below 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=00,M=13,B=01,K=10,P=01`	Event indicates incoming pairing request
←RX	`@E,0014,PKD,C=00,P=0001E240`	Event indicates passkey display (1E240 hex = **123456** dec)
←RX	`@E,001B,B,B=03,A=76C880C3F154,T=01`	Event indicates new bond entry created
←RX	`@E,000F,PR,C=00,R=0000`	Event indicates pairing process completed successfully

3.8.3 How to Use Out-of-Band Pairing

EZ-Serial supports the use of out-of-band (OOB) encryption key sharing for added security during pairing with compatible devices. Use the smp_generate_oob_data (/GOOB, ID=7/7) API command to generate OOB data based on a 16-byte input key. You must use the same key on the remote device to generate matching OOB data in order to successfully pair using out-of-band key exchange.

Ensure that you generate OOB data on both sides of the connection before initiating the pairing process on either side.

NOTE: EZ-Serial will always attempt to use OOB encryption data for pairing if you have set it using the smp_generate_oob_data (/GOOB, ID=7/7) API command. If you set OOB data and then attempt to pair with a device that does not support OOB pairing, or that does not have the correct matching key set, pairing will always fail. To clear OOB data and revert to the standard pairing and key generation/exchange process, either reset the module via hardware or software or use the smp_clear_oob_data (/COOB, ID=7/8) API command.

NOTE: Most smartphones and tablets available at the time of this publication do not support out-of-band pairing for BLE connections. The example shown here works between two Infineon BLE modules running EZ-Serial firmware.

The example below assumes that you have already connected to a remote peer device. An active connection is required for any type of pairing operation.

Example 1: Apply OOB key on two devices and initiate pairing

Device	Direction	Content	Effect
#1	**TX→**	`/GOOB,K=00112233445566778899AABBCCDDEEFF`	Generate new OOB data with a 128-bit key
#1	**←RX**	`@R,000B,/GOOB,0000`	Response indicates success
#2	**TX→**	`/GOOB,K=00112233445566778899AABBCCDDEEFF`	Generate new OOB data with a 128-bit key
#2	**←RX**	`@R,000B,/GOOB,0000`	Response indicates success
#1	**TX→**	`/P,S=1,K=10`	Pair without bonding, security type=1, key size=16
#1	**←RX**	`@R,0008,/P,0000`	Response indicates success

Device	Direction	Content	Effect
#1	**←RX**	`@E,000F,PR,C=00,R=0000`	Event indicates pairing completed successfully
#2	**←RX**	`@E,000F,PR,C=01,R=0000`	Event indicates pairing completed successfully

3.8.4 How to Encrypt and Decrypt Arbitrary Data

The EZ-Serial platform exposes the internal AES encryption engine via two simple API commands to allow encryption and decryption of arbitrary data. Use the system_aes_encrypt (/AESE, ID=2/9) API command to encrypt data, and the system_aes_decrypt (/AESD, ID=2/10) API command to decrypt data.

The encryption and decryption processes require a 16-byte key to initialize the engine, followed by 16 bytes of data to process. You must supply the key for every new operation. The combination of both parts of input data are transmitted in a single argument to the relevant encryption or decryption command:

- Bytes 0-15 = 16-byte Key
- Bytes 16-31 = 16-byte Data to encrypt or decrypt

In the examples below, the text-mode input data blob is broken apart for clarity. However, the actual command requires all data in a single non-broken command.

Example 1: Encrypting 8 bytes of cleartext data

Direction	Content	Effect

TX→	`/AESE,I=`
	`00112233445566778899AABBCCDDEEFF`
	`41424344454647484950515253545556`
		Request encryption of “ABCDEFGHIPQRSTUV” data with simple key

←RX	`@R,002E,/AESE,0000,O=` `48F5051A5DBFAC460601E3665D2D20CB`	Response indicates success, cyphertext returned

Example 2: Decrypting 8 bytes of cyphertext data

Direction	Content	Effect

TX→	`/AESD,I=`
	`00112233445566778899AABBCCDDEEFF`
	`48F5051A5DBFAC460601E3665D2D20CB`
		Request decryption of cyphertext data with input key matching encryption command

←RX	`@R,002E,/AESD,0000,O=41424344454647484950515253545556`	Response indicates success, cleartext returned

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 modification of advertising behavior and does not allow scheduled interleaving with other types of behavior simultaneously.

3.9.1 How to Configure 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:** that 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 How to Configure 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.Infineon.com/" URL at 100 ms interval

Direction	Content	Effect
TX→	`.EDDYSP,E=02,I=00A0,T=10,D=00637970726573307`	Set Eddystone configuration with scheme and encoding
←RX	`@R,000D,.EDDYSP,0000`	Response indicates success

3.10 Performance Testing Examples

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

3.10.1 How to Maximize Throughput to a Remote Peer

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

- **Minimize the connection interval.** The BLE specification allows 7.5 ms minimum 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 reject 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 much 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 to the remote peer on 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 in 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 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 increase 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,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,H=04,I=0006,L=0000,O=0064`	Event indicates new connection parameters accepted

3.10.1.1 How to Maximize 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.

Refer to 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,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,H=04,I=0010,L=0000,O=0064`	Event indicates new connection parameters accepted

3.10.1.2 How to Maximize 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 7.5ms 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 How to Minimize 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 help decrease 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 Section 3.1.5 (How to Manage 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 below may not make a notable difference.

3.10.2.1 How to Minimize 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 time 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 as 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.

- **Don't 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 usec after every advertisement packet to listen for a connection request or scan request packet. When using all three advertising channels, this means three complete TX-RX cycles occur repeatedly at the configured advertisement interval. If a peripheral device only needs to broadcast (e.g. 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 commands. 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 and/or scan response data payload length**. 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 usec per byte. The fixed encapsulation and overhead data in every advertisement or scan response packet takes roughly 140 usec to transmit, but the payload can add up to 248 usec to this duration. In other words, a 31-byte payload (~390 usec) requires twice as much transmission time as a 7-byte payload (~195 usec).

  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 Section 3.4.3 (How to Customize Advertisement and Scan Response Data).

3.10.2.2 How to Minimize 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 re-send 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 over the air 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 How to Communicate Using an L2CAP Channel

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

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256k of flash memory. The behavior described in this section will not function on devices with only 128k of flash.

> **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 Infineon modules with 256k of flash memory running EZ-Serial firmware.

L2CAP uses a credit-based system for managing data flow. Upon connection or at any point afterwards, the receiving end of a data channel grants a certain number of credits to the transmitting side. The transmitting side may send exactly that many packets (regardless of length) before it must wait for additional credits. EZ-Serial provides the following API methods to work with this credit-based system:

- l2cap_send_credits (/LSC, ID=8/5) command for the receiving side to send credits to the transmitting side
- l2cap_rx_credits_low (LRCL, ID=8/5) event on the receiving side when the transmitting side has few or no credits remaining
- l2cap_tx_credits_received (LTCR, ID=8/6) event on the transmitting side when it has received additional credits

The example below assumes that you have already connected the two 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 (Pairing required)

Device	Direction	Content	Effect
#1	**TX→**	`/LRP,N=43,W=0`	Register PSM on channel 43, watermark=0
#1	**←RX**	`@R,000A,/LRP,0000`	Response indicates success
#2	**TX→**	`/LRP,N=73,W=0`	Register PSM on channel 73, watermark=0
#2	**←RX**	`@R,000A,/LRP,0000`	Response indicates success
#1	**TX→**	`/LC,C=0,R=73,L=43,T=17,Z=3`	Open L2CAP connection, 3 TX credits for peer

Device	Direction	Content	Effect
#1	←RX	@R,0009,/LC,0000	Response indicates success
#2	←RX	@E,002C,LCR,C=01,N=0041,L=0073,M=0017,P=0017	Event indicates incoming L2CAP connection
#2	TX→	/LCR,C=0,N=41,R=0,M=17,Z=3	Accept connection, 3 TX credits for peer
	←RX	@R,000A,/LCR,0000	Response indicates success
	←RX	@E,002B,LC,C=01,R=0000,N=0040,M=0017,P=0017, Z=0003	Event indicates connection established
#1	←RX	@E,002B,LC,C=00,R=0000,N=0041,M=0017,P=0017, Z=0003	Event indicates connection request accepted
#1	TX→	/LD,N=41,D=11223344	Send 4-byte data packet to peer
#1	←RX	@R,0009,/LD,0000	Response indicates success
	←RX	@E,0015,LD,N=0040,D=11223344	Event indicates 4-byte data packet received
#1	TX→	/LD,N=41,D=11223344	Send 4-byte data packet to peer
#1	←RX	@R,0009,/LD,0000	Response indicates success
	←RX	@E,0015,LD,N=0040,D=11223344	Event indicates 4-byte data packet received
#1	TX→	/LD,N=41,D=11223344	Send 4-byte data packet to peer
#1	←RX	@R,0009,/LD,0000	Response indicates success
	←RX	@E,0015,LD,N=0040,D=11223344	Event indicates 4-byte data packet received
	←RX	@E,0011,LRCL,C=01,N=0040,Z=0000	Event indicates peer has zero credits remaining
	TX→	/LSC,N=40,Z=3	Send 3 transmit credits to peer
	←RX	@R,000A,LSC,0000	Response indicates success
#1	←RX	@E,0018,LTCR,C=00,N=0041,Z=0003	Event indicates additional credits received

3.11 Device Firmware Update Examples

EZ-Serial provides multiple methods for updating or replacing firmware on the module, as well as the ability to perform a remote update on a compatible target device using the standard Infineon Bootloader GATT profile. These methods are described below. Please refer to Section 2.6.1 (Latest EZ-Serial Firmware Image) for information on where to find the latest EZ-Serial firmware images.

3.11.1 How to Update Firmware Locally Using SWD

If you have access to the local debug interface (**P01**, **P02**), you can use standard Infineon software and programming hardware to flash a new firmware image onto the module.

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 How to Update Firmware Using the DFU Bootloader

Use the dfu_reboot (/CDFU, ID=3/1) API command to reboot into DFU mode for bootloader-based updates over BLE (over the air). This command reboots the firmware into DFU mode, awaiting bootloader protocol communication. Any ongoing activity prior to sending this command will be immediately terminated, including an active BLE connection.

Key aspects of DFU bootloader behavior are:

- Customizations (settings, bonding data, and GATT structure) are retained through the UART DFU process.
- The BLE stack and EZ-Serial application are stored in two separate parts of flash. Updates to each part must be applied separately.
- Updating either the BLE stack or the EZ-Serial application requires that original “application” area of flash to be used for temporary image storage. Therefore, the update process always overwrites the original application.
- Updating the application can be done in a single step, because the old application is replaced with the new one.
- Updating the BLE stack requires that the application be updated after the stack update completes, since the old application will have been overwritten by the temporary stack image.

- If the application update process stops or fails mid-transfer, the stack will remain intact, and a reset will automatically return to the same DFU mode attempted previously.
- If the stack update process stops or fails mid-transfer, the original stack will remain intact, and a reset will return to the same DFU mode attempted previously.
- In UART mode, the DFU bootloader uses 115200 baud, 8/N/1 with no flow control.

The behaviors above apply equally when either UART-based or over-the-air transfer is used for the DFU process.

3.11.2.1 How to Use the DFU Bootloader over UART

To perform a local DFU process over the UART interface, use the dfu_reboot (/CDFU, ID=3/1) with the “mode” argument set to 0 (automatic) or 2 (UART-only). Once this is done, use the standard Infineon bootloader communication protocol over the UART interface to update the EZ-Serial application, or the stack and application during the same session.

Example 1: Enter DFU bootloader in UART mode

Direction	Content	Effect
TX→	`/CDFU$,M=1`	Request reboot into DFU bootloader in UART-only mode
←RX	`@R,000B,/CDFU$,0000`	Response indicates success, reset occurs immediately
←RX	`@E,000A,DFUE,R=1`	System has reset into UART-only DFU bootloader, ready for update

3.11.2.2 How to Use the DFU Bootloader Over the Air (OTA)

To perform a local DFU process over the air, use the dfu_reboot (/RDFU, ID=3/1) with the “mode” argument set to 0 (automatic) or 1 (OTA-only). Once this is done, use the Infineon bootloader communication protocol over the BLE interface to update the EZ-Serial application, or the stack + application during the same session.

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:

1. Module **UART_TX** pin to host UART RX pin
2. Module **UART_RX** pin to host UART TX pin
3. Module **UART_CTS** pin to host UART RTS pin
4. Module **UART_RTS** pin to host UART CTS pin
5. Module **FACTORY_TR**, **CYSPP**, **CP_ROLE**, **LP_MODE**, and **ATEN_SHDN** pins to digital output host GPIOs
6. 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 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 and/or security requirements 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 Chapter 5 (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 as described in Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition) 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:

1. Module **CYSPP** pin to GND (force CYSPP data mode at all times, no API communication)
2. Module **UART_TX** pin to host UART RX pin
3. Module **UART_RX** pin to host UART TX pin
4. Optional for flow control:
    a. Module **UART_CTS** pin to host UART RTS pin
    b. Module **UART_RTS** pin to host UART CTS pin
5. Optional for connectivity status:
    a. 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 and/or security requirements 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 may need or wish to make these changes over CYCommand mode, as described in Section 3.3 (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. So,.

If the host supports a simple "enable" control line for whether or not 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 may need or wish to apply this initial configuration over CYCommand mode, as described in Section 3.3 (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/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 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 (e.g. 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

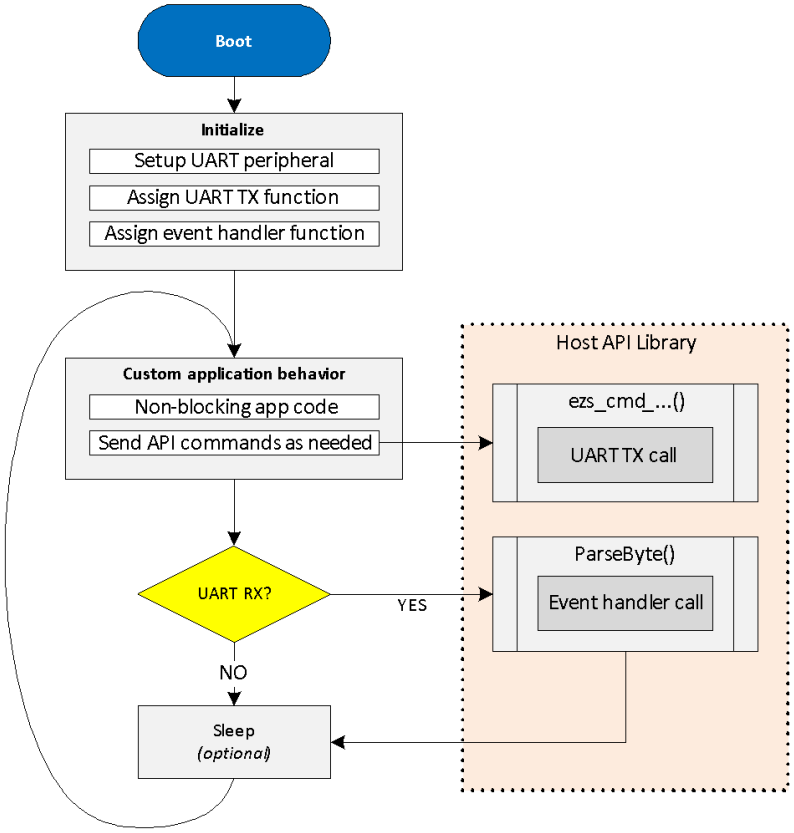
5.2.1 Basic Application Architecture

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

This process is shown in the following flowchart:

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 following methods:

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**)

Function	Description
`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)

The application is responsible for providing implementation functions for three methods, assigned to the function pointers below:

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 will be 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 will be manually calling the **EZSerial_Parse** method after reading bytes in over UART, this may be assigned as NULL.

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 how many arguments are needed for any given command and what their data types are.

This macro-based approach means it is not possible for to perform type checking at compile time, but it also means that the entire command generator implementation uses a tiny quantity of flash memory (well under one kByte 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**

Function	Description
`EZS_CHECK_FOR_PACKET()`	Wrapper for **EZS_WAIT_FOR_PACKET(0)**, a non-blocking attempt to read data

The assignable “return value” (evaluated expression result) for all of 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, etc.—then the pointer value will be 0 (NULL).

5.3 Porting the 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 in order 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 (e.g. “gap”)
  - `commands` (list of dictionaries) […
    - `id` – Numeric ID assigned to command
    - `name` – Alpha name assigned to command (e.g. “start_adv”)
    - `flashopt` – Boolean flag indicating flash storage for settings
    - `parameters` (list of dictionaries) […
      - `type` – Data type (e.g. “uint16”)
      - `name` – Alpha name assigned to parameter (e.g. “mode”)
      - `textname` – text-mode equivalent (e.g. “M”)
      - `required` – Boolean flag indicating optional or required parameter
      - `format` – Intended data presentation format (e.g. “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 instructions below can help narrow down the cause of failure in 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 **VDD**, **VDDR**, and **GND** pins are properly connected (**VDDR** also requires power)
2. Ensure **VDD** and **VDDR** have a stable supply within the supported range (typically 3 V – 5 V)
3. Ensure 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 the UART flow control pins are properly connected:
    a. Module **UART_RTS** to host CTS
    b. Module **UART_CTS** to host RTS
5. Ensure 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 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. See Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition) for more detail.
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 Section 7.1 (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> [data0, data1, ..., dataN]` 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 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 the remote device is not too far away or in any other situation resulting in very low signal strength. Scanning as described in (a) will also reveal this with observation of scan result RSSI values.
    d. Ensure you have specified the correct Bluetooth connection (MAC) address *and* address type (public or private). A connection attempt with the right Bluetooth address but the wrong address type will fail.
    e. Ensure 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 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 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 Section 3.4.3 (How to Customize Advertisement and Scan Response Data) for detail.

6.3 GPIO Signal Issues

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

1. Ensure that the pins you have connected are correct based on your chosen module. See Section 8.1 (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
3. gpio_set_function (SIOF, ID=9/3) API command. Note that all functions are enabled in the factory default configuration and should not need to be re-enabled in order to work out of the box.
4. 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
5. 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.

The material in this revision of the User Guide describes version 1.3 of the API protocol.

7.1 Protocol Structure and Communication Flow

7.1.1 API Protocol Formats

EZ-Serial implements a unified set of functionality that can be accessed using either text or binary API communication. These two formats cover the same feature set, and do not offer more or less control in any way (with the exception of optional argument support in text mode, described below).

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 shall end with “\r\n” characters (0x0D, 0x0A). Commands sent to the module may end with either or both. Unlike the binary mode described below, 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 below 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 Section 7.1.2 (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 below, 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.

Table 7-1. API Protocol Data Types

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.	

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 Infineon EZ-BLE 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

The binary packet 4-byte header structure is described in the table below:

Table 7-2. Binary Packet 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), and 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 0LLL` 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 atypically long packets shall be documented specifically.
2	Group ID	All protocol methods are organized into logically separate groups, such as GAP, GATT server, L2CAP, CYSPP, etc. 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.

7.2 API Commands and Responses

All commands and responses implemented in the API protocol are described in detail below. API events are documented separately in Section 7.3). A master list of all possible error codes resulting from commands can be found in Section 7.4 (Error Codes).

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, due to the fact that 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

Refer to Section 7.4 (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)
- DFU 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 in Section 7.3.1, 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 Section 7.1 (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 info, 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_unique_id (/QUID, ID=2/7)
- system_query_random_number (/QRND, ID=2/8)
- system_aes_encrypt (/AESE, ID=2/9)
- system_aes_decrypt (/AESD, ID=2/10)
- 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_eco_parameters (SECO, ID=2/15)
- system_get_eco_parameters (GECO, ID=2/16)
- system_set_wco_parameters (SWCO, ID=2/17)
- system_get_wco_parameters (GWCO, ID=2/18)
- 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)
- system_force_hibernation (/SLEEP, ID=2/27)

Events within this group are documented in Section 7.3.2, 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 Bluetooth Low Energy 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. Refer to Section 3.1.5 (How to Manage 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
uint32	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). Refer to Section 2.5.2 (Saving Runtime Settings in Flash) for details about 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) – Use to store 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 Infineon 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

Response Parameters:

Data Type	Name	Text	Description
uint16	length	L	● Number of bytes to be dumped

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)

Store all configuration settings into flash.

This command applies all runtime settings into the boot-level configuration area stored in non-volatile flash. Refer to Section 2.5 (Configuration Settings, Storage, and Protection) for details about 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)

Reset 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. Refer to Section 2.2 (Factory Default Behavior) for details about 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:

- Section 3.1.6.2 (Factory Reset via API Command)

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

Query EZ-Serial firmware version info.

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: • 0x01 = CYBLE-01201X-X0 • 0x02 = CYBLE-014008-00 • 0x03 = CYBLE-022001-00 • 0x04 = CYBLE-2X20XX-X1 • 0x05 = CYBLE-2120XX-X0 • 0x06 = CYBLE-212020-01 • 0x07 = CYBLE-214009-00 • 0x08 = CYBLE-214015-01 • 0x09 = CYBLE-222005-00 • 0x0A = CYBLE-222014-01 • 0x0B = CYBLE-224110-00 • 0x0C = CYBLE-224116-01

Data Type	Name	Text	Description
			• 0X40 = CYW20822-P4TAI040_P4EPI040

Related Events:

- system_boot (BOOT, ID=2/1)

7.2.2.7 system_query_unique_id (/QUID, ID=2/7)

Query EZ-Serial module unique identifier. This command is not implemented.

The module's unique identifier comes from factory-stored data in the chipset's supervisory flash (SFLASH) area. The four bytes returned are:

1. Die X position
2. Die Y position
3. Die wafer number
4. Die lot number

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	02	07	None.
RSP	C0	07	02	07	None.

Text Info:

Text Name	Response Length	Category	Notes
/QUID	0x0016	ACTION	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8a	id	U	Unique ID (1 length byte equal to 0x04, followed by 4 data bytes) **NOTE:** uint8a data type requires one prefixed "length" byte before binary parameter payload

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

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

This command provides simple access to the random number generator in the EZ-BLE module's chipset. 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.
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

7.2.2.9 system_aes_encrypt (/AESE, ID=2/9)

Generate AES-encrypted cyphertext using provided key, initialization info, and cleartext.

This command provides access to the internal hardware AES engine inside the EZ-BLE module's chipset. The encryption process takes a 16-byte key to initialize the engine, and can encrypt 16 bytes at a time. Encrypted data may be decrypted with the system_aes_decrypt (/AESD, ID=2/10) API command, using the same key and nonce.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	1D	02	09	None.
RSP	C0	12	02	09	None.

Text Info:

Text Name	Response Length	Category	Notes
/AESE	0x2E	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8a	in_struct	I*	Input structure (32 bytes): • Bytes 0-15 = 16-byte Key • Bytes 16-31 = Clear text **NOTE:** uint8a data type requires one prefixed "length" byte before binary parameter payload

Response Parameters:

Data Type	Name	Text	Description
uint8a	out	O	Cyphertext output (16 bytes) **NOTE:** uint8a data type requires one prefixed "length" byte before binary parameter payload

Related Commands:

- system_aes_decrypt (/AESD, ID=2/10)

Example Usage:

- Section 0 (
- How to Encrypt and Decrypt Arbitrary Data)

7.2.2.10 system_aes_decrypt (/AESD, ID=2/10)

Generate AES-decrypted plaintext using provided key, initialization info, and cyphertext.

This command provides access to the internal hardware AES engine inside the EZ-BLE module's chipset. The decryption process takes a 16-byte key and 13-byte nonce to initialize the engine, and can decrypt up to 27 bytes at a time. Cleartext data may be encrypted with the system_aes_encrypt (/AESE, ID=2/9) API command, and later decrypted using this API command with the same key and nonce.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	1D	02	0A	None.
RSP	C0	12	02	0A	None.

Text Info:

Text Name	Response Length	Category	Notes
/AESD	0x2E	ACTION	

Command Arguments:

Data Type	Name	Text	Description
uint8a	in_struct	I*	Input structure (32 bytes): • Bytes 0-15 = 16-byte Key • Bytes 16-31 = Cyphertext data to be decrypted. **NOTE:** uint8a data type requires one prefixed “length” byte before binary parameter payload

Response Parameters:

Data Type	Name	Text	Description
uint8a	out	O	Cleartext output (16 bytes) **NOTE:** uint8a data type requires one prefixed “length” byte before binary parameter payload

Related Commands:

- system_aes_encrypt (/AESE, ID=2/9)

Example Usage:

- Section 0 (
- How to Encrypt and Decrypt Arbitrary Data)

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

Write arbitrary data to the user flash storage area.

EZ-serial provides 256 bytes of non-volatile flash storage for application data. This command allows writing 1-32 bytes to any position within this 256-byte area.

> **NOTE:** You must specify a data offset and length which do not exceed 256 when combined. For example, if you are writing 32 bytes of data, the specified “offset” argument must be 224 (0xE0) or less.

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)

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

Read arbitrary data from the user flash storage area.

EZ-serial provides 256 bytes of non-volatile flash storage for application data. This command allows reading 1-32 bytes from any position within this 256-byte area.

> **NOTE:** You must specify a data offset and length which do not exceed 256 when combined. For example, if you are reading 32 bytes of data, the specified "offset" argument must be 224 (0xE0) or less.

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)

7.2.2.13 system_set_bluetooth_address (SBA, ID=2/13)

Configure a new Bluetooth address.

This address will be visible to remote scanning or connected devices, as long as the module is not operating with privacy enabled. EZ-Serial uses a fixed public or static random address by default, which is generated dynamically based on unique properties of the chipset inside each module (including wafer/die data). Normally, you do not need to change the Bluetooth address using 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.
>
> **NOTE:** EZ-serial received this command, it will return an error code 0x0111 in response package. It means that you need reset the device to make the address avaliable.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	06	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 Bluetooth address. Set all six 0x00 bytes to revert to factory-provided 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:

- Section 3.8.1 (How to Use Peripheral and Central Privacy)

7.2.2.14 system_get_bluetooth_address (GBA, ID=2/14)

Obtain the current Bluetooth address.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	02	0E	None.
RSP	C0	08	02	0E	None.

Text Info:

Text Name	Response Length	Category	Notes
GBA	0x0018	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
macaddr	address	A	Current Bluetooth 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)

7.2.2.15 system_set_eco_parameters (SECO, ID=2/15)

Configure a new External Clock Oscillator (ECO) trim value. This command is not implemented.

> **WARNING:** You should not need to modify this value under normal circumstances. ECO trim values are set within tolerance from the factory on all EZ-BLE modules during manufacturing.

Binary Header:

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

Text Info:

Text Name	Response Length	Category	Notes
SECO	0x000A	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint16	trim	T	New ECO trim value. Set to 0x0000 to clear any custom setting and revert to factory defaults.

Response Parameters:

None.

Related Commands:

- system_get_eco_parameters (GECO, ID=2/16)
- system_set_wco_parameters (SWCO, ID=2/17)
- system_get_wco_parameters (GWCO, ID=2/18)

7.2.2.16 system_get_eco_parameters (GECO, ID=2/16)

Obtain the current External Clock Oscillator (ECO) trim value.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	02	10	None.
RSP	C0	04	02	10	None.

Text Info:

Text Name	Response Length	Category	Notes
GECO	0x0011	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint16	trim	T	Current ECO trim value

Related Commands:

- system_set_eco_parameters (SECO, ID=2/15)
- system_set_wco_parameters (SWCO, ID=2/17)
- system_get_wco_parameters (GWCO, ID=2/18)

7.2.2.17 system_set_wco_parameters (SWCO, ID=2/17)

Configure a new Watch Clock Oscillator (WCO) accuracy value.

WARNING: You should not need to modify this value under normal circumstances. WCO accuracy values are set from the factory based on the hardware design of each EZ-BLE module.

NOTE: EZ-serial received this command, it will return an error code 0x0111 in response package. It means that you need reset the device to make the settings avaliable.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01	02	11	None.
RSP	C0	02	02	11	None.

Text Info:

Text Name	Response Length	Category	Notes
SWCO	0x000A	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	accuracy	A	New WCO accuracy value: • 0 = 251-500 ppm • 1 = 151-250 ppm • 2 = 101-150 ppm • 3 = 76-100 ppm • 4 = 51-75 ppm • 5 = 31-50 ppm • 6 = 21-30 ppm • 7 = 0-20 ppm

Response Parameters:

None.

Related Commands:

- system_set_eco_parameters (SECO, ID=2/15)
- system_get_eco_parameters (GECO, ID=2/16)
- system_get_wco_parameters (GWCO, ID=2/18)

7.2.2.18 system_get_wco_parameters (GWCO, ID=2/18)

Obtain the current Watch Clock Oscillator (WCO) accuracy value.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	02	12	None.
RSP	C0	03	02	12	None.

Text Info:

Text Name	Response Length	Category	Notes
GWCO	0x000F	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8	accuracy	A	Current WCO accuracy value: • 0 = 251-500 ppm • 1 = 151-250 ppm • 2 = 101-150 ppm • 3 = 76-100 ppm • 4 = 51-75 ppm • 5 = 31-50 ppm • 6 = 21-30 ppm • 7 = 0-20 ppm

Related Commands:

- system_set_eco_parameters (SECO, ID=2/15)
- system_get_eco_parameters (GECO, ID=2/16)
- system_set_wco_parameters (SWCO, ID=2/17)

7.2.2.19 system_set_sleep_parameters (SSLP, ID=2/19)

Configure new system-wide sleep settings.

EZ-Serial automatically enters the most low-power sleep mode available in order 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. Refer to Section 3.1.5 (How to Manage Sleep States) for detail.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01	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 • 1 = Normal sleep when possible (factory default) • 2 = Deep sleep when possible

Response Parameters:

None.

Related Commands:

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

Example Usage:

- Section 3.1.5.1 (Configuring the System-Wide Sleep Level)

7.2.2.20 system_get_sleep_parameters (GSLP, ID=2/20)

Obtain the current system-wide sleep settings. This command is not implemented.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	02	14	None.
RSP	C0	03	02	14	None.

Text Info:

Text Name	Response Length	Category	Notes
GSLP	0x000F	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8	level	L	Current maximum system-wide sleep level: • 0 = Sleep disabled • 1 = Normal sleep when possible (factory default)

Data Type	Name	Text	Description
			• 2 = Deep sleep when possible

Related Commands:

- system_set_sleep_parameters (SSLP, ID=2/19)

7.2.2.21 system_set_tx_power (STXP, ID=2/21)

Configure 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 take effect immediately, as soon as 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	New transmit power: • 1 = -20 dBm • 2 = -10 dBm • 3 = -6 dBm • 4 = -4 dBm • 5 = -2 dBm • 6 = 0 dBm (factory default) • 7 = +2 dBm • 8 = +4 dBm

Response Parameters:

None.

Related Commands:

- system_get_tx_power (GTXP, ID=2/22)

7.2.2.22 system_get_tx_power (GTXP, ID=2/22)

Obtain 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 transmit power: • 1 = -20 dBm

Data Type	Name	Text	Description
			• 2 = -10 dBm (default/maximum for CYBLE-2X20XX-X1) • 3 = -6 dBm (default/maximum for CYBLE-224110-00 and CYBLE-224116-01) • 4 = -4 dBm • 5 = -2 dBm • 6 = 0 dBm (factory default) • 7 = +2 dBm • 8 = +4 dBm

Related Commands:

- system_get_tx_power (GTXP, ID=2/22)

7.2.2.23 system_set_transport (ST, ID=2/23)

Configure 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: • 0 = reserved • 1 = UART (factory default)

Response Parameters:

None.

Related Commands:

- system_get_transport (GT, ID=2/24)
- system_set_uart_parameters (STU, ID=2/25)

7.2.2.24 system_get_transport (GT, ID=2/24)

Obtain 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: • 0 = reserved • 1 = UART (factory default)

Related Commands:

- system_set_transport (ST, ID=2/23)
- system_get_uart_parameters (GTU, ID=2/26)

7.2.2.25 system_set_uart_parameters (STU, ID=2/25)

Configure 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. In order to store new settings in non-volatile memory, you must send the command once without the flash storage bit/flag, and then re-send the same command again 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, refer to Section 2.5.3 (Protected Configuration Settings).

> **WARNING:** If you have deep sleep enabled using the system_set_sleep_parameters (SSLP, ID=2/19) API command and you are relying on UART data reception to wake the module 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 or not. For detail, refer to Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition).

> **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, but may be worked around by one 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 (Recommand baudrates: 300,9600,19200,115200,230400,460800,1000000): ● Minimum = 300 baud (0x12C) ● Factory default = 115,200 baud (0x1C200) ● Maximum = 1,000,000 baud (0xf4240)
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 in current version) ● 9 = 9 data bits
uint8	parity	P	UART parity: ● 0 = Disabled (factory default, must always be disabled in current version) ● 1 = Odd parity ● 2 = Even parity
uint8	stopbits	S	UART stop bits: ● 1 = 1 stop bit (factory default, must always be 1 in current version) ● 2 = 1.5 stop bits ● 3 = 2 stop bits ● 4 = 2.5 stop bits ● 5 = 3 stop bits ● 6 = 3.5 stop bits ● 7 = 4 stop bits

Response Parameters:

None.

Related Commands:

- system_set_transport (ST, ID=2/23)
- system_get_uart_parameters (GTU, ID=2/26)

Example Usage:

- Section 3.1.2 (How to Change the Serial Communication Parameters)
- Section 3.1.5.5 (Avoiding UART Data Loss or Corruption due to Deep Sleep Transition)

7.2.2.26 system_get_uart_parameters (GTU, ID=2/26)

Obtain 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 in current version) ● 9 = 9 data bits
uint8	parity	P	UART parity: ● 0 = Disabled (factory default, must always be disabled in current version) ● 1 = Odd parity ● 2 = Even parity
uint8	stopbits	S	UART stop bits: ● 1 = 1 stop bit (factory default, must always be 1 in current version) ● 2 = 1.5 stop bits ● 3 = 2 stop bits ● 4 = 2.5 stop bits ● 5 = 3 stop bits ● 6 = 3.5 stop bits ● 7 = 4 stop bits

Related Commands:

- system_get_transport (GT, ID=2/24)
- system_set_uart_parameters (STU, ID=2/25)

7.2.2.27 system_force_hibernation (/SLEEP, ID=2/29)

Forced hibernation mode

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	04	02	1D	None.
RSP	C0	02	02	1D	None.

Text Info:

Text Name	Response Length	Category	Notes
/SLEEP	0x000C	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint32	timeout	T	Wake up timeout in ms ● 0 = disabled (default)

Response Parameters:

None.

Related Commands:

None.

7.2.3 DFU Group (ID=3)

DFU methods relate to the firmware update process, using wired UART transfer.

Commands within the DFU group are listed below:

- dfu_reboot (/CDFU, ID=3/1)

Events within this group are documented in Section 7.3.3, DFU Group (ID=3).

7.2.3.1 dfu_reboot (/CDFU, ID=3/1)

Reboot into DFU mode.

> **NOTE:** There must be a $ followed this command.

This command reboots into the bootloader environment, to begin a local or remote device firmware update (DFU) procedure. Using this command will immediately stop any current activity, and any configuration settings not stored in flash will be lost.

The bootloader will automatically reboot back into the EZ-Serial application after 60 seconds if you do not start the bootloading process within that time. Refer to Section 3.11.2 (How to Update Firmware Using the DFU Bootloader) for details concerning DFU operation.

Binary Header:

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

Text Info:

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

Command Arguments:

Data Type	Name	Text	Description
uint8	mode	M*	DFU boot mode: ● 1 = Allow only UART bootloading

Response Parameters:

None.

Related Events:

- dfu_boot (DFUE, ID=3/1)

Example Usage:

- Section 3.11.2 (How to Update Firmware Using the DFU Bootloader)

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_send_connupdate_response (/CUR, ID=4/4)
- 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_set_adv_legacy_coded_phy_parameters (SACP, ID=4/29)
- gap_get_adv_legacy_coded_phy_parameters (GACP, ID=4/30)
- gap_start_legacy_coded_adv(/CA, ID=4/31)
- gap_stop_legacy_coded_adv(/CAX, ID=4/32)
- set_scan_legacy_coded_parameters (SSCP, ID=4/33)
- get_scan_legacy_coded_parameters (GSCP, ID=4/34)
- gap_start_legacy_coded_scan (/CS, ID=4/35)
- gap_stop_legacy_coded_scan (/CSX, ID=4/36)
- gap_phy_update (/UP, ID=4/37)
- gap_set_extended_adv_data (SEAD, ID=4/38)
- gap_get_extended_adv_data (GEAD, ID=4/39)
- gap_set_extended_scan_response_data(SERD, ID=4/40)
- gap_get_extended_scan_response_data(GERD, ID=4/41)

Events within this group are documented in Section 7.3.4, GAP Group (ID=4).

7.2.4.1 gap_connect (/C, ID=4/1)

Initiate a connection to a remote device.

In order 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 connected already. 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, or until the connection attempt is canceled with the gap_cancel_connection (/CX, ID=4/2) API command, or the connection scan timeout period expires (if it has been set).

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 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:

Data Type	Name	Text	Description
uint8	conn_handle	C	Handle assigned to new pending connection **(always 0 in current release due to internal BLE stack functionality, final non-zero connection handle will be present in connection event occurring after the connection is established)**

Related Commands:

- gap_connect (/C, ID=4/1)
- gap_disconnect (/DIS, ID=4/5)

Related Events:

- gap_connected (C, ID=4/5) – Occurs when an outgoing connection attempt succeeds

Example Usage:

- Section 3.5.3 (How to Connect to a Peripheral Device)

7.2.4.2 gap_cancel_connection (/CX, ID=4/2)

Cancel 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 only applies 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:

- Section 3.5.4 (How to Cancel a Pending Connection to a Peripheral Device)

7.2.4.3 gap_update_conn_parameters (/UCP, ID=4/3)

Request 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 changes from its previous value.

The behavior following this command depends on the link-layer role (master or slave) of the device which initiated the request. The master device has final authority over connection parameters.

If used while in the **master** role (connection to peer initiated locally):

- 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

If used while in the **slave** role (connection from peer initiated remotely):

- New connection parameters must be confirmed by the master
- Remote peer (master) will generate gap_connection_update_requested (UCR, ID=4/7) event if running EZ-Serial
- Remote peer (master) must use gap_send_connupdate_response (/CUR, ID=4/4) command if running EZ-Serial
- Local device will generate gap_connection_updated (CU, ID=4/8) event if master accepts parameters

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 **(Ignored in current release due to internal BLE stack functionality, set to 0)**
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)
- gap_send_connupdate_response (/CUR, ID=4/4)

Related Events:

- gap_connection_update_requested (UCR, ID=4/7)
- gap_connection_updated (CU, ID=4/8)

7.2.4.4 gap_send_connupdate_response (/CUR, ID=4/4)

Accept or rejects a connection update request.

Use this command after receiving the gap_connection_update_requested (UCR, ID=4/7) API event, which indicates that a connected slave has requested a connection parameter update.

Binary Header:

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

Text Info:

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

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Handle of connection for which to send response **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint8	response	R*	Response: • 0 = Accept (new parameters will be applied) • 1 = Reject (new parameters will not be applied)

Response Parameters:

None.

Related Commands:

- gap_update_conn_parameters (/UCP, ID=4/3)

Related Events:

- gap_connection_update_requested (UCR, ID=4/7)

7.2.4.5 gap_disconnect (/DIS, ID=4/5)

Close an open connection to a remote device.

Use this command to cleanly close an established connection with a remote peer device. The connection must first 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 disconnect **(Ignored in current release due to internal BLE stack functionality, set to 0)**

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)

7.2.4.6 gap_add_whitelist_entry (/WLA, ID=4/6)

Add a new Bluetooth address to the whitelist.

The whitelist is an optional filter for determining which remote peers are allowed to connect, or which the local module may try to connect to. When whitelist filtering is active, any devices which are not on the whitelist will not be allowed to connect with the module. You can control 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 you may not need any manual additions or removals from the whitelist.

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

7.2.4.7 gap_delete_whitelist_entry (/WLD, ID=4/7)

Remove 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, refer to documentation for 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)

7.2.4.8 gap_start_adv (/A, ID=4/8)

Start legacy 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 in order 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.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. Shorter intervals 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	mode	M	Discovery mode: ● 0 = Non-discoverable/broadcast-only ● 1 = General discovery
uint8	type	T	Advertisement type: ● 0 = Connectable, undirected ● 1 = Connectable, directed ● 2 = Scannable, undirected ● 3 = Non-connectable, undirected
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**)
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
uint8	filter	F	Advertisement filter policy: ● 0 = Scan request and connect request from any ● 1 = Scan request whitelist-only, connect request from any ● 2 = Scan request from any, connect request whitelist-only ● 3 = Scan request and connect request whitelist-only
uint16	timeout	O	Advertisement timeout (seconds): ● 0 to disable

Data Type	Name	Text	Description
macaddr	directAddr	A	Directed advertisement address
uint8	directAddr Type	Y	Directed address type(if using directed advertisement mode): • 0: BLE_ADDR_PUBLIC • 1: BLE_ADDR_RANDOM

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:

- Section 3.4.1 (How to Advertise as Peripheral Device)

7.2.4.9 gap_stop_adv (/AX, ID=4/9)

Stop advertising.

This command immediately stops advertising if it is currently active. Note that advertising may have started as a result of the gap_start_adv (/A, ID=4/8) or gap_start_legacy_coded_adv (/CA, ID=4/31) API command, or due to specific configuration settings (GAP parameters, CYSPP profile, iBeacon, or Eddystone) that automatically begin 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)

7.2.4.10 gap_start_scan (/S, ID=4/10)

Start 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 in order for this command to succeed. However, it is possible to advertise and scan simultaneously.

EZ-Serial will generate the gap_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 = Observation mode • 1 = General discovery mode (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**)
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
uint8	active	A	Active scanning: • 0 = Passive scanning (default) • 1 = Active scanning
uint8	filter	F	Whitelist filter policy: • 0 = Accept all advertising packets (default) • 1 = Accept only from whitelisted devices • 2 = Accept only from devices sending directed advertisements to this device (not support) • 3 = Accept only from whitelisted devices sending directed advertisements to this device.(not support)
uint8	nodupe	D	Duplicate filter policy: • 0 = Disable duplicate result filtering (default) • 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): • 0 to disable. Only discovery mode (0 = Observation mode) supports continuous scanning (timeout = 0). • 0x0A (default)

Response Parameters:

None.

Related Commands:

- gap_stop_scan (/SX, ID=4/11)
- gap_set_scan_parameters (SSP, ID=4/25)

Related Events:

- gap_scan_state_changed (SSC, ID=4/3)
- gap_scan_result (S, ID=4/4)

7.2.4.11 gap_stop_scan (/SX, ID=4/11)

Stop scanning.

This command immediately stops scanning if it is currently active. Note that advertising may have started as a result of the gap_start_scan (/S, ID=4/10) or gap_start_legacy_coded_scan (/CS, ID=4/35) API command, or due to specific configuration settings (particularly the CYSPP profile settings if the central role is enabled).

EZ-Serial will generate the gap_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:

- gap_scan_state_changed (SSC, ID=4/3)

7.2.4.12 gap_query_peer_address (/QPA, ID=4/12)

Query remote peer Bluetooth address.

This command provides returns the Bluetooth address of the currently connected remote peer device. An active connection is required in order 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	Handle of connection for which to query remote peer address **(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)

7.2.4.13 gap_query_rssi (/QSS, ID=4/13)

This command provides returns the remote signal strength indication (RSSI) value detected in the packet received most recently from the currently connected remote peer device. An active connection is required in order 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**
- 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	Handle of connection for which to query signal strength **(Ignored in current release due to internal BLE stack functionality, set to 0)**

Response Parameters:

Data Type	Name	Text	Description
int8	rssi	R	RSSI value in dBm (between -85 and +5), or 0 if used while not connected

Related Commands:

- gap_query_peer_address (/QPA, ID=4/12)

7.2.4.14 gap_query_whitelist (/QWL, ID=4/14)

Request 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)

7.2.4.15 gap_set_device_name (SDN, ID=4/15)

Configure 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)

Response Parameters:

None.

Related Commands:

- gap_get_device_name (GDN, ID=4/16)

Example Usage:

- 3.1.3 (How to Change the Device Name and Appearance)

7.2.4.16 gap_get_device_name (GDN, ID=4/16)

Obtain 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)

7.2.4.17 gap_set_device_appearance (SDA, ID=4/17)

Configure a new device name.

Define 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)

7.2.4.18 gap_get_device_appearance (GDA, ID=4/18)

Obtain 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)

7.2.4.19 gap_set_adv_data (SAD, ID=4/19)

Configure new custom advertisement packet data.

Define 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 EZ-BLE 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:

- Section 3.4.3 (How to Customize Advertisement and Scan Response Data)

7.2.4.20 gap_get_adv_data (GAD, ID=4/20)

Obtain 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)

7.2.4.21 gap_set_sr_data (SSRD, ID=4/21)

Configure 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 EZ-BLE 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:

- Section 3.4.3 (How to Customize Advertisement and Scan Response Data)

7.2.4.22 gap_get_sr_data (GSRD, ID=4/22)

Obtain 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)

7.2.4.23 gap_set_adv_parameters (SAP, ID=4/23)

Configure 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.

The parameters are synchronized with gap_set_adv_legacy_coded_phy_parameters (SACP, ID=4/29) API command.

> **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 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	mode	M	Discovery mode: ● 0 = Non-discoverable/broadcast-only ● 1 = General discovery (factory default)
uint8	type	T	Advertisement type: ● 0 = Connectable, undirected (factory default) ● 1 = Connectable, directed ● 2 = Scannable, undirected ● 3 = Non-connectable, undirected
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**)

Data Type	Name	Text	Description
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)
uint8	filter	L	Advertisement filter policy: ● 0 = Scan request and connect request from any (factory default) ● 1 = Scan request whitelist-only, connect request from any ● 2 = Scan request from any, connect request whitelist-only ● 3 = Scan request and connect request whitelist-only
uint16	timeout	O	Advertisement timeout (seconds): ● 0 to disable (factory default)
uint8	flags	F	Advertisement behavior flags bitmask: ● Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection ● Bit 1 (0x2) = Use custom advertisement and scan response data ● **NOTE:** Factory default = 0x00 (no bits set)
macaddr	directAddr	A	Directed advertisement address
uint8	directAddrType	Y	Directed address type(if using directed advertisement mode): ● 0: BLE_ADDR_PUBLIC ● 1: BLE_ADDR_RANDOM

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_get_adv_parameters (GAP, ID=4/24)

7.2.4.24 gap_get_adv_parameters (GAP, ID=4/24)

Obtain 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	mode	M	Discovery mode: ● 0 = Non-discoverable/broadcast-only ● 1 = General discovery (factory default)
uint8	type	T	Advertisement type: (0-3 for legacy adv, 4-5 for periodic adv, 6-A for extended adv) ● 0x00 = Legacy: Connectable, undirected (factory default) ● 0x01 = Legacy:Connectable, directed ● 0x02 = Legacy:Scannable, undirected ● 0x03 = Legacy:Non-connectable, undirected ● 0x04 = Periodic: Undirected ● 0x05 = Periodic: Directed ● 0x06 = Extended: Undirected connectable

Data Type	Name	Text	Description
			• 0x07 = Extended: Directed connectable • 0x08 = Extended: Non-connectable, non-scannable • 0x09 = Extended: Non-connectable, scannable • 0x0A = Extended: Non-connectable anonymous directed
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**)
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)
uint8	filter	L	Advertisement filter policy: • 0 = Scan request and connect request from any (factory default) • 1 = Scan request whitelist-only, connect request from any • 2 = Scan request from any, connect request whitelist-only • 3 = Scan request and connect request whitelist-only
uint16	timeout	O	Advertisement timeout (seconds): • 0 to disable (factory default)
uint8	flags	F	Advertisement behavior flags bitmask: • Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection • Bit 1 (0x2) = Use custom advertisement and scan response data • **NOTE:** Factory default = 0x00 (no bits set)
macaddr	directAddr	A	Directed advertisement address
uint8	directAddr Type	Y	Directed address type(if using directed advertisement mode): • 0: BLE_ADDR_PUBLIC • 1: BLE_ADDR_RANDOM

Related Commands:

- gap_set_adv_parameters (SAP, ID=4/23)

7.2.4.25 gap_set_scan_parameters (SSP, ID=4/25)

Configure 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 = General discovery mode (factory default)
uint16	interval	I	Scan interval (625 µs units):

Data Type	Name	Text	Description
			• 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	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
uint8	active	A	Active scanning: • 0 = Passive scanning (factory default) • 1 = Active scanning
uint8	filter	F	Whitelist filter policy: • 0 = Accept all advertising packets (factory default) • 1 = Accept only from whitelisted devices • 2 = Accept only from devices sending directed advertisements to this device (not support) • 3 = Accept only from whitelisted devices sending directed advertisements to this device (not support) •
uint8	nodupe	D	Duplicate filter policy: • 0 = Disable duplicate result filtering (factory default) • 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): • 0x0A to disable (factory default)

Response Parameters:

None.

Related Commands:

- gap_start_scan (/S, ID=4/10)
- gap_get_scan_parameters (GSP, ID=4/26)

7.2.4.26 gap_get_scan_parameters (GSP, ID=4/26)

Obtain 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 = General discovery mode (factory default)

Data Type	Name	Text	Description
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**)
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`
uint8	active	A	Active scanning: ● 0 = Passive scanning (factory default) ● 1 = Active scanning
uint8	filter	F	Whitelist filter policy: ● 0 = Accept all advertising packets (factory default) ● 1 = Accept only from whitelisted devices ● 2 = Accept only from devices sending directed advertisements to this device ● 3 = Accept only from whitelisted devices sending directed advertisements to this device
uint8	nodupe	D	Duplicate filter policy: ● 0 = Disable duplicate result filtering (factory default) ● 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): ● 0x0A to disable (factory default)

Related Commands:

- gap_set_scan_parameters (SSP, ID=4/25)

7.2.4.27 gap_set_conn_parameters (SCP, ID=4/27)

Configure 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.

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**)
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 (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)

7.2.4.28 gap_get_conn_parameters (GCP, ID=4/28)

Used to get 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**)
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 (factory default)

Related Commands:

- gap_set_conn_parameters (SCP, ID=4/27)

7.2.4.29 gap_set_adv_legacy_coded_phy_parameters (SACP, ID=4/29)

Configure new default advertisement parameters for legacy, extended or periodic advertisement.

These parameters will be used when sending the gap_start_adv (/A, ID=4/8) or gap_start_legacy_coded_adv (/CA, ID=4/31) API commands in text mode without specifying non-default arguments.

The parameters are synchronized with gap_get_adv_parameters (GAP, ID=4/24) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	1D	None.
RSP	C0		04	1D	None.

Text Info:

Text Name	Response Length	Category	Notes
SACP	0x000A	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	adv mode	P	Advertisement mode: ● 0 = Legacy (factory default) ● 1 = Extended ● 2 = Periodic
uint8	disc mode	M	Discovery mode: ● 0 = Non-discoverable/broadcast-only ● 1 = General discovery (factory default)
uint8	type	T	Advertisement type: (0-3 for legacy adv, 4-5 for periodic adv, 6-A for extended adv) ● 0x00 = Legacy: Connectable, undirected (factory default) ● 0x01 = Legacy:Connectable, directed ● 0x02 = Legacy:Scannable, undirected ● 0x03 = Legacy:Non-connectable, undirected ● 0x04 = Periodic: Undirected ● 0x05 = Periodic: Directed ● 0x06 = Extended: Undirected connectable ● 0x07 = Extended: Directed connectable ● 0x08 = Extended: Non-connectable, non-scannable ● 0x09 = Extended: Non-connectable, scannable ● 0x0A = Extended: Non-connectable anonymous directed
uint8_t	primary phy	H	PHY: ● 0 = 1M (factory default) ● 1 = 2M ● 2 = Coded
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)
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)
uint8	filter	L	Advertisement filter policy: ● 0 = Scan request and connect request from any (factory default) ● 1 = Scan request whitelist-only, connect request from any ● 2 = Scan request from any, connect request whitelist-only ● 3 = Scan request and connect request whitelist-only
uint16	timeout	O	Advertisement timeout (seconds): ● 0 to disable (factory default)
uint8	flags	F	Advertisement behavior flags bitmask: ● Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection ● Bit 1 (0x2) = Use custom advertisement and scan response data ● NOTE: Factory default = 0x00 (no bits set)
macaddr	directAddr	A	Directed advertisement address
uint8	directAddr Type	Y	Directed address type(if using directed advertisement mode): ● 0: BLE_ADDR_PUBLIC ● 1: BLE_ADDR_RANDOM
uint8_t	secondary phy	E	PHY: ● 0 = 1M (factory default) ● 1 = 2M ● 2 = Coded

Data Type	Name	Text	Description
uint8_t	secondary max skip	S	Maximum number of advertising events the controller can skip before sending the AUX_ADV_IND packets. 0: that AUX_ADV_IND PDUs shall be sent prior each advertising events (factory default)
uint8_t	secondary SID	D	Advertising SID • Minimum: 0x00 • Maximum: 0x0F • Factory default = 0x00
uint8_t	periodic interval	N	Advertisement interval (1.25ms units): • Minimum = 20ms • Maximum = 81.91875s • Factory default = 30ms

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_start_legacy_coded_adv(/CA, ID=4/31)
- gap_get_adv_legacy_coded_phy_parameters (GACP, ID=4/30)

7.2.4.30 gap_get_adv_legacy_coded_phy_parameters (GACP, ID=4/30)

Obtain the current advertisement parameters for all advertisement modes.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	1E	None.
RSP	C0		04	1E	None.

Text Info:

Text Name	Response Length	Category	Notes
GACP	0x0009	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8	adv mode	P	Advertisement mode: • 0 = Legacy (factory default) • 1 = Extended • 2 = Periodic
uint8	disc mode	M	Discovery mode: • 0 = Non-discoverable/broadcast-only • 1 = General discovery (factory default)

Data Type	Name	Text	Description
uint8	type	T	Advertisement type: (0-3 for legacy adv, 4-5 for periodic adv, 6-A for extended adv) ● 0x00 = Legacy: Connectable, undirected (factory default) ● 0x01 = Legacy:Connectable, directed ● 0x02 = Legacy:Scannable, undirected ● 0x03 = Legacy:Non-connectable, undirected ● 0x04 = Periodic: Undirected ● 0x05 = Periodic: Directed ● 0x06 = Extended: Undirected connectable ● 0x07 = Extended: Directed connectable ● 0x08 = Extended: Non-connectable, non-scannable ● 0x09 = Extended: Non-connectable, scannable ● 0x0A = Extended: Non-connectable anonymous directed
uint8_t	primary phy	H	PHY: ● 0 = 1M (factory default) ● 1 = 2M ● 2 = Coded
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)
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)
uint8	filter	L	Advertisement filter policy: ● 0 = Scan request and connect request from any (factory default) ● 1 = Scan request whitelist-only, connect request from any ● 2 = Scan request from any, connect request whitelist-only ● 3 = Scan request and connect request whitelist-only
uint16	timeout	O	Advertisement timeout (seconds): ● 0 to disable (factory default)
uint8	flags	F	Advertisement behavior flags bitmask: ● Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection ● Bit 1 (0x2) = Use custom advertisement and scan response data ● NOTE: Factory default = 0x00 (no bits set)
macaddr	directAddr	A	Directed advertisement address
uint8	directAddr Type	Y	Directed address type(if using directed advertisement mode): ● 0: BLE_ADDR_PUBLIC ● 1: BLE_ADDR_RANDOM
uint8_t	secondary phy	E	PHY: ● 0 = 1M (factory default) ● 1 = 2M ● 2 = Coded
uint8_t	secondary max skip	S	Maximum number of advertising events the controller can skip before sending the AUX_ADV_IND packets. 0: that AUX_ADV_IND PDUs shall be sent prior each advertising events (factory default)
uint8_t	secondary SID	D	Advertising SID ● Minimum: 0x00 ● Maximum: 0x0F ● Factory default = 0x00

Data Type	Name	Text	Description
uint8_t	periodic interval	N	Advertisement interval (1.25ms units): ● Minimum = 20ms ● Maximum = 81.91875s ● Factory default = 30ms

Related Commands:

- gap_set_adv_parameters (SAP, ID=4/23)
- gap_set_adv_legacy_coded_phy_parameters (SACP, ID=4/29)

7.2.4.31 gap_start_legacy_coded_adv (/CA, ID=4/31)

Start advertising for legacy or extended or periodic advertisement.

Function is the same as gap_start_adv (/A, ID=4/8), 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		04	1F	None.
RSP	C0		04	1F	None.

Text Info:

Text Name	Response Length	Category	Notes
/CA	0x0009	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	adv mode	P	Advertisement mode: ● 0 = Legacy (factory default) ● 1 = Extended ● 2 = Periodic
uint8	disc mode	M	Discovery mode: ● 0 = Non-discoverable/broadcast-only ● 1 = General discovery (factory default)
uint8	type	T	Advertisement type: (0-3 for legacy adv, 4-5 for periodic adv, 6-A for extended adv) ● 0x00 = Legacy: Connectable, undirected (factory default) ● 0x01 = Legacy:Connectable, directed ● 0x02 = Legacy:Scannable, undirected ● 0x03 = Legacy:Non-connectable, undirected ● 0x04 = Periodic: Undirected ● 0x05 = Periodic: Directed ● 0x06 = Extended: Undirected connectable ● 0x07 = Extended: Directed connectable ● 0x08 = Extended: Non-connectable, non-scannable ● 0x09 = Extended: Non-connectable, scannable ● 0x0A = Extended: Non-connectable anonymous directed
uint8_t	primary phy	H	PHY: ● 0 = 1M (factory default) ● 1 = 2M ● 2 = Coded

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)
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)
uint8	filter	F	Advertisement filter policy: ● 0 = Scan request and connect request from any (factory default) ● 1 = Scan request whitelist-only, connect request from any ● 2 = Scan request from any, connect request whitelist-only ● 3 = Scan request and connect request whitelist-only
uint16	timeout	O	Advertisement timeout (seconds): ● 0 to disable (factory default)
macaddr	directAddr	A	Directed advertisement address
uint8	directAddr Type	Y	Directed address type(if using directed advertisement mode): ● 0: BLE_ADDR_PUBLIC ● 1: BLE_ADDR_RANDOM
uint8_t	secondary phy	E	PHY: ● 0 = 1M (factory default) ● 1 = 2M ● 2 = Coded
uint8_t	secondary max skip	S	Maximum number of advertising events the controller can skip before sending the AUX_ADV_IND packets. 0: that AUX_ADV_IND PDUs shall be sent prior each advertising events (factory default)
uint8_t	secondary SID	D	Advertising SID ● Minimum: 0x00 ● Maximum: 0x0F ● Factory default = 0x00
uint8_t	periodic interval	N	Advertisement interval (1.25ms units): ● Minimum = 20ms ● Maximum = 81.91875s ● Factory default = 30ms

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)
- gap_stop_legacy_coded_adv(/CAX, ID=4/32)
- gap_set_adv_legacy_coded_phy_parameters (SACP, ID=4/29)

Related Events:

- gap_adv_state_changed (ASC, ID=4/2)

7.2.4.32 gap_stop_legacy_coded_adv(/CAX, ID=4/32)

Stop advertising.

This command immediately stops advertising if it is currently active.

Function is the same as gap_stop_adv (/AX, ID=4/9), 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		04	20	None.
RSP	C0		04	20	None.

Text Info:

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

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_start_legacy_coded_adv(/CA, ID=4/31)

Related Events:

- gap_adv_state_changed (ASC, ID=4/2)

7.2.4.33 set_scan_legacy_coded_parameters(SSCP, ID=4/33)

Configure new default scan parameters with coded phy related.

These parameters will be used when sending the gap_start_scan (/S, ID=4/10) or gap_start_legacy_coded_scan (/CS, ID=4/35) API commands in text mode without specifying non-default arguments.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	21	None.
RSP	C0		04	21	None.

Text Info:

Text Name	Response Length	Category	Notes
SSCP	0x000A	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	mode	M	Discovery mode: 0 = Observation mode 1 = 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)
uint16	window	W	Scan window (625 µs units): • Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms)

Data Type	Name	Text	Description
			• Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) • Factory default = 0x0100 (256 * 0.625 ms = 160 ms) • Cannot be greater than interval
uint8	active	A	Active scanning: • 0 = Passive scanning (factory default) • 1 = Active scanning
uint8	filter	F	Whitelist filter policy: • 0 = Accept all advertising packets (factory default) • 1 = Accept only from whitelisted devices • 2 = Accept only from devices sending directed advertisements to this device • 3 = Accept only from whitelisted devices sending directed advertisements to this device
uint8	nodupe	D	Duplicate filter policy: • 0 = Disable duplicate result filtering (factory default) • 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): • 0 to disable (factory default)
uint8	phy	P	PHY: . Bit 0 (0x1) = 1M . Bit 1 (0x2) = Coded
uint16	coded interval	C	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	coded window	E	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

Response Parameters:

None.

Related Commands:

- gap_start_scan (/S, ID=4/10)
- gap_get_scan_parameters (GSP, ID=4/26)
- get_scan_legacy_coded_parameters (GSCP, ID=4/34)
- gap_start_legacy_coded_scan (/CS, ID=4/35)

7.2.4.34 get_scan_legacy_coded_parameters (GSCP, ID=4/34)

Obtain the current scan parameters with coded phy related..

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	22	None.
RSP	C0		04	22	None.

Text Info:

Text Name	Response Length	Category	Notes
GSCP	0x0046	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8	mode	M	Discovery mode: • 0 = Observation mode • 1 = 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)
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
uint8	active	A	Active scanning: • 0 = Passive scanning (factory default) • 1 = Active scanning
uint8	filter	F	Whitelist filter policy: • 0 = Accept all advertising packets (factory default) • 1 = Accept only from whitelisted devices • 2 = Accept only from devices sending directed advertisements to this device • 3 = Accept only from whitelisted devices sending directed advertisements to this device
uint8	nodupe	D	Duplicate filter policy: • 0 = Disable duplicate result filtering (factory default) • 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): • 0 to disable (factory default)
uint8	phy	P	PHY: . Bit 0 (0x1) = 1M . Bit 1 (0x2) = Coded
uint16	coded interval	C	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	coded window	E	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

Related Commands:

- gap_set_scan_parameters (SSP, ID=4/25)
- set_scan_legacy_coded_parameters (SSCP, ID=4/33)

7.2.4.35 gap_start_legacy_coded_scan(/CS, ID=4/35)

Start scanning with coded phy parameters.

Function is the same as gap_start_scan (/S, ID=4/10), EZ-Serial will generate the gap_scan_state_changed (SSC, ID=4/3) API event when the scanning state changes.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	23	None.
RSP	C0		04	23	None.

Text Info:

Text Name	Response Length	Category	Notes
/CS	0x0009	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	mode	M	Discovery mode: • 0 = Observation mode • 1 = General discovery mode (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)
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
uint8	active	A	Active scanning: • 0 = Passive scanning (default) • 1 = Active scanning
uint8	filter	F	Whitelist filter policy: • 0 = Accept all advertising packets (default) • 1 = Accept only from whitelisted devices • 2 = Accept only from devices sending directed advertisements to this device (not support) • 3 = Accept only from whitelisted devices sending directed advertisements to this device.(not support)
uint8	nodupe	D	Duplicate filter policy: • 0 = Disable duplicate result filtering (default) • 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): • 0 to disable. Only discovery mode (0 = Observation mode) supports continuous sanning (timeout = 0). • 0x0A (default)
uint8	phy	P	PHY: • Bit 0 (0x1) = 1M • Bit 1 (0x2) = Coded
uint16	coded interval	C	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	coded window	E	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)

Data Type	Name	Text	Description
			• Cannot be greater than interval

Response Parameters:

None.

Related Commands:

- gap_stop_scan (/SX, ID=4/11)
- gap_set_scan_parameters (SSP, ID=4/25)
- get_scan_legacy_coded_parameters (GSCP, ID=4/34)
- gap_stop_legacy_coded_scan (/CSX, ID=4/36)

Related Events:

- gap_scan_state_changed (SSC, ID=4/3)
- gap_scan_result (S, ID=4/4)

7.2.4.36 gap_stop_legacy_coded_scan(/CSX, ID=4/36)

Stop scanning.

Function is the same as gap_stop_scan (/SX, ID=4/11), EZ-Serial will generate the gap_scan_state_changed (SSC, ID=4/3) API event when the scanning state changes.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	24	None.
RSP	C0		04	24	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)
- gap_start_legacy_coded_scan (/CS, ID=4/35)

Related Events:

- gap_scan_state_changed (SSC, ID=4/3)

7.2.4.37 gap_phy_update (/UP, ID=4/37)

Request a PHY update for an active connection.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	25	None.
RSP	C0		04	25	None.

Text Info:

Text Name	Response Length	Category	Notes
/UP	0x0009	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Handle of connection to update (Ignored in current release due to internal BLE stack functionality, set to 0)
uint8	TX phy	T*	PHY bit: • 0x00 = Any • Bit 0 (0x1) = 1M • Bit 1 (0x2) = 2M • Bit 2 (0x4) = Coded
uint8	RX phy	R*	PHY: • 0x00 = Any • Bit 0 (0x1) = 1M • Bit 1 (0x2) = 2M • Bit 2 (0x4) = Coded
uint8	phy option	O*	PHY: • 0 = No preferred coding when transmitting on the LE Coded PHY. • 1 = Prefers that S=2 coding • 2 = Prefers that S=8 coding

Response Parameters:

None.

Related Commands:

- gap_connect (/C, ID=4/1)

Related Events:

- gap_phy_updated (PU, ID=4/9)

7.2.4.38 gap_set_extended_adv_data (SEAD, ID=4/38)

Configure new custom advertisement packet data for extended or periodic advertisement mode. The maximum extended data length is 1270 bytes.

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) or gap_set_extended_adv_data (SEAD, ID=4/38) API command. If you only set custom data but do not enable user-defined content, the data here will remain unused.

NOTE: Not supported in version 1.0.x.x, please use gap_set_adv_data (SAD, ID=4/19) API command instead.

NOTE: Default advertising data [02 01 06] is set.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	26	Variable-length command payload, minimum of 1 (0x01), maximum of 251(0xFB)
RSP	C0		04	26	None.

Text Info:

Text Name	Response Length	Category	Notes
SEAD		SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8a	data	D	New extended advertisement payload data (0-251 bytes), should be a valid advertisement payload format. NOTE: uint8a data type requires one prefixed “length” byte before binary parameter payload
uint8	type	T*	Type: • 0 = Erase all • 1 = Append advertisement data

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_set_adv_parameters (SAP, ID=4/23)
- gap_set_adv_legacy_coded_phy_parameters (SACP, ID=4/29)
- gap_start_legacy_coded_adv(/CA, ID=4/31)
- gap_get_extended_adv_data (GEAD, ID=4/39)

7.2.4.39 gap_get_extended_adv_data (GEAD, ID=4/39)

Obtain the current custom advertisement packet data for extended or periodic advertisement mode.

> NOTE: Not supported in version 1.0.x.x, please use gap_get_adv_data (GAD, ID=4/20) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	39	None.
RSP	C0		04	39	Variable-length response payload, minimum of 3 (0x03), maximum of 1653 (0x675)

Text Info:

Text Name	Response Length	Category	Notes
GEAD	B	GET	Variable-length response payload, minimum of 13 (0x0D), maximum of 3373 (0xCF1)

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8a	data	D	Current advertisement payload data (3-1270 bytes) Default data is 020106. NOTE: uint8a data type requires one prefixed “length” byte before binary parameter payload

Related Commands:

- gap_set_extended_adv_data (SEAD, ID=4/38)

7.2.4.40 gap_set_extended_scan_response_data(SERD, ID=4/40)

Configure new custom scan response packet payload for extended or periodic advertisement mode. The maximum extended data length is 1270 bytes.

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) or gap_set_extended_adv_data (SEAD, ID=4/38) API command. If you only set custom data but do not enable user-defined content, the data here will remain unused.

NOTE: Not supported in version 1.0.x.x, please use gap_set_sr_data (SSRD, ID=4/21) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	26	Variable-length command payload, minimum of 1 (0x01), maximum of 251(0xFB)
RSP	C0		04	26	None.

Text Info:

Text Name	Response Length	Category	Notes
SERD		SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8a	data	D	New extended advertisement payload data (0-251 bytes), should be a valid advertisement payload format. NOTE: uint8a data type requires one prefixed “length” byte before binary parameter payload
uint8	type	T*	Type: • 0 = Erase all • 1 = Append advertisement data

Response Parameters:

None.

Related Commands:

- gap_start_adv (/A, ID=4/8)
- gap_set_adv_parameters (SAP, ID=4/23)
- gap_set_adv_legacy_coded_phy_parameters (SACP, ID=4/29)
- gap_start_legacy_coded_adv(/CA, ID=4/31)
- gap_get_extended_scan_response_data(GERD, ID=4/41)

7.2.4.41 gap_get_extended_scan_response_data(GERD, ID=4/41)

Obtain the current custom scan response packet data for extended or periodic advertisement mode.

> NOTE: Not supported in version 1.0.x.x, please use gap_get_adv_data (GAD, ID=4/20) API command instead.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0		04	39	None.
RSP	C0		04	39	Variable-length response payload, minimum of 3 (0x03), maximum of 1653 (0x675)

Text Info:

Text Name	Response Length	Category	Notes
GERD	B	GET	Variable-length response payload, minimum of 13 (0x0D), maximum of 3373 (0xCF1)

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8a	data	D	Current advertisement payload data (0-1270 bytes) NOTE: uint8a data type requires one prefixed “length” byte before binary parameter payload

Related Commands:

gap_set_extended_scan_response_data(SERD, ID=4/40)

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)
- gatts_service_active (/SACT, ID=5/16)
- gatts_service_handle_reset (/RSHL, ID=5/17)

Events within this group are documented in Section 7.3.5, GATT Server Group (ID=5).

7.2.5.1 gatts_create_attr (/CAC, ID=5/1)

Add 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 Section 7.4.2 (EZ-Serial GATT Database Validation Error Codes).

Refer to Section 3.6.1 (How to Define Custom Local GATT Services and Characteristics) and Section 10.2 (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 34 (0x0022). Therefore, the first custom attribute created in a factory default state will receive the handle value 35 (0x0023).

> **NOTE:**
>
> Additions to the GATT structure are not effective and stored in flash immediately until sending gatts_service_active (/SACT, ID=5/16) API command.
>
> 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: ● 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*
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

Response Parameters:

Data Type	Name	Text	Description
uint16	handle	H	New attribute handle (0x0023-0xFFFF)
uint16	valid	V	GATT database validity status

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:

- Section 3.6.1 (How to Define Custom Local GATT Services and Characteristics)
- Section 10.2 (Adopted Bluetooth SIG GATT Profile Structure Snippets)

7.2.5.2 gatts_delete_attr (/CAD, ID=5/2)

Remove one or more attributes from the GATT structure. CYW20822 device only support remove a service handle, if specpic the handle number is not a service handle, it will report an error.

If you use this command without a handle in text mode or you supply handle value 0 in either text or binary mode, then 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.

After removing an attribute with this command, the local GATT database may no longer be strictly valid. Refer to Section 7.4.2 (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 34 (0x0022). Therefore, you cannot delete any attribute with a handle value less than 34 (0x0022).

> **NOTE:** Removals from the GATT structure are set the service to invisible and is stored in flash once the "result" value in the response indicates success. The deleted handle numbers would be blocked until send gatts_service_handle_reset (/RSHL, ID=5/17) API command.
>
> 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	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)

Check to ensure 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. Refer to Section 7.4.2 (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).

NOTE: If EZ-serial received error code 0x0111 in response package means that you need to reset the device to make the settings available.

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

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)

Store the current custom GATT structure in flash.

> **NOTE: This command has been deprecated and has no effect when used.** As of 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.

	Type	Length	Group	ID	Notes
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)

List 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 0x0023, do not include fixed attributes (default) • 1 = Start from handle 0x0012.

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)

Request a list of all services in the local GATT structure which the attribute handle is reported from 0x12.

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, you should not 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(Minimum from 0x12)
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:

Section 0 (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.

- How to List Local GATT Services, Characteristics, and Descriptors)

7.2.5.7 gatts_discover_characteristics (/DLC, ID=5/7)

Request a list of all characteristics in the local GATT structure which the attribute handle is reported from 0x12

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, you should not 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(Minimum from 0x12)
uint16	end	E	Handle to end searching (inclusive)
uint16	service	S	Service UUID filter (0 for all) – **Currently not implemented in firmware, set to 0**

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:

- Section 3.6.2 (How to List Local GATT Services, Characteristics, and Descriptors)

7.2.5.8 gatts_discover_descriptors (/DLD, ID=5/8)

Request a list of all descriptors in the local GATT structure which the attribute handle is reported from 0x12

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, you should not 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 (Minimum from 0x12)
uint16	end	E	Handle to end searching (inclusive)
uint16	service	S	Service UUID filter (0 for all) **(Ignored in current release, set to 0)**
uint16	characteristic	C	Characteristic UUID filter (0 for all) **(Ignored in current release, set to 0)**

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:

- Section 3.6.2 (How to List Local GATT Services, Characteristics, and Descriptors)

7.2.5.9 gatts_read_handle (/RLH, ID=5/9)

Read 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 attribute to read value from, which attributes handle is from 0x23, not include fixed attributes.

Response Parameters:

Data Type	Name	Text	Description
longuint8a	data	D	Data read from 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)

Write a new value to an 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 fo 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 attribute to write new value to, which attributes handle is from 0x23, not include fixed attributes.
longuint8a	data	D*	New data to write to attribute **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)

Notify 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.

	Type	Length	Group	ID	Notes
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 use for notification **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint16	attr_handle	H*	Handle of attribute to notify
uint8a	data	D*	Data to push to 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)

Indicate a new attributes 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.

Because this method requires client acknowledgement, 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 one for the acknowledgement. Using this type of transfer has effects on potential throughput; refer to Section 3.10.1 (How to Maximize 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 use for indication **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint16	attr_handle	H*	Handle of attribute to indicate

Data Type	Name	Text	Description
uint8a	data	D*	Data to indicate **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)

Respond 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.

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 use 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)

Configure 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 bitmask: ● Bit 0 (0x01) = Enable automatic response to acknowledged writes ● **NOTE:** Factory default is 0x01 (all bits set)

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)

Obtain 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 bitmask: ● Bit 0 (0x01) = Enable automatic response to acknowledged writes ● **NOTE:** Factory default is 0x01 (all bits set)

Related Commands:

- gatts_set_parameters (SGSP, ID=5/14)

7.2.5.16 gatts_service_active (/SACT, ID=5/16)

After adding all attributes by the gatts_create_attr command, externally must send the gatts_service_active command to apply. After applying, the attribute added by the gatts_create_attr command will be stored in flash.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	05	0F	None.
RSP	C0	02	05	0F	None.

Text Info:

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

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- gatts_create_attr (/CAC, ID=5/1)

7.2.5.17 gatts_service_handle_reset (/RSHL, ID=5/17)

This command reorder the attribute handles, and removed the deleted attribute handles which are deleted by the gatts_delete_attr command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	05	10	None.
RSP	C0	02	05	10	None.

Text Info:

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

Command Arguments:

None.

Response Parameters:

None.

Related Commands:

- system_reboot (/RBT, ID=2/2)
- gatts_delete_attr (/CAD, ID=5/2)

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 Section 7.3.6 (GATT Client Group (ID=6)

7.2.6.1 gattc_discover_services (/DRS, ID=6/1)

Request 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:** Because this command works with remote data, it cannot determine the number of records to be returned in advance. 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 use for discovery **(Ignored in current release due to internal BLE stack functionality, set to 0)**
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:

- Section 3.7.1 (How to Discover a Remote Server's GATT Structure)

7.2.6.2 gattc_discover_characteristics (/DRC, ID=6/2)

Request 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:** Because this command works with remote data, it cannot determine the number of records to be returned in advance. 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	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 use for discovery **(Ignored in current release due to internal BLE stack functionality, set to 0)**
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:

- Section 3.7.1 (How to Discover a Remote Server's GATT Structure)

7.2.6.3 gattc_discover_descriptors (/DRD, ID=6/3)

Request 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 instead, use the gatts_discover_descriptors (/DLD, ID=5/8) API command.

> **NOTE:** Because this command works with remote data, it cannot determine the number of records to be returned in advance. 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	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 use for discovery **(Ignored in current release due to internal BLE stack functionality, set to 0)**
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)**
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_services (/DRS, 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:

- Section 3.7.1 (How to Discover a Remote Server's GATT Structure)

7.2.6.4 gattc_read_handle (/RRH, ID=6/4)

Read 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 instead, use the gatts_read_handle (/RLH, ID=5/9) API command.

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 use for read operation **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint16	attr_handle	H*	Handle of remote attribute to 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)

Write 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 instead, use the gatts_write_handle (/WLH, ID=5/10) API command.

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 use for write operation **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint16	attr_handle	H*	Handle of remote attribute to write
uint8	type	T	Type of write to perform: • 0 = Simple write – acknowledged (default) • 1 = Write without response – unacknowledged
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)

Confirm 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 event documentation for detail.

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 use for confirmation **(Ignored in current release due to internal BLE stack functionality, set to 0)**

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) – Configure 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)

Configure 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 of 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)

Get 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_generate_oob_data (/GOOB, ID=7/7)
- smp_clear_oob_data (/COOB, ID=7/8)
- smp_set_privacy_mode (SPRV, ID=7/9)
- smp_get_privacy_mode (GPRV, ID=7/10)
- smp_set_security_parameters (SSBP, ID=7/11)
- smp_get_security_parameters (GSBP, ID=7/12)
- smp_set_fixed_passkey (SFPK, ID=7/13)
- smp_get_fixed_passkey (GFPK, ID=7/14)

Events within this group are documented in Section 7.3.7, SMP Group (ID=7).

7.2.7.1 smp_query_bonds (/QB, ID=7/1)

Request 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 that many smp_bond_entry (B, ID=7/1) API events that provide details for each device.

> **NOTE:** EZ-Serial currently supports a maximum of 3 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)

Remove 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).

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)

Initiate pairing process with a connected device.

> **NOTE:** EZ-Serial currently supports a maximum of 3 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	05	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 **(Ignored in current release due to internal BLE stack functionality, set to 0)**

Data Type	Name	Text	Description
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, factory default) ● 0x12 = Mode 1, Level 3 – Authenticated pairing with encryption (with MITM) ● 0x13 = Mode 1, Level 4 – LE Secure Connections (reported by remote peers only, not locally implemented in current EZ-Serial firmware) ● 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: (Ignored in current release due to internal BLE stack functionality, set to 1 always) ●
uint8	keysize	K	Encryption key size (7-16), value ignored if pairing initiated by slave device ● **NOTE:** Factory default is 16 bytes (0x10)
uint8	pairprop	P	Pairing properties: ● Bit 0 (0x01): MITM enabled for Secure Connections (SC) ● **NOTE:** Factory default is 0x00 (no bits set)

Response Parameters:

None.

Related Commands:

- smp_send_pairreq_response (/PR, ID=7/5) – Use when remote device initiates pairing and auto-accept flag bit is not disabled
- smp_send_passkeyreq_response (/PE, ID=7/6) – Use if MITM protection is enabled and pairing requires passkey entry
- smp_set_security_parameters (SSBP, ID=7/11) – Use to configure default security settings

Related Events:

- smp_pairing_requested (P, ID=7/2) – Occurs when remote device initiates pairing
- smp_pairing_result (PR, ID=7/3) – Occurs when pairing process completes (success or failure)
- smp_encryption_status (ENC, ID=7/4) – Occurs when encryption status changes during a pairing process
- smp_passkey_display_requested (PKD, ID=7/5) – Occurs when pairing process requires displaying a passkey to the user
- smp_passkey_entry_requested (PKE, ID=7/6) – Occurs when pairing process requires the user to enter a passkey

7.2.7.4 smp_query_random_address (/QRA, ID=7/4)

Request 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 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)

Send a response to a 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_security_parameters (SSBP, ID=7/11) API command. If the auto-accept feature is disabled, use this command to manually accept or deny a remotely initiated pairing process.

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 use for sending response **(Ignored in current release due to internal BLE stack functionality, set to 0)**
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_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)

Send 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.

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 use for sending response **(Ignored in current release due to internal BLE stack functionality, set to 0)**

Data Type	Name	Text	Description
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_generate_oob_data (/GOOB, ID=7/7)

Generate out-of-band data for pairing.

EZ-Serial supports the use of out-of-band (OOB) encryption key sharing for added security during pairing with compatible devices. This command does not directly set OOB data. Instead, it generates OOB data based on a 16-byte input key. You must use the same key on the remote device to generate matching OOB data in order to successfully pair using out-of-band key exchange.

Ensure that you generate OOB data on both sides of the connection before initiating the pairing process on either side.

> **NOTE:** EZ-Serial will always attempt to use OOB encryption data for pairing if you have set it using this command. If you set OOB data and then attempt to pair with a device that does not support OOB pairing, or that does not have the correct matching key set, pairing will always fail. To clear OOB data and revert to the standard pairing and key generation/exchange process, either reset the module via hardware or software or use the smp_clear_oob_data (/COOB, ID=7/8) API command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	12	07	07	None.
RSP	C0	02	07	07	None.

Text Info:

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

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for applying OOB data **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint8a	key	K*	16-byte key with which to generate OOB data **NOTE:** uint8a data type requires one prefixed “length” byte before binary parameter payload

Response Parameters:

None.

Related Commands:

- smp_clear_oob_data (/COOB, ID=7/8)

Example Usage:

- Section 3.8.3 (How to Use Out-of-Band Pairing)

7.2.7.8 smp_clear_oob_data (/COOB, ID=7/8)

Clear previously set out-of-band data for pairing.

NOTE: EZ-Serial will always attempt to use OOB encryption data for pairing if you have set it using the smp_generate_oob_data (/GOOB, ID=7/7) API command. If you set OOB data and then attempt to pair with a device that does not support OOB pairing, or that does not have the correct matching OOB security data set, pairing will always fail. To clear OOB data and revert to the standard pairing and key generation/exchange process, use this command or else reset the module via hardware or software.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01	07	08	None.
RSP	C0	02	07	08	None.

Text Info:

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

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for applying OOB data **(Ignored in current release due to internal BLE stack functionality, set to 0)**

Related Commands:

- smp_generate_oob_data (/GOOB, ID=7/7)

7.2.7.9 smp_set_privacy_mode (SPRV, ID=7/9)

Configure new privacy settings.

Use this 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.

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 bitmask: • Bit 0 (0x01) = Enable peripheral privacy • Bit 1 (0x02) = Enable central privacy • **NOTE:** Factory default is 0x00 (no bits set)
uint16	interval	I	Randomization interval (seconds)

Response Parameters:

None.

Related Commands:

- smp_get_privacy_mode (GPRV, ID=7/10)

7.2.7.10 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 bitmask: • Bit 0 (0x01) = Enable peripheral privacy • Bit 1 (0x02) = Enable central privacy • **NOTE:** Factory default is 0x00 (no bits set)
uint16	interval	I	Randomization interval (seconds)

Related Commands:

- smp_set_privacy_mode (SPRV, ID=7/9)

7.2.7.11 smp_set_security_parameters (SSBP, ID=7/11)

Configure new 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. Refer to the related commands and events for details concerning each one's use. Also, MITM protection requires I/O capabilities other than "No Input + No Output" in order to function correctly.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	06	07	0B	None.
RSP	C0	02	07	0B	None.

Text Info:

Text Name	Response Length	Category	Notes
SSBP	0x000A	SET	None.

Command Arguments:

Data Type	Name	Text	Description
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, factory default) ● 0x12 = Mode 1, Level 3 – Authenticated pairing with encryption (with MITM) ● 0x13 = Mode 1, Level 4 – LE Secure Connections (reported by remote peers only, not locally implemented in current EZ-Serial firmware) ● 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: ● (Ignored in current release due to internal BLE stack functionality, set to 1 always)
uint8	keysize	K	Encryption key size (7-16), value ignored if pairing initiated by slave device ● **NOTE:** Factory default is 16 bytes (0x10)
uint8	pairprop	P	Pairing properties: ● Bit 0 (0x01): MITM enabled for Secure Connections (SC) ● **NOTE:** Factory default is 0x00 (no bits set)
uint8	io	I	I/O capabilities: ● 0 = Display Only – ability to convey a 6-digit number to user ● 1 = Display + Yes/No – display and the ability to have user indicate “yes” or “no” ● 2 = Keyboard Only – ability for the user to enter ‘0’ through ‘9’ and “yes” or “no” ● 3 = No Input + No Output – no ability to display or input anything (factory default) ● 4 = Keyboard + Display – ability to provide full numeric input and display
uint8	flags	F	Security behavior flags bitmask: ● Bit 0 (0x01) = Enable auto-accept for incoming pairing requests ● Bit 1 (0x02) = Enable use of fixed passkey during pairing ● **NOTE:** Factory default is 0x01

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_get_security_parameters (GSBP, ID=7/12)
- 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)

7.2.7.12 smp_get_security_parameters (GSBP, ID=7/12)

Obtain current security and bonding parameters.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	07	0C	None.
RSP	C0	08	07	0C	None.

Text Info:

Text Name	Response Length	Category	Notes
GSBP	0x0028	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
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, factory default) ● 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: ● (Ignored in current release due to internal BLE stack functionality, set to 1 always)
uint8	keysize	K	Encryption key size (7-16), value ignored if pairing initiated by slave device ● **NOTE:** Factory default is 16 bytes (0x10)
uint8	pairprop	P	Pairing properties: ● Bit 0 (0x01): MITM enabled for Secure Connections (SC) ● **NOTE:** Factory default is 0x00 (no bits set)
uint8	io	I	I/O capabilities: ● 0 = Display Only – ability to convey a 6-digit number to user ● 1 = Display + Yes/No – display and the ability to have user indicate “yes” or “no” ● 2 = Keyboard Only – ability for the user to enter ‘0’ through ‘9’ and “yes” or “no” ● 3 = No Input + No Output – no ability to display or input anything (factory default) ● 4 = Keyboard + Display – ability to provide full numeric input and display
uint8	flags	F	Security behavior flags bitmask: ● Bit 0 (0x01) = Enable auto-accept for incoming pairing requests ● Bit 1 (0x02) = Enable use of fixed passkey during pairing ● **NOTE:** Factory default is 0x01

Related Commands:

- smp_set_security_parameters (SSBP, ID=7/11)

7.2.7.13 smp_set_fixed_passkey (SFPK, ID=7/13)

Configure 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 in order to pair successfully.

> **NOTE:** The fixed passkey defined here will only take effect 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_security_parameters (SSBP, ID=7/11) API command. If both of these conditions are not met, then the stack will revert to the default behavior of using a random passkey.

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)

Data Type	Name	Text	Description
			• **NOTE:** Factory default is 0

Response Parameters:

None.

Related Commands:

- smp_pair (/P, ID=7/3)
- smp_send_pairreq_response (/PR, ID=7/5)
- Example Usage:
- Section 3.8.2.5 (Pairing and Bonding with a Fixed Passkey)
- smp_get_fixed_passkey (GFPK, ID=7/14)
- smp_set_security_parameters (SSBP, ID=7/11)

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:

- Section 3.8.2.5 (Pairing and Bonding with a Fixed Passkey)

7.2.7.14 smp_get_fixed_passkey (GFPK, ID=7/14)

Obtain 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:** Factory default is 0

Related Commands:

- smp_set_fixed_passkey (SFPK, ID=7/13)

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_connect (/LC, ID=8/1)
- l2cap_disconnect (/LDIS, ID=8/2)
- l2cap_register_psm (/LRP, ID=8/3)
- l2cap_send_connreq_response (/LCR, ID=8/4)
- l2cap_send_credits (/LSC, ID=8/5)
- l2cap_send_data (/LD, ID=8/6)

Events within this group are documented in Section 7.3.8, L2CAP Group (ID=8).

7.2.8.1 l2cap_connect (/LC, ID=8/1)

Open a direct L2CAP channel to a connected device.

EZ-Serial provides one extra dedicated L2CAP channel for connection-oriented communication, bypassing the GATT/ATT layers of the stack. L2CAP connections use a credit-based flow control mechanism, where the receiving side grants a certain number of credits to the transmitting side to control its ability to send data over the open channel. For further details, refer to the example usage in Section 3.10.3 (How to Communicate Using an L2CAP Channel).

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

> **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.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	0B	08	01	None.
RSP	C0	02	08	01	None.

Text Info:

Text Name	Response Length	Category	Notes
/LC	0x0009	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for L2CAP channel **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint16	remote	R*	Remote Protocol Service Multiplexer (PSM)
uint16	local	L*	Local Protocol Service Multiplexer (PSM)
uint16	mtu	T*	Maximum Transmission Unit (MTU)
uint16	mps	P*	Maximum Payload Size (MPS), must be less than or equal to MTU
uint16	credits	Z*	Transmission credits initially granted to remote device

Response Parameters:

None.

Related Commands:

- l2cap_disconnect (/LDIS, ID=8/2)
- l2cap_register_psm (/LRP, ID=8/3) – Use on both local and remote devices to register a PSM before initiating a connection
- l2cap_send_connreq_response (/LCR, ID=8/4) – Use on the remote device to accept or reject a connection request

Related Events:

- l2cap_connection_requested (LCR, ID=8/1) – Occurs on the remote device after requesting a connection
- l2cap_connection_response (LC, ID=8/2) – Occurs locally after a remote device responds to a connection request

Example Usage:

- Section 3.10.3 (How to Communicate Using an L2CAP Channel)

7.2.8.2 l2cap_disconnect (/LDIS, ID=8/2)

Close a previously opened L2CAP channel.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Binary Header:

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

Text Info:

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

Command Arguments:

Data Type	Name	Text	Description
uint16	channel	N*	Local PSM channel to disconnect

Response Parameters:

None.

Related Commands:

- l2cap_connect (/LC, ID=8/1)

Related Events:

- l2cap_disconnected (LDIS, ID=8/4)

Example Usage:

- Section 3.10.3 (How to Communicate Using an L2CAP Channel)

7.2.8.3 l2cap_register_psm (/LRP, ID=8/3)

Register a new L2CAP PSM channel.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

You must use this command before initiating an L2CAP connection to a remote device. The remote device must also have the same command (or equivalent) run prior to the connection attempt. The low credit watermark value controls at which point the local device will generate the l2cap_rx_credits_low (LRCL, ID=8/5) API event, signaling that you should send additional credits to allow continued data flow.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	04	08	03	None.
RSP	C0	02	08	03	None.

Text Info:

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

Command Arguments:

Data Type	Name	Text	Description
uint16	channel	N*	Local PSM channel to register
uint16	watermark	W	Low credit watermark (default = 0)

Response Parameters:

None.

Related Commands:

- l2cap_connect (/LC, ID=8/1)

Related Events:

- l2cap_rx_credits_low (LRCL, ID=8/5) – Occurs locally when the remote device's transmit credits reach the watermark level

Example Usage:

- Section 3.10.3 (How to Communicate Using an L2CAP Channel)

7.2.8.4 l2cap_send_connreq_response (/LCR, ID=8/4)

Respond to an incoming L2CAP connection request.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	0B	08	04	None.
RSP	C0	02	08	04	None.

Text Info:

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

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for L2CAP response **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint16	channel	N*	Remote Protocol Service Multiplexer (PSM)
uint16	response	R*	Response (0 = accept, non-zero = reject)
uint16	mtu	M*	Maximum Transmission Unit (MTU)
uint16	mps	P*	Maximum Payload Size (MPS), must be less than or equal to MTU

Data Type	Name	Text	Description
uint16	credits	Z*	Transmission credits initially granted to remote device

Response Parameters:

None.

Related Commands:

- l2cap_connect (/LC, ID=8/1) – Used to initiate an L2CAP connection

Related Events:

- l2cap_connection_requested (LCR, ID=8/1) – Occurs locally when a remote device initiates an L2CAP connection
- l2cap_connection_response (LC, ID=8/2) – Occurs on the remote device after sending the response to a connection request

Example Usage:

- Section 3.10.3 (How to Communicate Using an L2CAP Channel)

7.2.8.5 l2cap_send_credits (/LSC, ID=8/5)

Send additional transmission credits for L2CAP channel.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Use this command if you receive the l2cap_rx_credits_low (LRCL, ID=8/5) API event, indicating that the remote end of a given L2CAP channel has few or no credits remaining to send data. You can also use this command preemptively to keep the remote device from running out of credits. The remote device will be unable to send more data if it runs out of credits until the local device grants additional credits with this command.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	04	08	05	None.
RSP	C0	02	08	05	None.

Text Info:

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

Command Arguments:

Data Type	Name	Text	Description
uint16	channel	N*	Channel ID
uint16	credits	Z*	Credits

Response Parameters:

None.

Related Commands:

- l2cap_connect (/LC, ID=8/1) – Used on the initiating side to grant first block of credits to the remote device
- l2cap_send_connreq_response (/LCR, ID=8/4)

Related Events:

- l2cap_data_received (LD, ID=8/3)
- l2cap_rx_credits_low (LRCL, ID=8/5)
- l2cap_tx_credits_received (LTCR, ID=8/6)

Example Usage:

- Section 3.10.3 (How to Communicate Using an L2CAP Channel)

7.2.8.6 l2cap_send_data (/LD, ID=8/6)

Send data over an open L2CAP channel.

NOTE: L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Each transmission with this command uses one TX credit, regardless of length. To maximize throughput, make sure you fill the packet with as many bytes as possible based on the data available in your transmission buffer.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	05	08	06	Variable-length command payload, value specified is minimum
RSP	C0	02	08	06	None.

Text Info:

Text Name	Response Length	Category	Notes
/LD	0x0009	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle over which to send data **(Ignored in current release due to internal BLE stack functionality, set to 0)**
uint16	channel	N*	Channel ID over which to send data
longuint8a	data	D*	Data (0-23 bytes) **NOTE:** longuint8a data type requires two prefixed “length” bytes before binary parameter payload

Response Parameters:

None.

Related Events:

- l2cap_data_received (LD, ID=8/3) – Occurs on the remote device after data arrives

Example Usage:

- Section 3.10.3 (How to Communicate Using an L2CAP Channel)

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 Section 7.3.9 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 features in common:

- 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 Section 8 (GPIO Reference) for details about which pins 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 CYBLE-212019-00 module. Note that the CYBLE-212019-00 module does not expose P2.1, P2.5, or P2.7.

<table>
<tr><th>Step</th><th>Result</th></tr>
<tr><td>Command received:

SIOL,P=2,M=FF,L=AA
Port: 2
Pins: FF (select all)
Logic: AA (0b10101010)</td><td>
<table>
<tr><td>P2.7</td><td>P2.6</td><td>P2.5</td><td>P2.4</td><td>P2.3</td><td>P2.2</td><td>P2.1</td><td>P2.0</td></tr>
<tr><td>HIGH</td><td>LOW</td><td>HIGH</td><td>LOW</td><td>HIGH</td><td>LOW</td><td>HIGH</td><td>LOW</td></tr>
</table>
</td></tr>
<tr><td>Command processor clears bits from the selection mask for any non-exposed pins to avoid unexpected behavior</td><td>
<table>
<tr><td>P2.7</td><td>P2.6</td><td>P2.5</td><td>P2.4</td><td>P2.3</td><td>P2.2</td><td>P2.1</td><td>P2.0</td></tr>
<tr><td>X</td><td></td><td>X</td><td></td><td></td><td></td><td>X</td><td></td></tr>
</table>
</td></tr>
<tr><td>Logic states applied, response sent:

@R,000F,SIOL,0000,A=5D
Result: 0000 (success)
Affected: 5D (01011101)</td><td>
<table>
<tr><td>P2.7</td><td>P2.6</td><td>P2.5</td><td>P2.4</td><td>P2.3</td><td>P2.2</td><td>P2.1</td><td>P2.0</td></tr>
<tr><td>n/a</td><td>LOW</td><td>n/a</td><td>LOW</td><td>HIGH</td><td>LOW</td><td>n/a</td><td>LOW</td></tr>
</table>
</td></tr>
</table>

7.2.9.1 gpio_query_logic (/QIOL, ID=9/1)

Read the active low/high logic state of pins on the selected port.

See Section 8.1 (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.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	01	09	01	None.
RSP	C0	03	09	01	None.

Text Info:

Text Name	Response Length	Category	Notes
/QIOL	0x0010	ACTION	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	Pin	P*	Pin number

Response Parameters:

Data Type	Name	Text	Description
uint8	logic	L	Pin logic mask (set bit for high, clear for low)

Related Commands:

- gpio_set_logic (SIOL, ID=9/7) – Use to set output/pull logic state internally (may be overridden by external connections)
- gpio_get_logic (GIOL, ID=9/8) – Use 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 a single dedicated ADC input pin (**ADC**) for reading analog voltages. The ADC supports an input voltage range of **0 V** minimum to **VBAT** 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** pin 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 Section 8.1 (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	02	09	02	None.

Text Info:

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

Command Arguments:

Data Type	Name	Text	Description
uint8	channel	N*	ADC channel (4 only)
uint8	reference	R	Voltage reference for conversion **(Ignored in current release, set to 0)**

Response Parameters:

Data Type	Name	Text	Description
uint16	value	A	Raw ADC conversion value, 0 – 1023(0x0 – 0x7FF)
uint32	uvolts	U	Scaled ADC result in microvolts, 0 – VBAT

7.2.9.3 gpio_set_function (SIOF, ID=9/3)

Configure new special function assignment on selected pins.

See Section 8.1 (GPIO Pin Map for Supported Modules) for a pin map table showing pin availability and default assignment. Refer to the general overview in Section 7.2.9, GPIO Group (ID=9), for guidelines on how pin selection and configuration masks work.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	04	09	03	None.
RSP	C0	03	09	03	None.

Text Info:

Text Name	Response Length	Category	Notes
SIOF	0x000F	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	Pin	P*	GPIO Pin number
uint8	enable	E	Pin function mask (1 to enable, 0 to disable)
uint8	drive	D	Pin function drive mode

Response Parameters:

Data Type	Name	Text	Description
uint8	affected	A	1 for affected, 0 for unaffected

Related Commands:

- gpio_get_function (GIOF, ID=9/4)

7.2.9.4 gpio_get_function (GIOF, ID=9/4)

Get current special function assignment on selected pins.

See Section 8.1 (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	04	None.
RSP	C0	04	09	04	None.

Text Info:

Text Name	Response Length	Category	Notes
GIOF	0x0014	GET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	Pin	P*	GPIO Pin number

Response Parameters:

Data Type	Name	Text	Description
uint8	enable	E	Pin function (1 indicates enabled, 0 indicates disabled)
uint8	drive	D	Pin function drive mode

Related Commands:

- gpio_set_function (SIOF, ID=9/3)

7.2.9.5 gpio_set_drive (SIOD, ID=9/5)

Configure 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. describes each resulting drive mode from all combinations:

Table 7-3. GPIO Drive Mode Table

DriveModeIndex	Drive mode
0	Analog input, high impedance
1	Digital input, high impedance
2	Digital input, pull-up
3**1**	Digital output, strong drive
4	Digital output, open-drain drives high
5	Digital output, open-drain drives low

See Section 8.1 (GPIO Pin Map for Supported Modules) for a pin map table showing pin availability and default assignment. Refer to the general overview in Section 7.2.9, GPIO Group (ID=9), for guidelines on how pin selection and configuration masks work.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	06	09	05	None.
RSP	C0	03	09	05	None.

Text Info:

Text Name	Response Length	Category	Notes
SIOD	0x000F	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	Pin	P*	Pin number
uint8	drive	D	GPIO Pin drive mode

Response Parameters:

Data Type	Name	Text	Description
uint8	affected	A	1 for affected, 0 for unaffected)

Related Commands:

- gpio_get_drive (GIOD, ID=9/6)

7.2.9.6 gpio_get_drive (GIOD, ID=9/6)

Get current new drive mode for selected pins.

See Section 8.1 (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	06	09	06	None.

Text Info:

Text Name	Response Length	Category	Notes
GIOD	0x001E	GET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	pin	P*	GPIO Pin number

Response Parameters:

Data Type	Name	Text	Description
uint8	drive	D	GPIO Pin drive mode

Related Commands:

- gpio_set_drive (SIOD, ID=9/5)

7.2.9.7 gpio_set_logic (SIOL, ID=9/7)

Configure new output logic for selected pins.

See Section 8.1 (GPIO Pin Map for Supported Modules) for a pin map table showing pin availability and default assignment. Refer to the general overview in Section 7.2.9, GPIO Group (ID=9), for guidelines on how pin selection and configuration masks work.

> **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	03	09	07	None.
RSP	C0	03	09	07	None.

Text Info:

Text Name	Response Length	Category	Notes
SIOL	0x000F	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	pin	P*	GPIO Pin number
uint8	logic	L	Pin logic (1 for high, 0 for low)

Response Parameters:

Data Type	Name	Text	Description
uint8	affected	A	1 for affected, 0 for unaffected

Related Commands:

- gpio_get_logic (GIOL, ID=9/8)

7.2.9.8 gpio_get_logic (GIOL, ID=9/8)

Obtain current output logic for selected pins.

See Section 8.1 (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	03	09	08	None.

Text Info:

Text Name	Response Length	Notes
GIOL	0x000F	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	pin	Pin*	GPIO Pin number

Response Parameters:

Data Type	Name	Text	Description
uint8	logic	L	Pin logic (1 for high, 0 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)

Configure 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.

See Section 8.1 (GPIO Pin Map for Supported Modules) for a pin map table showing pin availability and default assignment. Refer to the general overview in Section 7.2.9, GPIO Group (ID=9), for guidelines on how pin selection and configuration masks work.

> **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	04	09	09	None.
RSP	C0	03	09	09	None.

Text Info:

Text Name	Response Length	Category	Notes
SIOI	0x000F	SET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	pin	P*	GPIO Pin number
uint8	rising	R	Rising-edge interrupts (set bit to enable, clear to disable)
uint8	falling	F	Falling-edge interrupts (set bit to enable, clear to disable)

Response Parameters:

Data Type	Name	Text	Description
uint8	affected	A	1 for affected, 0 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)

Obtain current edge detection interrupt settings on selected pins.

See Section 8.1 (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	0A	None.
RSP	C0	04	09	0A	None.

Text Info:

Text Name	Response Length	Category	Notes
GIOI	0x0014	GET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	pin	P*	GPIO Pin number

Response Parameters:

Data Type	Name	Text	Description
uint8	rising	R	Rising-edge interrupts (1 to enable, 0 to disable)
uint8	falling	F	Falling-edge interrupts (1 to enable, 0 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)

Configure new PWM output behavior for selected channel.

EZ-Serial provides two dedicated PWM output pins (**PWM0, PWM1**). You can enable PWM output on any of the two PWM channels using this API command. PWM channels are controlled via independent 8MHZ~250kHZ clocks, and can each use separate dutysettings available from 0% ~100% with 1/64 steps 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. See Section 8.1 (GPIO Pin Map for Supported Modules) for a pin map table showing pin availability and default assignment.

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. Refer to Section 3.1.5 (How to Manage Sleep States) for further detail.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	08	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 (2, 5)
uint8	enable	E*	Enable PWM output (0 to disable, 1 to enable)
uint8	polarity	P*	Output Polarity • 0: High • 1: Low
uint16	clock	F*	Clock frequency value: • 8000KHz ~ 250kHz
uint8	duty percentage	D*	PWM duty value: • 0% <= duty <= 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)

Obtain current PWM output behavior for selected channel.

See Section 8.1 (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	09	09	0C	None.

Text Info:

Text Name	Response Length	Category	Notes
GPWM	0x0027	GET	None.

Command Arguments:

Data Type	Name	Text	Description
uint8	channel	N*	Channel number (2, 5)

Response Parameters:

Data Type	Name	Text	Description
uint8	enable	E	Enable PWM output (0 to disable, 1 to enable)
uint8	polarity	P	Output Polarity ● 0: High ● 1: Low
uint16	clock	F	Clock frequency value: ● 8000KHz ~ 250kHz
uint8	duty percentage	D	PWM duty value: ● duty value

Related Commands:

- gpio_set_pwm_mode (SPWM, ID=9/11)

7.2.10 CYSPP Group (ID=10)

CYSPP methods relate to the Infineon Serial Port 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)
- p_cyspp_set_packetization (.CYSPPSK, ID=10/7)
- p_cyspp_get_packetization (.CYSPPGK, ID=10/8)

Events within this group are documented in Section 7.3.10 CYSPP Group (ID=10).
You can find further details and examples concerning CYSPP operation here:

- Section 2.4.5 (Using CYSPP Mode)
- Section 3.1.5.2 (Configuring the CYSPP Data Mode Sleep Level)
- Section 3.2 (Cable Replacement Examples with CYSPP)

7.2.10.1 p_cyspp_check (.CYSPPCHECK, ID=10/1)

Check 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)

Activate 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.

Refer to Section 2.4.5.9 (CYSPP State Machine) for details about how CYSPP moves between different operational states.

Binary Header:

	Type	Length	Group	ID	Notes
CMD	C0	00	0A	02	None.
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)

Configure new CYSPP behavior settings.

Use this command to control how CYSPP behaves. You can find example usage and practical explanations of how these settings affect behavior in Section 2.4.5 (Using CYSPP Mode) and Section 3.2 (Cable Replacement Examples with 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 and keep their original handles, but those inside the CYSPP attribute range will be hidden an unusable by connected clients. This will remain in effect until you enable the profile again or assert the **CYSPP** pin.

> **Note:** If `server_security parameter changed, there will be an error code (0x111) generated, the settings will be available after reboot.`

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 (Infineon Semiconductor)
uint32	local_key	L	Local connection key to present while advertising (peripheral role)
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	Maximum sleep level while connected with open CYSPP data pipe: ● 0 = Sleep disabled ● 1 = Normal sleep when possible ● 2 = Deep sleep when possible (factory default) ● **NOTE:** System-wide sleep overrides this if it is set to a lower level
uint8	server_security	S	CYSPP server security requirement to allow writing CYSPP data from a client: ● 0 = No security required (factory default) ● 1 = Encryption required ● 2 = Authentication required ● 3 = Encryption and Authentication 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)

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
- gap_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:

- Section 2.4.5 (Using CYSPP Mode)
- Section 3.1.5.2 (Configuring the CYSPP Data Mode Sleep Level)
- Section 3.2 (Cable Replacement Examples with CYSPP)

7.2.10.4 p_cyspp_get_parameters (.CYSPPGP, ID=10/4)

Obtain 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 (Infineon Semiconductor)
uint32	local_key	L	Local connection key to present while advertising (peripheral role)
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	Maximum sleep level while connected with open CYSPP data pipe: ● 0 = Sleep disabled ● 1 = Normal sleep when possible ● 2 = Deep sleep when possible (factory default) ● **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 = Authentication required ● 3 = Encryption and Authentication required

Data Type	Name	Text	Description
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)

Configure new preset attribute handles for CYSPP central/client operation.

Use this command to specify the remote GATT server handles manually 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	0x0014	0x0017	0x001A
Configuration	0x0015	0x0018	0x001B

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, whether or not you have configured the **client_flags** setting to require acknowledged data using 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.

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)

Obtain 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)

1.1.1.1 p_cyspp_set_packetization (.CYSPPSK, ID=10/7)

Control how 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 (i.e. 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 microseconds to transfer. EZ-Serial checks for new bytes at least every 20 microseconds and processes whatever is available. Because of 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 of the data could fit in a single packet (e.g., 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 as soon as 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 larger 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 how many 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.

- **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 how long the packet will be.

  The EOP byte defaults to 0x0D (the carriage return byte, often expressed as '`\r`' in 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, be sure to choose a *[wait]* value that allows the target packet length to be filled under normal operating conditions. Table 7-4 below contains "safe" wait values for 20-byte packets at common baud rates for reference.

Table 7-4. Common UART Timing for 20-Byte Packets

Baud Rate	Single Bit Duration	20 Bytes at 8/N/1 (200 Bits)	Safe Wait Value Example
300	3.333ms	~667ms	800ms (0x320)

9600	104 us	~21 ms	32 ms (0x20)
192	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)
1000000	1.09 us	217 us	1 ms (0x01)

The single-bit duration for any baud rate can be calculated in microseconds 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 allowance for 200 bits.

NOTE: If the packet length used in Anticipate, Fixed, Variable, or End-of-Packet modes exceeds the GATT MTU usable payload size (20 bytes on many platforms), then 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	04	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: ● 0 = Immediate: transmit incoming data as soon as possible ● 1 = Anticipate: wait 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), only support mode 0 and mode 1.
uint8	wait	W	Anticipation delay (uinit: 10milliseconds), used only in “Anticipate” mode: ● Minimum = 0x01 (10 millisecond) ● Maximum = 0x0D (130 millisecond) ● **NOTE:** Factory default is 0x1 (10 milliseconds)
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	End-Of-Packet byte: Note:Factory default is 0x0D(‘\r’ carriage return)

Response Parameters:

None.

Related Commands:

- p_cyspp_get_packetization (.CYSPPGK, ID=10/8)

1.1.1.2 p_cyspp_get_packetization (.CYSPPGK, ID=10/8)

Obtain 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	0x001D	GET	None.

Command Arguments:

None.

Response Parameters:

Data Type	Name	Text	Description
uint8	mode	M	Packetization mode: ● 0 = Immediate: transmit incoming data as soon as possible ● 1 = Anticipate: wait 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)
uint8	wait	W	Anticipation delay (unit: 10 milliseconds), used only in “Anticipate” mode: ● Minimum = 0x01 (10 millisecond) ● Maximum = 0x0D (130 millisecond) ● **NOTE:** Factory default is 0x1 (10 milliseconds)
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)

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 Section 7.3.11 CYCommand Group (ID=11).

7.2.11.1 p_cycommand_set_parameters (.CYCOMSP, ID=11/1)

Configure 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 all of 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 method will cause EZ-Serial to hide the relevant GATT database attributes from client discovery. All other visible attributes will remain the same and keep their original handles, but those inside the CYCommand attribute range will be hidden an 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 = 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
uint8	challenge	C	CYCommand challenge type ● 0 = None (factory default) ● 1 = Passphrase

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 = Authentication required ● 3 = Encryption and Authentication 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:

- Section 3.3 Remote Control Examples with CYCommand)

7.2.11.2 p_cycommand_get_parameters (.CYCOMGP, ID=11/2)

Obtain 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
uint8	challenge	C	CYCommand challenge type ● 0 = None (factory default) ● 1 = Passphrase

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

Related Commands:

- p_cycommand_set_parameters (.CYCOMSP, ID=11/1)

7.2.12 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 Section 7.3.12, iBeacon Group (ID=12).

7.2.12.1 p_ibeacon_set_parameters (.IBSP, ID=12/1)

Configure new iBeacon behavior.

For details on iBeacon broadcasting, refer to 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: • **NOTE:** Factory default is 0x0131 (Infineon Semiconductor)
uint8	major	J	iBeacon 16-bit major value: • **NOTE:** Factory default is 0x0001
uint8	minor	N	iBeacon 16-bit minor value: • **NOTE:** Factory default is 0x0001
uint8a	uuid	U	iBeacon UUID (must contain 16 bytes of data): • **NOTE:** Factory default is E2C56DB5-DFFB-48D2-B060-D0F5A71096E0 (AirLocate) **NOTE:** uint8a data type requires one prefixed “length” byte before binary parameter payload
uint8_t	Tx Power	T	A TX power level in 2's compliment, indicating the signal strength one meter from the device.

Data Type	Name	Text	Description
			• **NOTE: Factory default is 0xC0 (Convert it to int8_t by user).**

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:

- Section 3.9.1 (How to Configure iBeacon Transmissions)

7.2.12.2 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: • **NOTE:** Factory default is 0x0131 (Infineon Semiconductor)
uint8	major	J	iBeacon 16-bit major value: • **NOTE:** Factory default is 0x0001
uint8	minor	N	iBeacon 16-bit minor value: • **NOTE:** Factory default is 0x0001
uint8a	uuid	U	iBeacon UUID (must contain 16 bytes of data): • **NOTE:** Factory default is E2C56DB5-DFFB-48D2-B060-D0F5A71096E0 (AirLocate) **NOTE:** uint8a data type requires one prefixed “length” byte before binary parameter payload
uint8_t	Tx Power	T	A TX power level in 2's compliment, indicating the signal strength one meter from the device. • NOTE: Factory default is 0xC0 (Convert it to int8_t by user)..

Related Commands:

- p_ibeacon_set_parameters (.IBSP, ID=12/1)

7.2.13 Eddystone Group (ID=13)

Eddystone methods relate to Eddystone beacon setup 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 Section 7.3.13, Eddystone Group (ID=13).

7.2.13.1 p_eddystone_set_parameters (.EDDYSP, ID=13/1)

Configure new Eddystone beacon behavior.

For details on Eddystone frame types and data, refer to 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) ● NOTE: Factory default value results in http://www.Infineon.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:

- Section 3.9.2 (How to Configure Eddystone Transmissions)

7.2.13.2 p_eddystone_get_parameters (.EDDYGP, ID=13/2)

Obtain 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) ● NOTE: Factory default value results in http://www.Infineon.com/ **NOTE:** uint8a data type requires one prefixed “length” byte before binary parameter payload

Related Commands:

- p_eddystone_set_parameters (.EDDYSP, ID=13/1)

7.3 API Events

All events implemented in the API protocol are described in detail below. API commands and responses are documented separately in Section 7.2 (API Commands and Responses).

A master list of all possible error codes appearing in certain events can be found in Section 7.4 (Error Codes).

Commands and responses are broken down into the following groups:

- Protocol Group (ID=1)
- System Group (ID=2)
- DFU 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 Section 7.2.1, Protocol Group (ID=1).

7.3.2 System Group (ID=2)

System methods relate to the core device, describing things like boot, device address info, 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 Section 7.2.2, 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
uint32	stack	S	BLE stack version number

Data Type	Name	Text	Description
uint16	protocol	P	API protocol version number
uint8	hardware	H	Hardware identifier: • 0x01 = CYBLE-01201X-X0 • 0x02 = CYBLE-014008-00 • 0x03 = CYBLE-022001-00 • 0x04 = CYBLE-2X20XX-X1 • 0x05 = CYBLE-2120XX-X0 • 0x06 = CYBLE-212020-01 • 0x07 = CYBLE-214009-00 • 0x08 = CYBLE-214015-01 • 0x09 = CYBLE-222005-00 • 0x0A = CYBLE-222014-01 • 0x0B = CYBLE-224110-00 • 0x0C = CYBLE-224116-01 • 0x40 = CYW20822-P4TAI040_P4EPI040
uint8	cause	C	Cause of boot event: • 0x01 = Hardware power-on/reset • 0x02 = Wake from hibernation mode • 0x03 = Reserved • 0x04 = Software reboot via API command • 0x05 = Factory reset completed • 0x06 = DFU process completed with update • 0x07 = DFU process canceled without update
macaddr	address	A	Bluetooth address

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. Refer to Section 7.4 (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 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 of the 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 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 Infineon 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
uint16	offset	O	Blob start offset

Data Type	Name	Text	Description
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 DFU Group (ID=3)

DFU methods relate to the firmware update process, using either wired UART or over-the-air GATT-based firmware transfer.

> **NOTE:** DFU 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:

- dfu_boot (DFUE, ID=3/1)

Commands within this group are documented in Section 7.2.3, DFU Group (ID=3).

7.3.3.1 dfu_boot (DFUE, ID=3/1)

Booted into DFU mode.

This event indicates that the system is ready to receive a new firmware image from an external host (UART).

- .

> **NOTE:** In DFU mode, the UART interface default operates at 115200 baud, 8/N/1 with no flow control, the user can change parameters by system_set_uart_parameters (STU, ID=2/25).

Binary Header:

Type	Length	Group	ID	Notes
80	04	03	01	None.

Text Info:

Text Name	Event Length	Notes
DFUE	0x0019	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	mode	R	DFU mode: • 0 = successfully entered DFU mode. • 1 = Timeout. • 2 = Only Security check fail • 3 = DFU process more than 90s.

Related Commands:

- dfu_reboot (/CDFU, ID=3/1)

Related Events:

- system_boot (BOOT, ID=2/1)

Example Usage:

- Section 3.11.2 (How to Update Firmware Using the DFU Bootloader)

7.3.4 GAP Group (ID=4)

GAP methods relate to the Generic Access Protocol layer of the Bluetooth stack, which includes management of scanning, advertising, connection establishment, and connection maintenance.

Events within this group are listed below:

- gap_whitelist_entry (WL, ID=4/1)
- gap_adv_state_changed (ASC, ID=4/2)
- gap_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_phy_updated (PU, ID=4/9)

Commands within this group are documented in Section 7.3.4, 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

Data Type	Name	Text	Description
uint8	reason	R	Reason for state change: • 0 = User command • 1 = Reserved • 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 gap_scan_state_changed (SSC, ID=4/3)

Indicates that the module has started or stopped scanning, due to a scheduled timeout or 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 = Active
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 enable 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	0D-2C	04	04	Variable-length event payload, minimum of 13 (0x0D), maximum of 44 (0x2C)

Text Info:

Text Name	Event Length	Notes
S	0x0030-0x006E	Variable-length event payload, minimum of 48 (0x30), maximum of 110(0x6E)

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 • 5 = Extended advertisement packet • 6 = Extended scan response packet • 7 = Periodic advertisement packet
macaddr	address	A	Bluetooth address
uint8	address_type	T	Address type: • 0 = Public • 1 = Random/private
int8	rssi	S	RSSI
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
uint8_t	primary PHY	P	Primary PHY: • 1.: 1M • 3: Coded PHY
uint8_t	secondary PHY	C	Secondary PHY: • 0: No packet on secondary PHY • 1. 1M • 2. 2M • 3. Coded PHY

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:

- Section 3.5.1 (How to Scan 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
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_send_connupdate_response (/CUR, ID=4/4)
- 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:

- Section 3.5.3 (How to Connect to a Peripheral Device)

7.3.4.6 gap_disconnected (DIS, ID=4/6)

Connection to a remote device has closed.

For a list of possible disconnection reasons, refer to the 0x900 range of codes in Section 7.4.1 (EZ-Serial System Error Codes). These are the most common reasons:

- 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:

- Section 3.5.5 (How to Disconnect 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_send_connupdate_response (/CUR, ID=4/4) API command. An argument of “0” for that command will accept, and non-zero will reject.

> **NOTE:** This event and the gap_send_connupdate_response (/CUR, ID=4/4) 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 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)
- gap_send_connupdate_response (/CUR, ID=4/4)

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)
- gap_send_connupdate_response (/CUR, ID=4/4)

Related Events:

- gap_connection_update_requested (UCR, ID=4/7)

7.3.4.9 gap_phy_updated(PU, ID=4/9)

Details about aPHY information.

Binary Header:

Type	Length	Group	ID	Notes
80		04	09	None.

Text Info:

Text Name	Event Length	Notes
PU	0x0012	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint8	TX phy	T	PHY: • 1 = 1M • 2 = 2M • 3 = Coded
uint8	RX phy	R	PHY: • 1 = 1M • 2 = 2M • 3 = Coded

Related Commands:

- gap_phy_update (/UP, ID=4/37)

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 Section 7.2.6 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

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 unacknowledged write operations Acknowledged writes require two full connection intervals to complete: one for the data transfer from client to server, and one 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 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 connection from which confirmation came
uint16	attr_handle	H	Attribute handle use 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, refer to Section 3.6.1 (How to Define 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. Be sure to remove the handle and absolute group end if you are directly copying the content from these output lines into new commands by hand.

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*

Data Type	Name	Text	Description
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*
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 Section 7.2.6 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
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 (16-bit, 32-bit, or 128-bit) **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:

- Section 3.7.1 (How to Discover 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 the Related Commands list below for specific commands which trigger this event.

Binary Header:

Type	Length	Group	ID	Notes
80	03	06	02	None.

Text Info:

Text Name	Event Length	Notes
RPC	0x000D	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)

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:

- Section 3.7.1 (How to Discover 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 which operation 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	05-19	06	03	Variable-length event payload, minimum of 5 (0x05), maximum of 25 (0x19)

Text Info:

Text Name	Event Length	Notes
D	0x0016-0x003E	Variable-length event payload, minimum of 22 (0x16), maximum of 62 (0x3E)

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 (0-20 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 Section 7.2.7 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 as a result 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	07	07	01	None.

Text Info:

Text Name	Event Length	Notes
B	0x001B	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	handle	B	Bonded device handle (1-3)
macaddr	address	A	Bluetooth address
uint8	type	T	Address type: • 0 = Public • 1 = Random/private

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_security_parameters (SSBP, ID=7/11) 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: ● (Ignored in current release due to internal BLE stack functionality, set to 1 always)
uint8	keysize	K	Encryption key size (7-16), value 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)
- smp_set_security_parameters (SSBP, ID=7/11)

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 plaintext and encrypted status during the pairing process. Indicates encrypted (S=01) for LTK encryption only not for STK encryption (LE legacy pairing)

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_security_parameters (SSBP, ID=7/11) 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 display (should be displayed to 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)

7.3.8 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_connection_requested (LCR, ID=8/1)
- l2cap_connection_response (LC, ID=8/2)
- l2cap_data_received (LD, ID=8/3)
- l2cap_disconnected (LDIS, ID=8/4)
- l2cap_rx_credits_low (LRCL, ID=8/5)
- l2cap_tx_credits_received (LTCR, ID=8/6)
- l2cap_command_rejected (LREJ, ID=8/7)

Commands within this group are documented in Section 7.2.8, L2CAP Group (ID=8).

7.3.8.1 l2cap_connection_requested (LCR, ID=8/1)

Received an L2CAP connection request.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Binary Header:

Type	Length	Group	ID	Notes
80	0B	08	01	None.

Text Info:

Text Name	Event Length	Notes
LCR	0x002C	None.

Event Parameters:

Data Type	Name	Text	Description
uintu	conn_handle	C	Connection handle
uint16	channel	N	Channel ID
uint16	local	L	Local device Protocol Service Multiplexer (PSM)
uint16	mtu	M	Maximum Transmission Unit (MTU)
uint16	mps	P	Maximum Payload Size (MPS)
uint16	credits	Z	Credits

Related Commands:

- l2cap_connect (/LC, ID=8/1)
- l2cap_send_connreq_response (/LCR, ID=8/4)

Related Events:

- l2cap_connection_response (LC, ID=8/2)

7.3.8.2 l2cap_connection_response (LC, ID=8/2)

Received a response to a transmitted L2CAP connection request.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Binary Header:

Type	Length	Group	ID	Notes
80	0B	08	02	None.

Text Info:

Text Name	Event Length	Notes
LC	0x002B	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	response	R	Response
uint16	channel	N	Channel
uint16	mtu	M	Maximum Transmission Unit (MTU)
uint16	mps	P	Maximum Payload Size (MPS)
uint16	credits	Z	Credits

Related Commands:

- l2cap_connect (/LC, ID=8/1)
- l2cap_send_connreq_response (/LCR, ID=8/4)

Related Events:

- l2cap_connection_requested (LCR, ID=8/1)

7.3.8.3 l2cap_data_received (LD, ID=8/3)

Received a data block from remote peer over an open L2CAP channel.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Binary Header:

Type	Length	Group	ID	Notes
80	04	08	03	Variable-length event payload, value specified is minimum

Text Info:

Text Name	Event Length	Notes
LD	0x000D	Variable-length event payload, value specified is minimum

Event Parameters:

Data Type	Name	Text	Description
uint16	channel	N	Channel ID
longuint8a	data	D	Data **NOTE:** longuint8a data type requires two prefixed “length” bytes before binary parameter payload

Related Commands:

- l2cap_send_data (/LD, ID=8/6)

Related Events:

- l2cap_connection_requested (LCR, ID=8/1)
- l2cap_connection_response (LC, ID=8/2)
- l2cap_rx_credits_low (LRCL, ID=8/5)

7.3.8.4 l2cap_disconnected (LDIS, ID=8/4)

Previously open L2CAP channel to a remote device has been disconnected.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Binary Header:

Type	Length	Group	ID	Notes
80	05	08	04	None.

Text Info:

Text Name	Event Length	Notes
LDIS	0x0018	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	channel	N	Channel ID
uint16	reason	R	Reason for disconnection

Related Commands:

- l2cap_connect (/LC, ID=8/1)
- l2cap_disconnect (/LDIS, ID=8/2)

- l2cap_register_psm (/LRP, ID=8/3)

Related Events:

- l2cap_connection_requested (LCR, ID=8/1)
- l2cap_connection_response (LC, ID=8/2)

7.3.8.5 l2cap_rx_credits_low (LRCL, ID=8/5)

Open L2CAP channel connection has crossed the defined threshold for low remaining credits.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

This event occurs on the receiving side and indicates that more credits must be sent to the transmitting device via the l2cap_send_credits (/LSC, ID=8/5) API command to ensure that the transmitting device will be able to continue to send data.

Binary Header:

Type	Length	Group	ID	Notes
80	05	08	05	None.

Text Info:

Text Name	Event Length	Notes
LRCL	0x0018	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	channel	N	Channel ID
uint16	credits	Z	Credits remaining

Related Commands:

- l2cap_send_credits (/LSC, ID=8/5)

7.3.8.6 l2cap_tx_credits_received (LTCR, ID=8/6)

Open L2CAP channel connection received more TX credits from the remote peer.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

This event occurs on the transmitting side, and indicates that it is safe to send more data to the remote device with the l2cap_send_data (/LD, ID=8/6) API command.

Binary Header:

Type	Length	Group	ID	Notes
80	05	08	06	None.

Text Info:

Text Name	Event Length	Notes
LTCR	0x0018	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle

Data Type	Name	Text	Description
uint16	channel	N	Channel ID
uint16	credits	Z	Credits received

Related Commands:

- l2cap_send_data (/LD, ID=8/6)

7.3.8.7 l2cap_command_rejected (LREJ, ID=8/7)

L2CAP command has been rejected by the remote peer.

> **NOTE:** L2CAP communication features within EZ-Serial are only available on devices with 256K of flash memory. The behavior described in this section will not function on devices with only 128K of flash.

Binary Header:

Type	Length	Group	ID	Notes
80	05	08	07	None.

Text Info:

Text Name	Event Length	Notes
LREJ	0x0018	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	channel	N	Channel ID
uint16	reason	R	Reason for rejection

7.3.9 GPIO Group (ID=9)

GPIO methods relate to the physical pins on the module.

Events within this group are listed below:

- gpio_interrupt (INT, ID=9/1)

Commands within this group are documented in Section 7.2.9, 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 in order 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	09	09	01	None.

Text Info:

Text Name	Event Length	Notes
INT	0x0025	None.

Event Parameters:

Data Type	Name	Text	Description
uint8	Pin	P	GPIO Pin number
uint8	trigger	T	Triggering pin mask (set bits indicate interrupt source)
uint8	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 Infineon Serial Port Profile.

Events within this group are listed below:

- p_cyspp_status (.CYSPP, ID=10/1)

Commands within this group are documented in Section 7.2.10, 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:

- Section 3.2 (Cable Replacement Examples with CYSPP)

7.3.11 CYCommand Group (ID=11)

CYCommand methods relate to CYCommand remote configuration channel behavior.

Events within this group are listed below:

- p_cycommand_status (.CYCOM, ID=11/1)

Commands within this group are documented in Section 7.2.11 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 the purpose of 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

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 Section 7.2.12, 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 Section 7.2.13 Eddystone Group (ID=13).

7.4 Error Codes

7.4.1 EZ-Serial System Error Codes

The complete list of all result/error codes generated by EZ-Serial is contained in the table below. Refer to the command and event reference material in Section 7.2 (API Commands and Responses) and Section 7.3 (API Events) for specific details about each result within the context of the responses and events where they are triggered.

Table 7-5. EZ-Serial System Error Codes

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
0111	EZS_ERR_CORE_SYS_REBOOT_REQUIRED	System reboot required or the settings would not be updated
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)*
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

Code (Hex)	Name	Description
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)
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

Code (Hex)	Name	Description
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
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

Code (Hex)	Name	Description
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
EEEE	**EZS_ERR_UNKNOWN**	**Unknown problem** *(internal error)*

7.4.2 EZ-Serial GATT Database Validation Error Codes

The complete list of result/error codes generated by EZ-Serial during dynamic GATT database validation is contained in the table below. Refer to Section 3.6.1 (How to Define Custom Local GATT Services and Characteristics) and the documentation for the related GATT Server Group (ID=5) API command methods for detail.

Table 7-6. EZ-Serial GATT Validation Error Codes

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

Code (Hex)	Name	Description
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

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, as long as 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 the various GPIO connections provided by the EZ-Serial firmware on supported modules. It also provides details on the default boot state and what behavior to expect in different operational modes.

8.1 GPIO Pin Map for Supported Modules

The EZ-Serial firmware can be run on multiple InfineonBLE 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 Infineon evaluation modules such as CYBLE-212019-EVAL are remapped from the module pin to the evaluation kit pin in order to provide more flexibility when design with PSoC Creator and the BLE Pioneer Kit. Pins which have been remapped on evaluation modules are shown in **bold** in the table below.

Table 8-1. GPIO Pin Map on Supported Modules

Pin Name	Pin Assignment
	CYW20822

		Module
DIGITAL FUNCTIONS	**UART_RX**	P25
	UART_TX	P23
	UART_RTS	P24
	UART_CTS	P11
	CONNECTION	P13
	CYSPP	P30
	LP_MODE	P22
	UART1_TX	P33
PWM	**PWM0**	P20
	PWM1	P10
ADC	**ADC**	P9

8.2 GPIO Pin Functionality

EZ-Serial provides 11 special-function digital GPIO pins, two 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 below 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.

Table 8-2. GPIO Pin Functionality Detail

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**
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 peer device is connected. ○ **HIGH** – no remote BLE 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**
CYSPP	Input	**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. If connected to a high-impedance input pin (weaker than 5.6k pull), this pin may be used as a status indicator for software-based CYSPP mode changes. Otherwise, it should be driven externally to the desired state. **Control signal logic (active-low):** ● **LOW** – module enters CYSPP data mode. ● **HIGH** – module exits CYSPP data mode and returns to API command mode. ● **Default boot state:** ● Internally pulled **HIGH** (command mode active, CYSPP data mode inactive)	No
UART1_TX	Output	Output UART log with baud rate 115200.	
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

Pin Name	Direction	Details	Optional
P32	**Output**	**Description:** **P32 is a strap option for benign boot that, if high, instructs the M0 CPU to stop boot process and go into an idle state.** **CYSPP status indicator logic:** • **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:** • **HIGH (API commands)**	

8.2.2 PWM Output Pins

EZ-Serial provides two dedicated PWM output pins (**PWM0, PWM1**). 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 can each 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. Refer to Section 3.1.5 (How to Manage Sleep States) for further detail.

8.2.3 Analog Input Pins (ADC)

EZ-Serial provides a single dedicated ADC input pin (**ADC**) for reading analog voltages. The ADC supports an input voltage range of **0 V** minimum to **VBAT** 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 **ADC**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 helps ensure that your expectations will be met.

8.3.1 Digital Interrupt Detection

The internal chipset is capable of detecting and responding 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. In order 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 PSoC Creator and the PSoC Components within the IDE.

8.3.2 Analog-to-Digital Conversion

Similar to the previous section describing interrupt detection, the ADC operates very quickly but incurs significant processing overhead in order 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 PSoC Creator to create a custom firmware implementation.

9. Infineon GATT Profile Reference

The EZ-Serial platform makes use of a few custom GATT profiles defined by Cypress Semiconductor. The service UUIDs, characteristic UUIDs, special permissions, and overall structure are outlined here for quick reference. Much more detailed reference material can be found on the Infineon website here:

http://www.cypress.com/documentation/software-and-drivers/cypresss-custom-ble-profiles-and-services

9.1 Bootloader Profile

N/A

9.2 CYSPP Profile

The Infineon Serial Port Profile (CYSPP) provides bidirectional serial data transfer between two remote devices, each of which passes data in 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 is able to 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: UUID **65333333-A115-11E2-9E9A-0800200CA102**
      (Write without response, Notify)

      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.
        - Configuration Descriptor: UUID **0x2902**
    - **RX Flow** Characteristic: UUID **65333333-A115-11E2-9E9A-0800200CA103**
      (Indicate)

      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.
        - Configuration Descriptor: UUID **0x2902**

9.3 CYCommand Profile

The Infineon 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: UUID **65333333-A115-11E2-9E9A-0800200CA201** (Write, Indicate)

      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: UUID **65333333-A115-11E2-9E9A-0800200CA202** (Write, Indicate)

      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.
        - Configuration Descriptor: UUID **0x2902**

10. Configuration Example Reference

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 all of 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 are 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 below demonstrate how to add various GATT service and characteristic structural elements in order 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.

> **NOTE:** 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). Refer to the official Bluetooth SIG documentation or other related resources linked under each service for further detail.

> **NOTE:** Additions to and removals from the GATT structure are always stored in flash. As long as 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.

The structural definitions in this table match those created using default settings in PSoC Creator's BLE component configurator. Not all applications require the full structure provided for reference.

For additional GATT services not included here, you can use PSoC Creator to quickly build a reference definition of any service by performing the following steps:

1. Create a new project and add a BLE component, or use an example project such as "BLE_FindMe"
2. Open BLE component settings in the TopDesign.cysch schematic definition and go to the "Profiles" tab
3. Build or add the service of interest to the end of the existing GATT structure
4. Build the project to refresh generated source files, or use the "Build -> Generate Application" menu item
5. Open the **/Generated_Source/PSoC4/BLE/BLE_gatt.c** source file and find the **`cyBle_gattDB`** definition
6. Identify the range of attributes just added (typically beginning with a 0x2800 "Primary service" declaration)
7. Refer to the structure defined in code here when forming new gatts_create_attr (/CAC, ID=5/1) commands.

The internal GATT definition structure changed as of v3.20 compared to v3.10. Refer to code generated by v3.20 of the BLE Component for the closest match with EZ-Serial syntax. The 32-bit "permissions" field is split up into four bytes which correlate to:

- Read permissions (B0)
- Write permissions (B1)
- Characteristic properties (B2)

The fourth byte is not relevant for EZ-Serial GATT structural definitions.

10.2.1 Generic Access Service (0x1800)

Official documentation for this service can be found on the Bluetooth SIG Developer website.

NOTE: This service is include 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.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0018
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=002A
/CAC,T=0000,R=01,W=00,C=02,L=0040,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=012A
/CAC,T=0000,R=01,W=00,C=02,L=0002,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=042A
/CAC,T=0000,R=01,W=00,C=02,L=0008,D=
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=A62A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/SACT
```

10.2.2 Generic Attribute Service (0x1801)

Official documentation for this service can be found on the Bluetooth SIG Developer website.

NOTE: This service is include 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.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0118
/CAC,T=2803,R=01,W=00,C=20,L=0000,D=052A
/CAC,T=0000,R=00,W=00,C=20,L=0004,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/SACT
```

10.2.3 Immediate Alert Service (0x1802)

Official documentation for this service can be found on the Bluetooth SIG Developer website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0218
/CAC,T=2803,R=01,W=00,C=04,L=0000,D=062A
/CAC,T=0000,R=02,W=02,C=04,L=0001,D=
/SACT
```

10.2.4 Link Loss Service (0x1803)

Official documentation for this service can be found on the Bluetooth SIG Developer website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0318
/CAC,T=2803,R=01,W=00,C=0A,L=0000,D=062A
/CAC,T=0000,R=01,W=01,C=0A,L=0001,D=
/SACT
```

10.2.5 TX Power Service (0x1804)

Official documentation for this service can be found on the Bluetooth SIG Developer website.

```
/CAC,T=2800,R=01,W=00,C=00,L=0000,D=0418
/CAC,T=2803,R=01,W=00,C=02,L=0000,D=072A
/CAC,T=0000,R=01,W=00,C=02,L=0001,D=
/CAC,T=2902,R=01,W=01,C=0A,L=0002,D=
/SACT
```

10.2.6 Current Time Service (0x1805)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.7 Reference Time Update Service (0x1806)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.8 Next DST Change Service (0x1807)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.9 Glucose Service (0x1808)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.10 Health Thermometer Service (0x1809)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
```

```
/SACT
```

10.2.11 Device Information Service (0x180A)

In the commands below, 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.

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.12 Heart Rate Service (0x180D)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.13 Phone Alert Status Service (0x180E)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.14 Battery Service (0x180F)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.15 Blood Pressure Service (0x1810)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.16 Alert Notification Service (0x1811)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.17 Human Interface Device Service (0x1812)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.18 Scan Parameters Service (0x1813)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.19 Running Speed and Cadence Service (0x1814)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.20 Cycling Speed and Cadence Service (0x1816)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.21 Cycling Power Service (0x1818)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.22 Location and Navigation Service (0x1819)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.23 Body Composition Service (0x181B)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

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 of these. Refer to the official specification for this service on the Bluetooth SIG website for details.

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.25 Weight Scale Service (0x181D)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.26 Bond Management Service (0x181E)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.27 Continuous Glucose Monitoring Service (0x181F)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

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 Section 3.6.1 (How to Define 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.

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.29 HTTP Proxy Service (0x1823)

Official documentation for this service can be found on the Bluetooth SIG Developer 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=
/SACT
```

10.2.30 Apple Notification Center Service (7905F431-B5CE-4E99-A40F-4B1E122D00D0)

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=
/SACT
```

Revision History

Document Title: EZ-Serial BLE Firmware Platform User Guide
Document Number: 002-11259

Revision	Issue Date	Origin of Change	Description of Change
**	06/27/2023	JROW	New user guide draft.