Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CE215124 - BLE HTTP Proxy with PSoC 6 MCU with BLE Connectivity

Objective

This example demonstrates the HTTP Proxy Client and Server operation of the Bluetooth Low Energy (BLE) PSoC Creator™ Component.

Overview

The HTTP Proxy Server and HTTP Proxy Client projects are used as a pair to demonstrate the BLE HTTP Proxy Service (HPS) operation. The HTTP Proxy Server uses one instance of the HTTP Proxy Service to simulate an HTTP Server on the BLE device. Also, the HTTP Proxy Server operates with other devices that implement the HTTP Proxy Client Role. To conserve power, the device switches to Deep Sleep mode between the BLE connection intervals.

An HTTP Proxy Client is designed to operate with the HTTP Proxy Server that can process GET and POST HTTP requests.

Requirements

Tool: PSoC Creator 4.2 or later

Programming Language: C (Arm® GCC 5.4-2016-q2-update or later)

Associated Parts: All PSoC® 6 MCU with BLE Connectivity (PSoC 6 BLE) parts

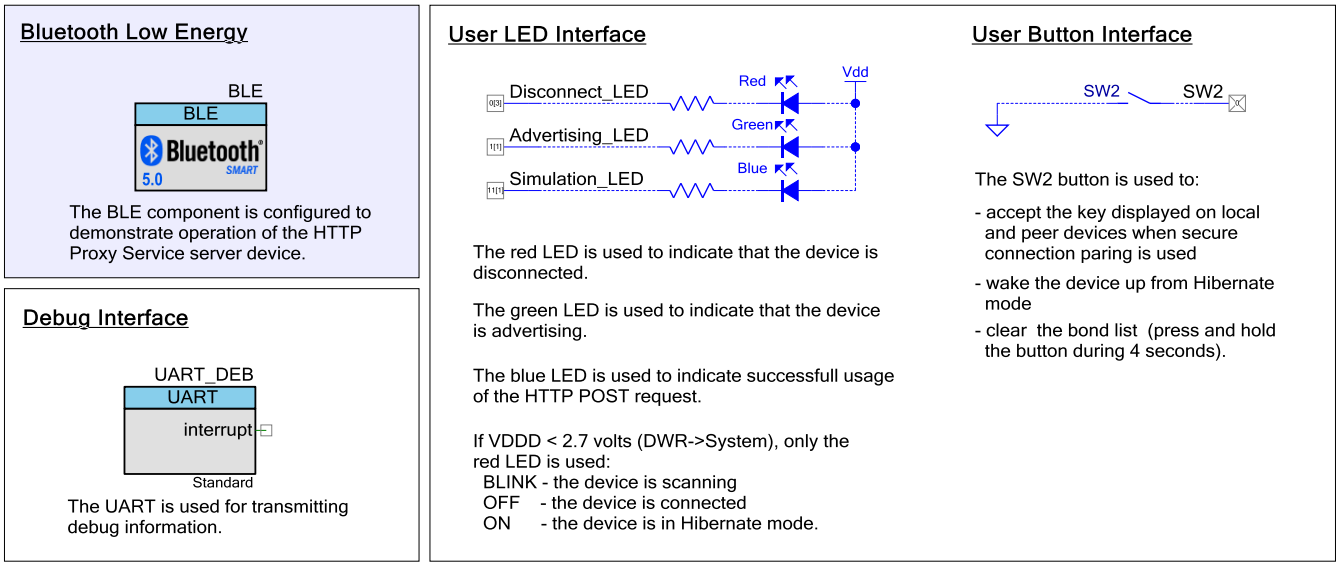
Related Hardware: CY8CKIT-062 PSoC 6 BLE Pioneer Kit

HTTP Proxy Server Code Example

Design

Figure 1 shows the top design schematic.

Figure 1. BLE HTTP Proxy Server Code Example Schematic



The project demonstrates the functionality of the BLE Component configured as an HTTP Proxy Server. The project is designed to work with the HTTP Proxy Client project provided with this document, but it also can be used with CySmart.

After a startup, the device initializes the BLE Component. To operate, the component requires several callback functions to receive events from the BLE Stack. The `AppCallBack()` is used to receive general BLE events. Another callback (`HpsCallBack()`) is used to receive events specific to the service's attribute operations.

The `CY_BLE_EVT_STACK_ON` event indicates successful initialization of the BLE Stack. After receiving this event, the component starts fast advertising with the packet structure as configured in the BLE Component Customizer. After a 30-second advertising period expires, the component switches to the slow advertisement parameters. On an advertisement event timeout, if the HTTP Proxy Server is not connected to any client, the device goes to Low-Power mode (Hibernate mode) and waits for **SW2** to be pressed to wake up the device again and start advertising.

You can connect to the HTTP Proxy Server device with an HTTP Proxy Client or any BLE 4.1 or BLE 4.2 compatible device configured in the GAP Central role and capable of discovering the HTTP Proxy Service. To connect to an HTTP Proxy Server device, send a connection request to the device when the device is advertising. The blinking green LED indicates that the device is advertising. If the client is connected to the HTTP Proxy Server, the green LED will stop blinking.

Design Considerations

Using UART for Debugging

Download and install a serial port communication program. Freeware such as Bray's Terminal and PuTTY are available on the web.

1. Connect the PC and kit with a USB cable.
2. Open the device manager program in your PC, find a COM port that the kit is connected to, and note the port number.
3. Open the serial port communication program and select the previously noted COM port.
4. Configure the Baud rate, Parity, Stop bits, and Flow control information in the PuTTY configuration window. The default settings: Baud rate – 115200, Parity – None, Stop bits – 1, Flow control – XON/XOFF. These settings must match the configuration of the PSoC Creator UART component in the project.

5. Start communicating with the device as explained in the Operation section.

The UART debugging can be disabled by setting the `DEBUG_UART_ENABLED` to `DISABLED` in the *common.h* file.

Switching the CPU Cores Usage

This section describes how to switch between different CPU cores usage (Single core and Dual core) in the BLE Peripheral Driver Library (PDL) examples.

The BLE Component has the CPU Core parameter that defines the cores usage. It can take the following values:

- **Single core (Complete Component on CM0+)** – only CM0+ core will be used.
- **Single core (Complete Component on CM4)** – only CM4 core will be used.
- **Dual core (Controller on CM0+, Host and Profiles on CM4)** – both cores will be used: CM0+ for the Controller and CM4 for the Host and Profiles.

The BLE examples' structure allows easy switching between different CPU cores options.

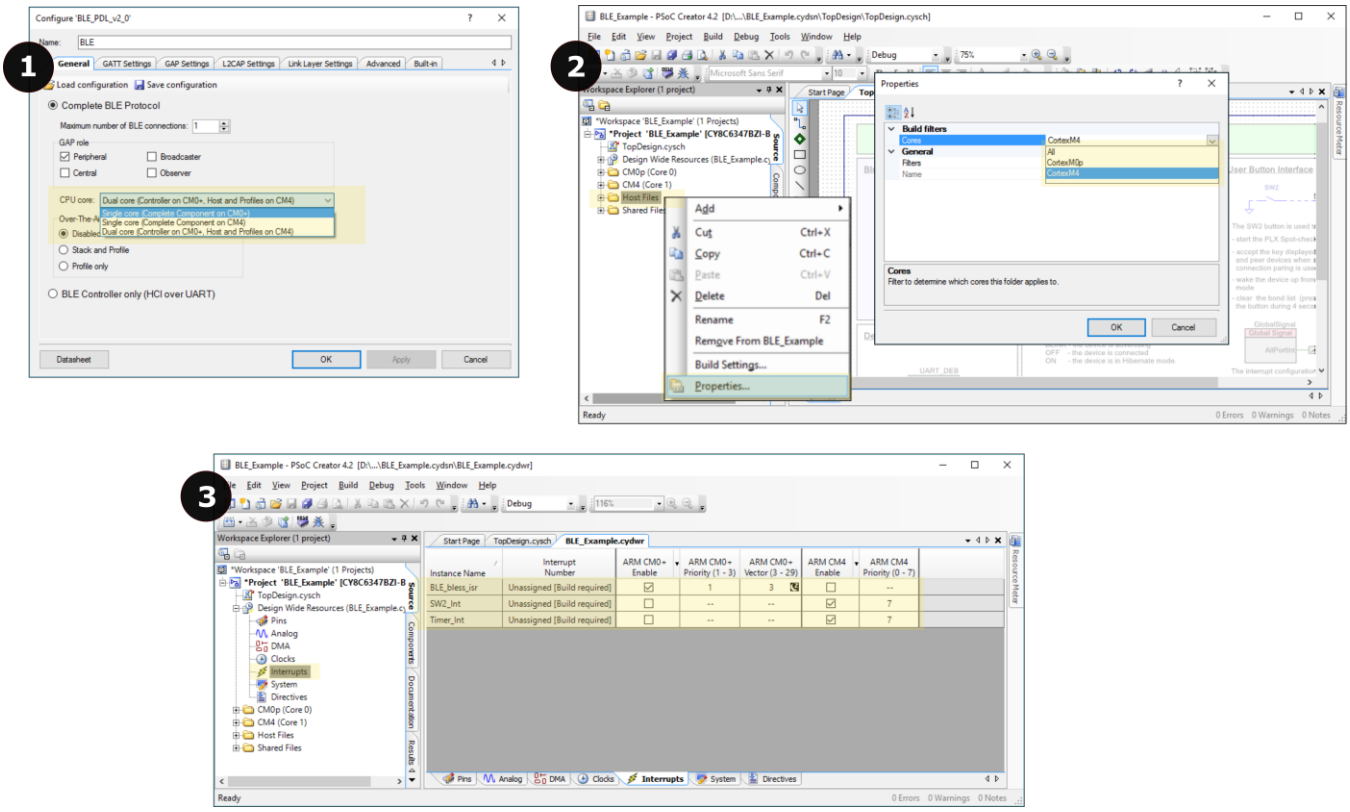
Important to remember:

- All application host-files must be run on the host core.
- The BLE Subsystem (BLESS) interrupt must be assigned to the core where the controller runs.
- All additional interrupts (SW2, MCWDT, etc.) used in the example must be assigned to the host core.

Do the following to switch the CPU cores usage:

1. In the BLE Component Customizer **General** tab, select appropriate CPU core option.
2. Change the core properties to CortexM4 or CortexC0p for the project folder Host Files based on the CPU core option selected in step 1. It should be:
    - For **Single core (Complete Component on CM0+)** option: CM0+
    - For **Single core (Complete Component on CM4)** option: **CM4**
    - For **Dual core (Controller on CM0+, Host and Profiles on CM4)** option: **CM4**
3. Assign the BLE_bless_isr and other peripheral (button – SW2, timer(s) etc.) interrupts to appropriate core in **DWR** > **Interrupts** tab.
    - For **Single core (Complete Component on CM0+)** option: BLE_bless_isr and peripheral interrupts on **CM0+**
    - For **Single core (Complete Component on CM4)** option: BLE_bless_isr and peripheral interrupts on **CM4**
    - For **Dual core (Controller on CM0+, Host and Profiles on CM4)** option: BLE_bless_isr interrupt on **CM0+,** other peripheral interrupts on **CM4**

Figure 2. Steps for Switching the CPU Cores Usage



Hardware Setup

The code example was designed for the CY8CKIT-062 PSoC 6 BLE Pioneer Kit.

Table 1 lists the pin assignments and connections required on the development board for supported kits.

Table 1. Pin Assignment

Pin Name	Development Kit	Comment
	CY8CKIT-062	
\UART_DEB:rx\	P5[0]	
\UART_DEB:tx\	P5[1]	
\UART_DEB:rts\	P5[2]	
\UART_DEB:cts\	P5[3]	
Disconnect_LED	P0[3]	The red color of the RGB LED
Advertising_LED	P1[1]	The green color of the RGB LED
Simulation_LED	P11[1]	The blue color of the RGB LED
SW2	P0[4]	

LED Behavior for V_{DDD} Voltage < 2.7 V

If the V_{DDD} voltage is set to less than 2.7 V in the DWR settings of the **System** tab, only the red LED is used. The red LED blinks to indicate that the device is advertising. The red LED is off when a device is connected to a peer device. When the device is in Hibernate mode, the red LED stays ON.

Components

Table 2 lists the PSoC Creator Components used in this example and the hardware resources used by each of the components.

Table 2. PSoC Creator Components List

Component	Hardware Resources
UART_DEB	1 SCB
BLE	1 BLE, 1 Interrupt
SW2	1 pin
Disconnect_LED, Advertising_LED, Simulation_LED	3 pins

Parameter Settings

The BLE Component is configured as an HPS Server in the GAP Peripheral role with the settings shown in Figure 3 to Figure 7.

Figure 3. General Settings

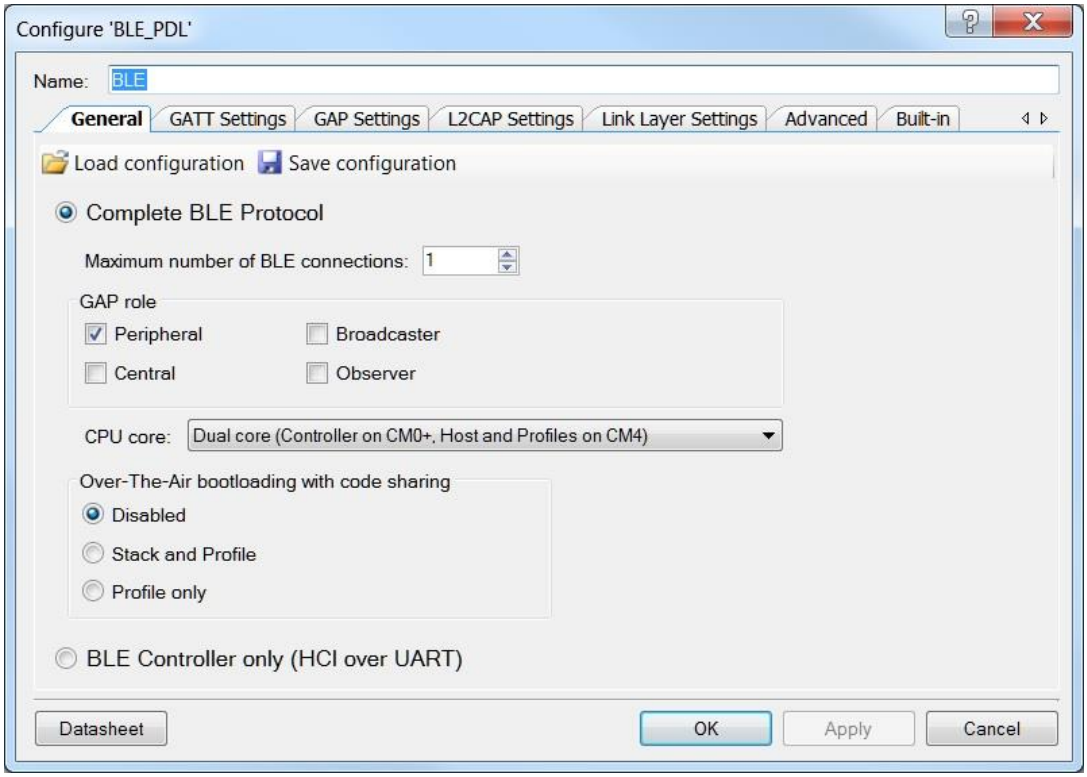


Figure 4. GATT Settings

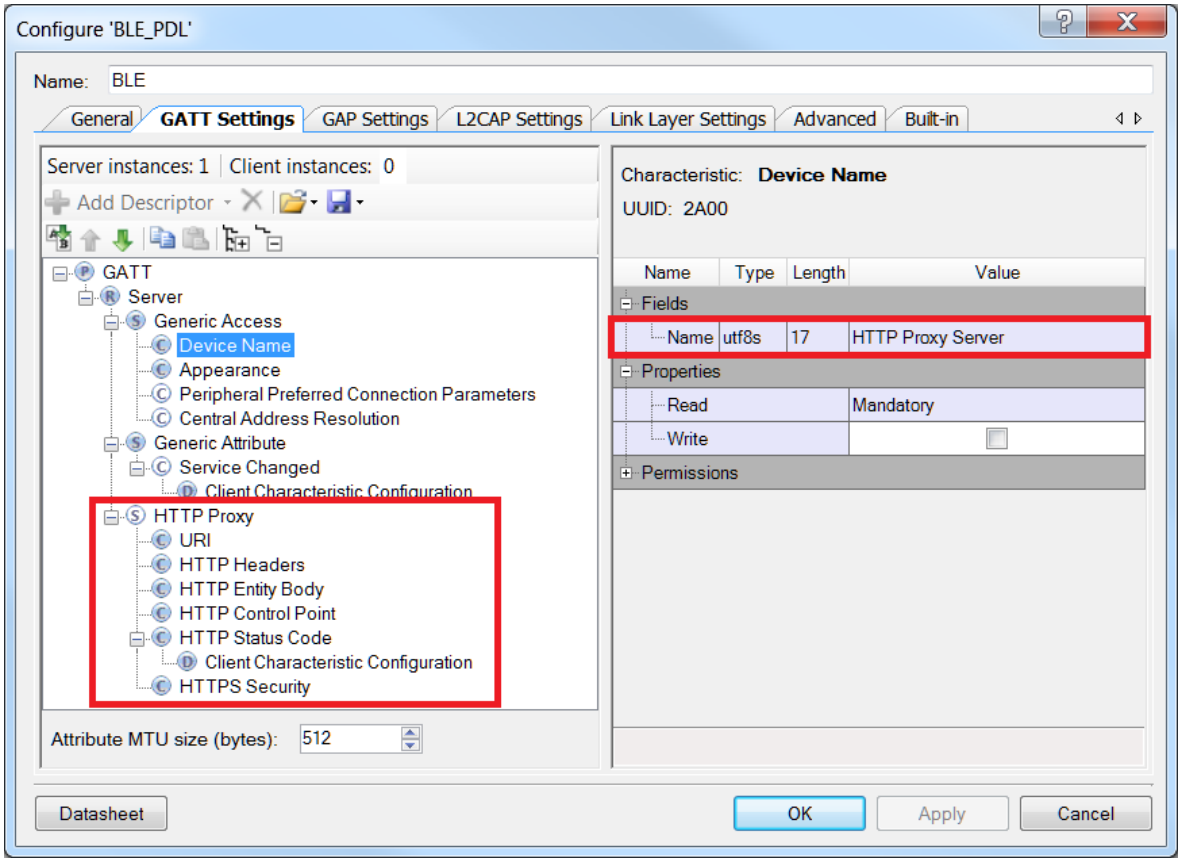


Figure 5. GAP Settings

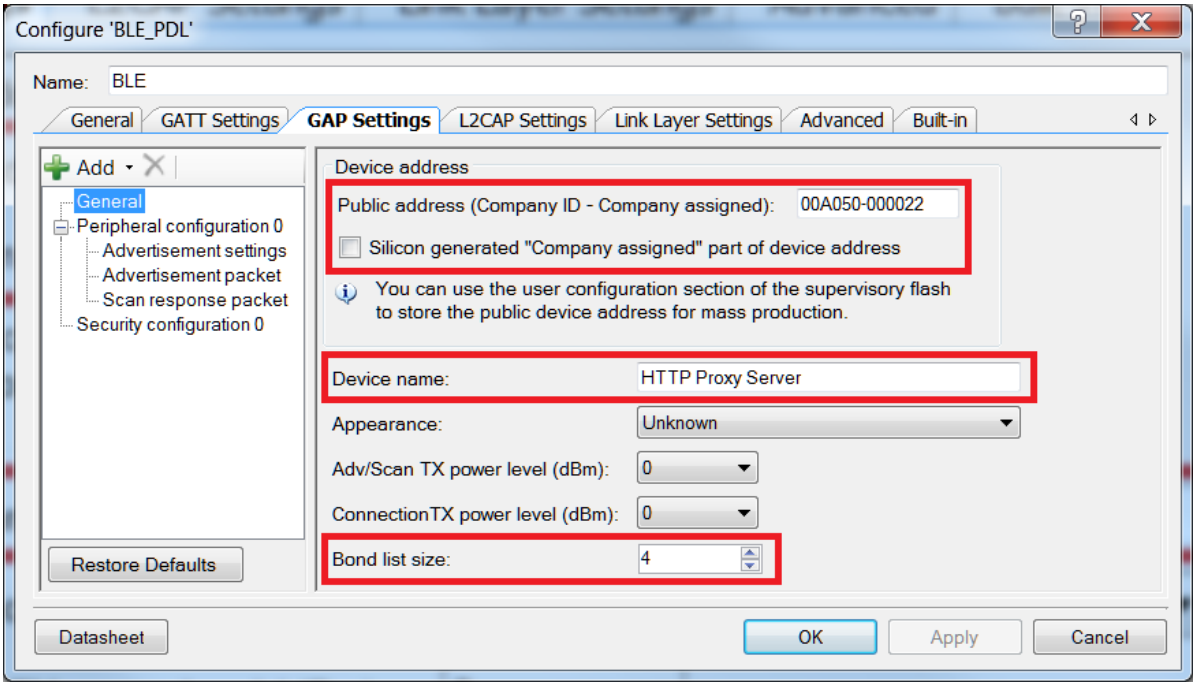
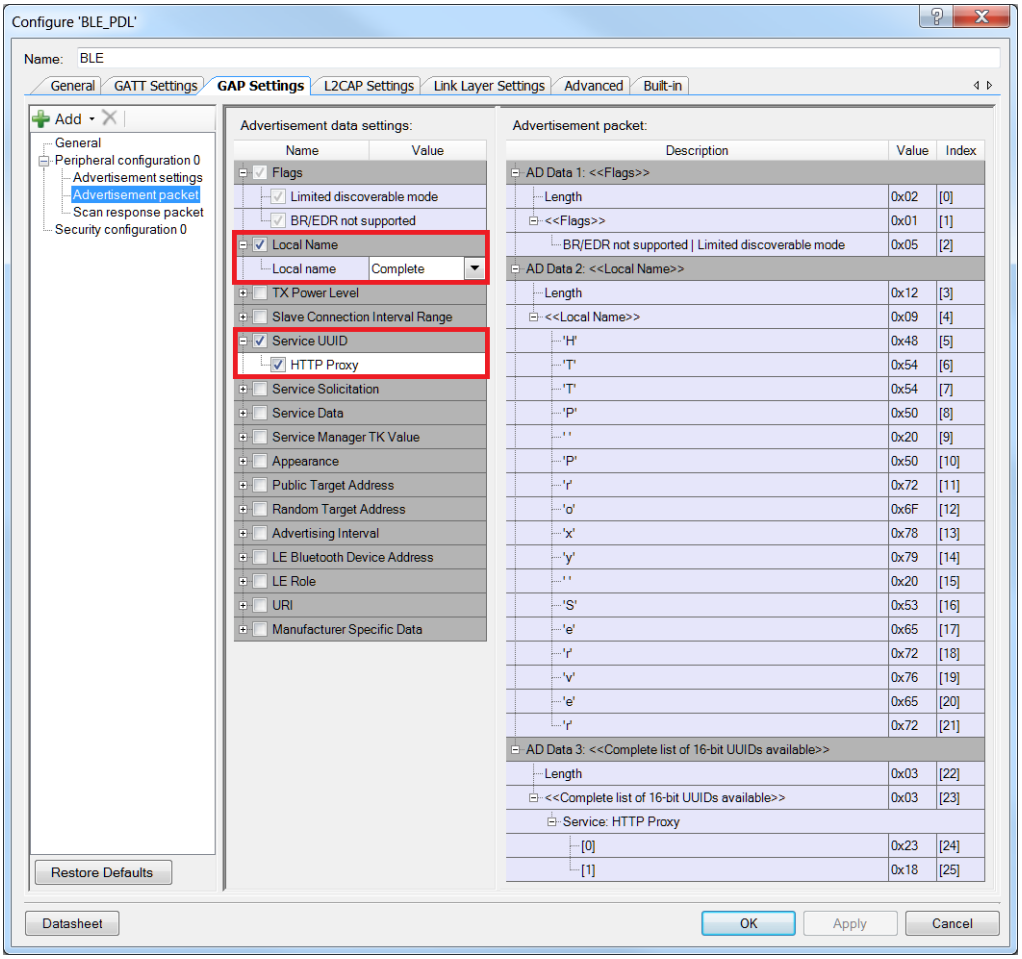
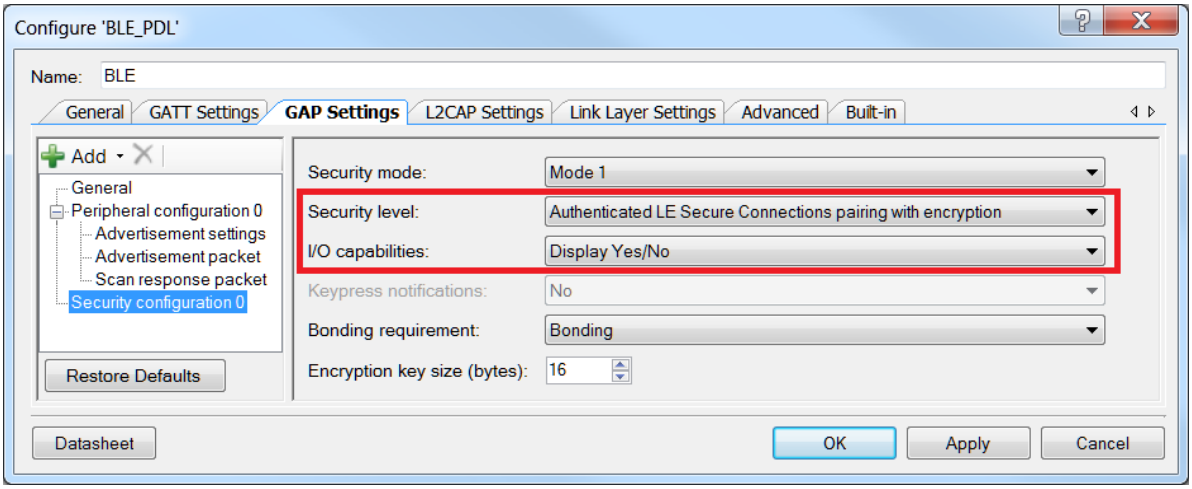


Figure 6. GAP Settings: Advertisement Setting



The Scan response packet settings are also configured to include the Local Name and all the service UUIDs into the Scan response packet.

Figure 7. GAP Settings > Security Configuration

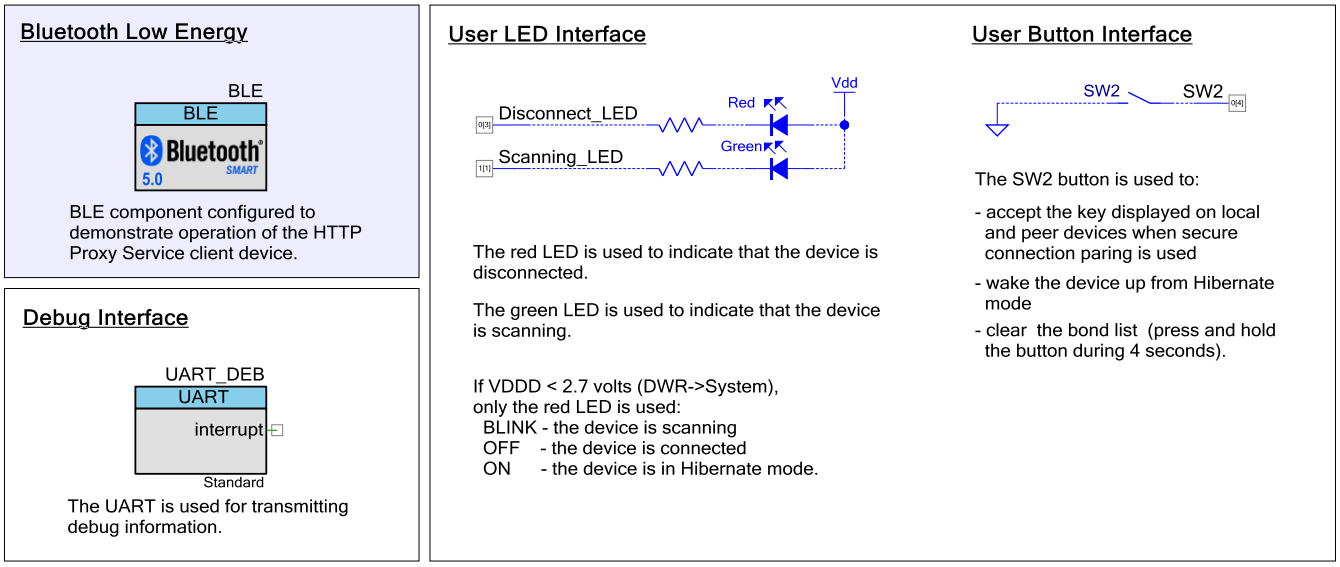


HTTP Proxy Client Code Example

Design

Figure 8 shows the top design schematic.

Figure 8. BLE HTTP Proxy Client Code-Example Schematic



The project demonstrates the functionality of the BLE HTTP Proxy Client. This project is designed to work with the BLE HTTP Proxy Server provided with this document.

The project is configured to operate using secure connection. To accept the password displayed on the HyperTerminal, press **SW2** on the CY8CKIT-062 PSoC 6 BLE Pioneer Kit or press '**y**' on the HyperTerminal. Optionally, the project can use legacy Security Mode 1 Level 3 (Authenticated pairing with encryption).

The project supports only one URI - http://www.cypress.com/led and two HTTP requests – HTTP GET and HTTP POST. The HTTP GET request is used to read the state of the blue LED on a device programmed with the BLE HTTP Proxy Server project. The HTTP POST request is used to set the state of the blue LED on a BLE HTTP Proxy Server device.

To start the project operation, build it and program it onto PSoC 6 MCU. After a startup, the device initializes the BLE Component and UART and ISR Components. In this project, two callback functions are required for the BLE operation. One callback function (`AppCallBack()`) is required for receiving general BLE events from the BLE Stack and the `HpsAppEventHandler()` required for receiving events related to the HTTP Proxy Service. The `CY_BLE_EVT_STACK_ON` event indicates successful initialization of the BLE Stack. After this event is received, the project will prompt a message asking to start scanning. The green LED will start blinking to indicate that the device has started scanning. During scanning, the project should prompt scan reports from the BLE devices that are advertising and are available for connection. After selecting the proper device from a prompted list, the project will connect to the selected device and initiate secure connection pairing. The green LED will stop blinking after the device stops scanning. When secure pairing is confirmed, the project will discover the connected Server device and configure the HPS service for notifications. Then it will prompt a message asking to send an HTTP GET request to the HTTP Proxy Server. Next, upon successful execution of the HTTP GET request, the project will ask to send an HTTP POST request to the HTTP Proxy Server.

While connected to the Server and between the connection intervals, the device is put into Sleep mode. When the device is not connected to the HTTP Proxy Server and there is no scanning in progress, the red LED will be ON to indicate that the device is in the disconnected state.

Design Considerations

Using UART for Debugging

Download and install a serial port communication program. Freeware such as Bray's Terminal and PuTTY. are available on the web.

4. Connect the PC and kit with a USB cable.
5. Open the device manager program in your PC, find a COM port that the kit is connected to, and note the port number.
6. Open the serial port communication program and select the previously noted COM port.
7. Configure the Baud rate, Parity, Stop bits, and Flow control information in the PuTTY configuration window. The default settings: Baud rate – 115200, Parity – None, Stop bits – 1, Flow control – XON/XOFF. These settings must match the configuration of the PSoC Creator UART component in the project.
8. Start communicating with the device as explained in the Operation section.

The UART debugging can be disabled by setting the `DEBUG_UART_ENABLED` to `DISABLED` in the *common.h* file.

Hardware Setup

The code example was designed for the CY8CKIT-062 PSoC 6 BLE Pioneer Kit.

Table 3 lists the pin assignments and connections required on the development board for supported kits .

Table 3. Pin Assignment

Pin Name	Development Kit	Comment
	PSoC6	
\UART_DEB:rx\	P5[0]	
\UART_DEB:tx\	P5[1]	
\UART_DEB:rts\	P5[2]	
\UART_DEB:cts\	P5[3]	The green color of the RGB LED
Disconnect_LED	P0[3]	The red color of the RGB LED
Scanning_LED	P1[1]	
SW2	P0[4]	

LED Behavior for V_{DDD} Voltage < 2.7 V

If the V_{DDD} voltage is set to less than 2.7 V in the DWR settings of the **System** tab, only the red LED is used. The red LED blinks to indicate that the device is scanning. The red LED is off when the device is connected to a peer device. When the device is in Hibernate mode, the red LED stays ON.

Components

Table 4 lists the PSoC Creator components used in this example and the hardware resources used by each of the components.

Table 4. PSoC Creator Components List

Component	Hardware Resources
UART_DEB	1 SCB
BLE	1 BLE, 1 Interrupt
SW2	1 pin
Disconnect_LED Scanning_LED	2 pins

Parameter Settings

The BLE Component is configured as an HPS Client in the GAP Central role with the settings shown in Figure 9 to Figure 13.

Figure 9. General Settings

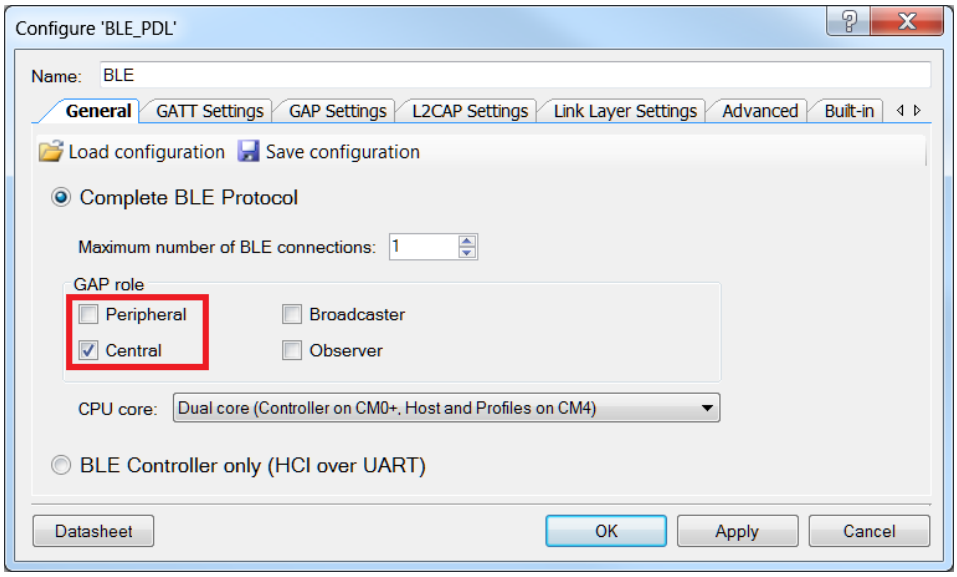


Figure 10. GATT Settings

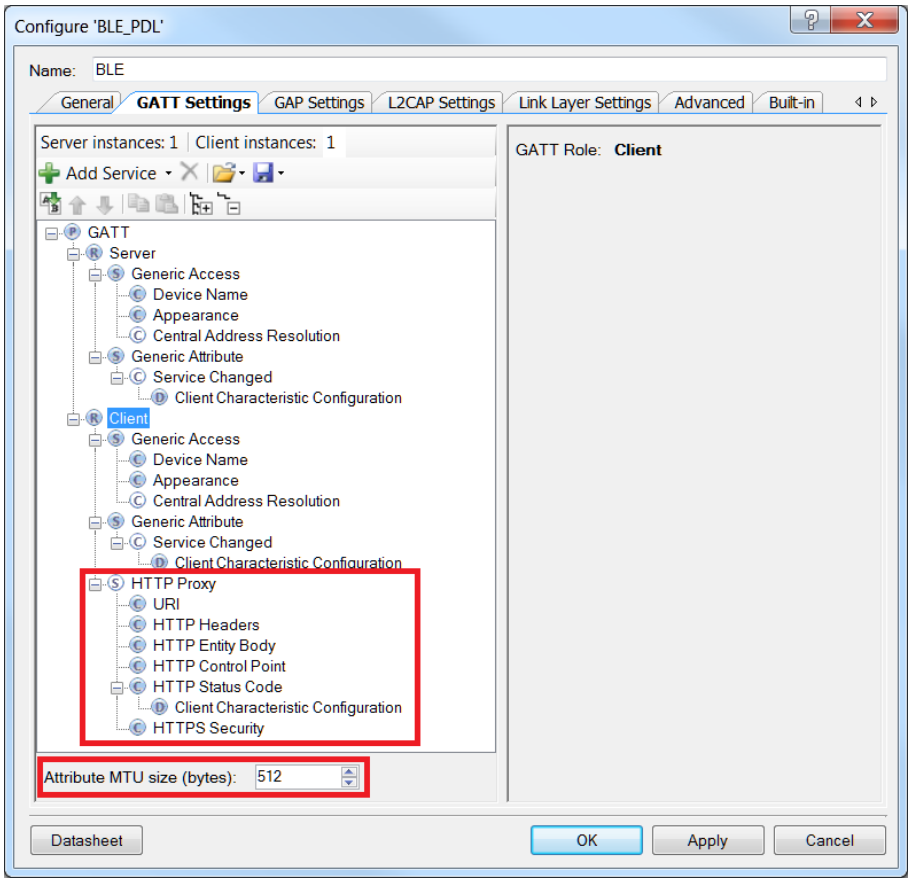


Figure 11. GAP Settings

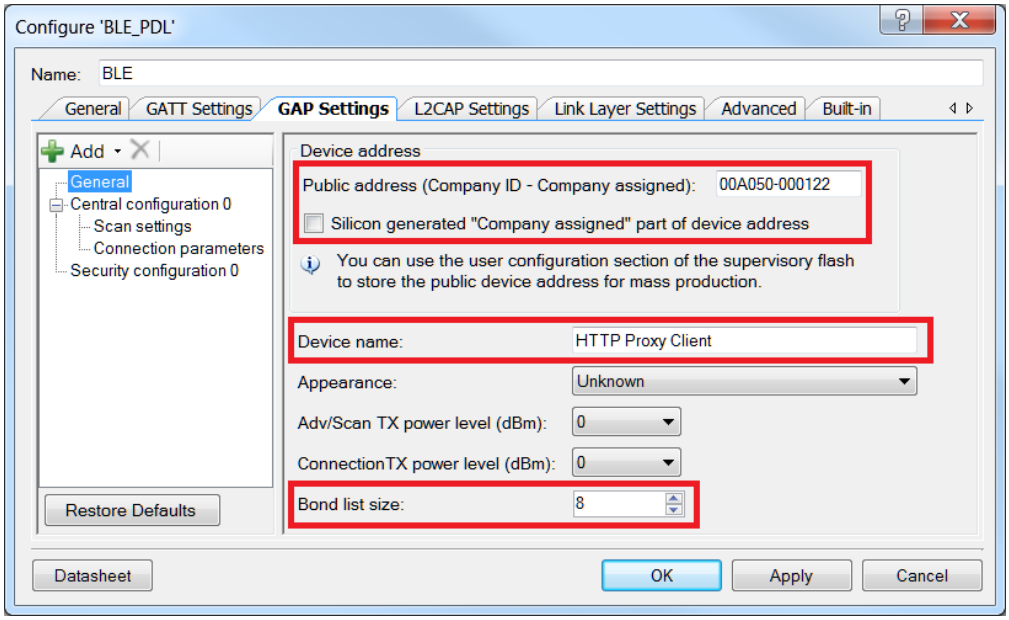


Figure 12. GAP Settings > Scan Settings

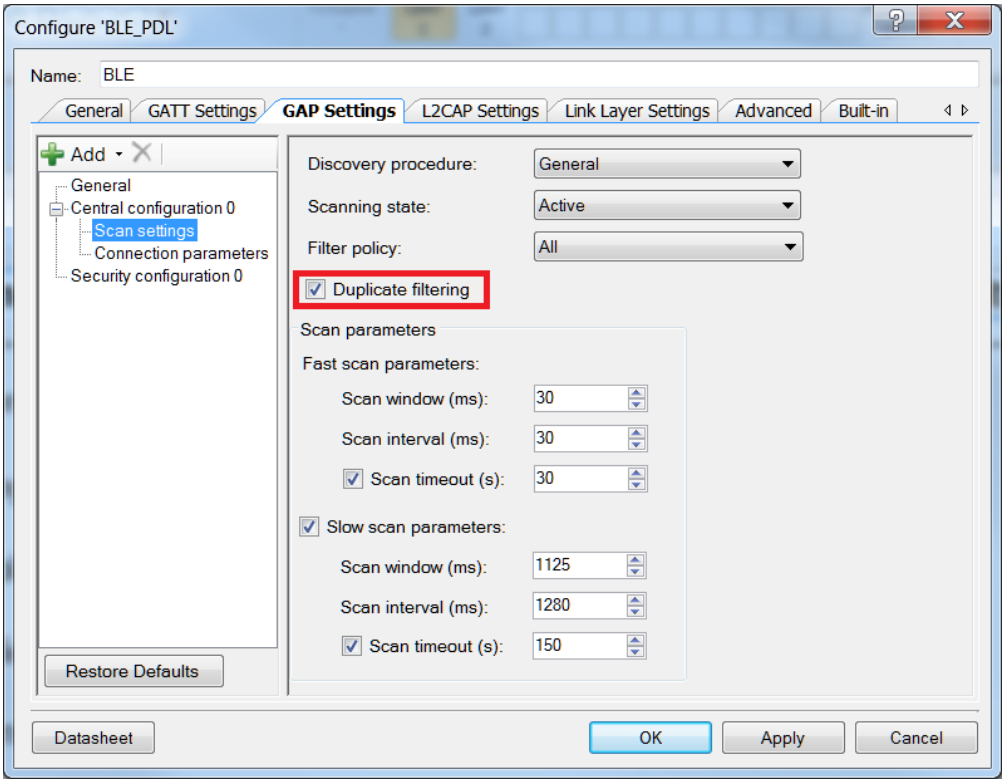
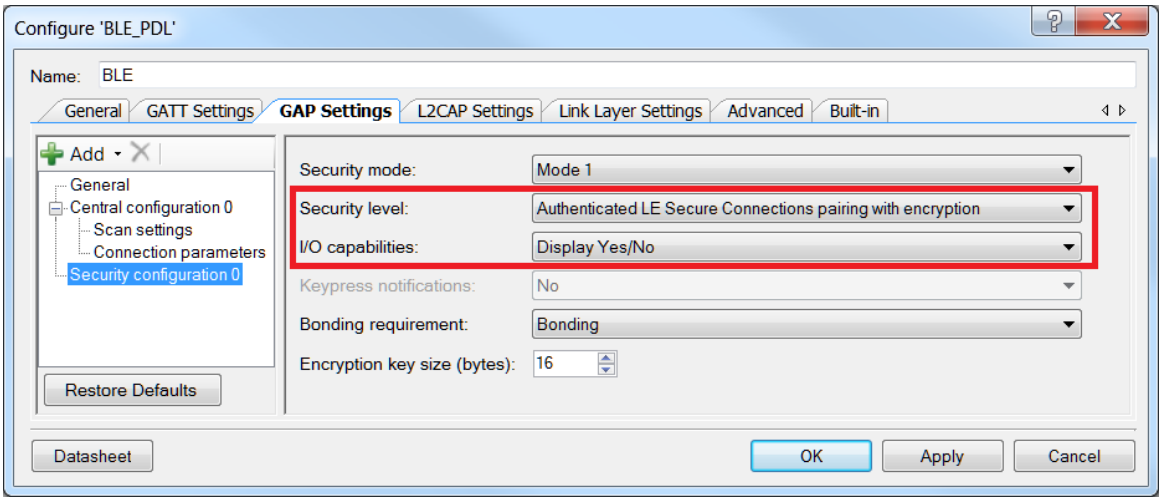


Figure 13. GAP Settings > Security Configuration



Operation

1. Build and program a BLE HTTP Proxy Server and BLE HTTP Proxy Client into CY8CKIT-062 PSoC® 6 BLE Pioneer Kits with PSoC 6 MCU with BLE.
2. Run two HyperTerminal (Bray's Terminal, PuTTY, and so on.) instances: one for the BLE HTTP Proxy Server and another for the BLE HTTP Proxy Client.
3. In the BLE HTTP Proxy Client HyperTerminal window, the client automatically starts scanning for the advertising devices. When the scan report from the device with address **0x00A050000022** is received, press '**c**'. Then, select the number corresponding to the device with address **0x00A050000022**. An approximate example of an output on the client's HyperTerminal may appear as follows:

```
BLE HTTP Proxy Client Example
BLE Stack Version: 5.0.0.718
CY_BLE_EVT_STACK_ON, StartAdvertisement

When you see a scan report from the device you wish to connect to - press 'c'.
CY_BLE_EVT_SET_DEVICE_ADDR_COMPLETE
CY_BLE_EVT_LE_SET_EVENT_MASK_COMPLETE
CY_BLE_EVT_GET_DEVICE_ADDR_COMPLETE: 00a050000122
CY_BLE_EVT_SET_TX_PWR_COMPLETE
CY_BLE_EVT_SET_TX_PWR_COMPLETE
CY_BLE_EVT_GAPC_SCAN_START_STOP
GAPC_START_SCANNING

-------------------------------------------------------------------------------
uuid: HTTP PROXY SERVICE - YES, added to the connect list
ADV type: 0x0 address: 1123ac17c7ef, rssi - -82 dBm, data - 02 01 06 02 08 54 0d 02 0f 18 0a
18 12 18 19 18 23 18 13 18
-------------------------------------------------------------------------------

ADV type: 0x0 address: 00430500dd27, rssi - -84 dBm, data - 02 01 06 02 08 43 05 02 0a 18 0f
18
ADV type: 0x0 address: 1123ac17c7a0, rssi - -83 dBm, data - 02 01 06 0e 09 43 79 63 6c 69 6e
67 20 50 6f 77 65 72 03 03 18 18 03 19 00 00

-------------------------------------------------------------------------------
uuid: HTTP PROXY SERVICE - YES, added to the connect list
ADV type: 0x0 address: 00a050000022, rssi - -77 dBm, data - 02 01 05 12 09 48 54 54 50 20 50
72 6f 78 79 20 53 65 72 76 65 72 03 03 23 18
-------------------------------------------------------------------------------

ADV type: 0x0 address: 001bdc072e9c, rssi - -86 dBm, data - 02 01 06 03 02 10 18 0d 09 50 54
53 2d 42 4c 50 2d 39 43 32 45
ADV type: 0x0 address: 00a05060544e, rssi - -91 dBm, data - 02 01 06 0c 09 43 59 38 43 4b 49
54 2d 31 34 35

c
Detected device:
Device 1: 1123ac17c7ef
Device 2: 00a050000022
select device for connection:  (1..2):
2

CY_BLE_EVT_GAPC_SCAN_START_STOP
Scan complete!

Connecting to the device: 00a050000022
CY_BLE_EVT_GATT_CONNECT_IND: 3, b
CY_BLE_EVT_GAPC_SCAN_START_STOP
Scan complete!

CY_BLE_EVT_GAP_SMP_NEGOTIATED_AUTH_INFO: bdHandle=b, security=3, bonding=1, ekeySize=10,
err=0
Compare this passkey with the one displayed in your peer device and press 'y' or 'n': 801851
y
Accept the displayed passkey
```

```
CY_BLE_EVT_GAP_ENCRYPT_CHANGE: 0
CY_BLE_EVT_GAP_KEYINFO_EXCHNGE_CMPLT
CY_BLE_EVT_GAP_AUTH_COMPLETE: security:3, bonding:1, ekeySize:10, authErr 0
```

4. When the message “Compare this passkey with displayed in your peer device and press '**y**' or '**n**':” is prompted on the HyperTerminal, and if the passkey that follows the message matches on both HyperTerminals, press **SW2** on CY8CKIT-062 PSoC 6 BLE Pioneer Kits or optionally press ‘**y**’ on both HyperTerminals. Wait until the client completes pairing and the peer device discovery.
5. When the message “Press '**g**' to send GET request to get the state of the LED on the peer device” is prompted on the HyperTerminal, press ‘**g**’. Wait until the client handles sending an HTTP GET request and receiving a response from the BLE HTTP Proxy Server. A client’s HyperTerminal’s output may appear as follows:

```
An HTTP 'GET' request was sent
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_URI
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_HEADERS
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_ENTITY_BODY
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_CP
CY_BLE_EVT_HPSC_NOTIFICATION                          Characteristic:
CY_BLE_HPS_HTTP_STATUS_CODE
                                       Status code: 200  "Success"
CY_BLE_EVT_HPSC_READ_CHAR_RESPONSE       Characteristic: CY_BLE_HPS_HTTP_HEADERS
Char data:Server: Apache/2.2.14 (Win32)
Content-Type: text/html; charset=iso-8859-1
Connection: Closed
CY_BLE_EVT_HPSC_READ_CHAR_RESPONSE       Characteristic: CY_BLE_HPS_HTTP_ENTITY_BODY
Char data:<html>
<head>
<title>LED State page</title>
</head>
<body>
<p>LED is off.</p>
</body>
</html>
```

6. When the message “Press '**1**' to send POST request to turn ON the LED on the peer device or press '**0**' to send POST request to turn OFF the LED on the peer device.” is prompted on the HyperTerminal, press ‘**1**’. Wait until the client handles sending an HTTP POST request (to set the state of the blue LED) and receives a response from the BLE HTTP Proxy Server. Observe the blue LED is turned ON on the BLE HTTP Proxy Server device. Press ‘**0**’ and wait until the client handles sending the HTTP POST request (to clear the state of the blue LED) and receives a response from BLE HTTP Proxy Server. Observe the blue LED is turned OFF on the BLE HTTP Proxy Server device. A client’s HyperTerminal’s output for LED on request may appear as follows:

```
An HTTP 'POST' request with parameter 'LED ON' was sent
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_URI
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_HEADERS
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_ENTITY_BODY
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_CP
CY_BLE_EVT_HPSC_NOTIFICATION                         Characteristic:
CY_BLE_HPS_HTTP_STATUS_CODE
                                       Status code: 200  "Success"
CY_BLE_EVT_HPSC_READ_CHAR_RESPONSE       Characteristic: CY_BLE_HPS_HTTP_HEADERS
Char data:Server: Apache/2.2.14 (Win32)
Content-Type: text/html; charset=iso-8859-1
Connection: Closed
CY_BLE_EVT_HPSC_READ_CHAR_RESPONSE       Characteristic: CY_BLE_HPS_HTTP_ENTITY_BODY
Char data:<html>
<head>
<title>LED State page</title>
</head>
<body>
<p>LED is on.</p>
</body>
</html>
```

A Client's HyperTerminal's output for the LED-OFF request may appear as follows:

```
An HTTP 'POST' request with parameter 'LED OFF' was sent
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_URI
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_HEADERS
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_ENTITY_BODY
CY_BLE_EVT_HPSC_WRITE_CHAR_RESPONSE      Characteristic: CY_BLE_HPS_HTTP_CP
CY_BLE_EVT_HPSC_NOTIFICATION                         Characteristic:
CY_BLE_HPS_HTTP_STATUS_CODE
                                        Status code: 200  "Success"
CY_BLE_EVT_HPSC_READ_CHAR_RESPONSE       Characteristic: CY_BLE_HPS_HTTP_HEADERS
Char data:Server: Apache/2.2.14 (Win32)
Content-Type: text/html; charset=iso-8859-1
Connection: Closed
CY_BLE_EVT_HPSC_READ_CHAR_RESPONSE       Characteristic: CY_BLE_HPS_HTTP_ENTITY_BODY
Char data:<html>
<head>
<title>LED State page</title>
</head>
<body>
<p>LED is off.</p>
</body>
</html>
```

7. Press '**d**' on any of HyperTerminal to disconnect the devices.

Note: When the bonding info is removed from the BLE HTTP Proxy Server (that may happen when the devices were previously bonded and the Server device was reprogrammed and Client was not), the attempt to pair the devices will fail for the INSUFFICIENT_ENCRYPTION_KEY_SIZE reason. To handle that, reprogram the client device or remove the bonding information from it. To do that, disconnect both devices and press '**r**' on BLE HTTP Proxy Client HyperTerminal.

Related Documents

Application Notes		
AN210781	Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	Describes the PSoC 6 MCU with BLE Connectivity, and how to build a basic code example.
AN215656	PSoC 6 MCU Dual-Core CPU System Design	Presents the theory and design considerations related to this code example.
Software and Drivers		
CySmart – BLE Test and Debug Tool		CySmart is a BLE host emulation tool for Windows PCs. The tool provides an easy-to-use GUI to enable the user to test and debug their BLE Peripheral applications.
PSoC Creator Component Datasheets		
Bluetooth Low Energy (BLE_PDL) Component		The Bluetooth Low Energy (BLE_PDL) Component provides a comprehensive GUI-based configuration window to facilitate designing applications requiring BLE connectivity.
Device Documentation		
PSoC 6 MCU: PSoC 63 with BLE Datasheet Programmable System-on-Chip		PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual (TRM)
Development Kit (DVK) Documentation		
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit		

Document History

Document Title: CE215124 - BLE HTTP Proxy with PSoC 6 MCU with BLE Connectivity

Document Number: 002-15124

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	5968152	NPAL	11/21/2017	New Code Example

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress 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 Cypress 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 Cypress 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 Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress 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, Cypress reserves the right to make changes to this document without further notice. Cypress 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. Cypress 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. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress 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 Cypress products.

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