

Chapter 10: Debugging

After completing this chapter, you will have gained a basic understanding of Host Controller Interface (HCI) commands and WICED HCI format. You will then learn how to enable WICED HCI trace messages from the Bluetooth® stack and view them using the BTSpy program.

Table of contents

10.1	Arm debug port	2
10.2	Host Controller Interface (HCI)	2
10.3	WICED HCI	3
10.3.1	Message Format	3
10.4	WICED HCI debug firmware architecture	4
10.4.1	Enable logging.....	4
10.4.2	Include WICED HCI header file	4
10.4.3	Setup debug UART	4
10.5	BTSpy Application	5
10.6	Exercises	8
Exercise 1: Add WICED HCI traces to pairing application and use BTSpy to test		8

Document conventions

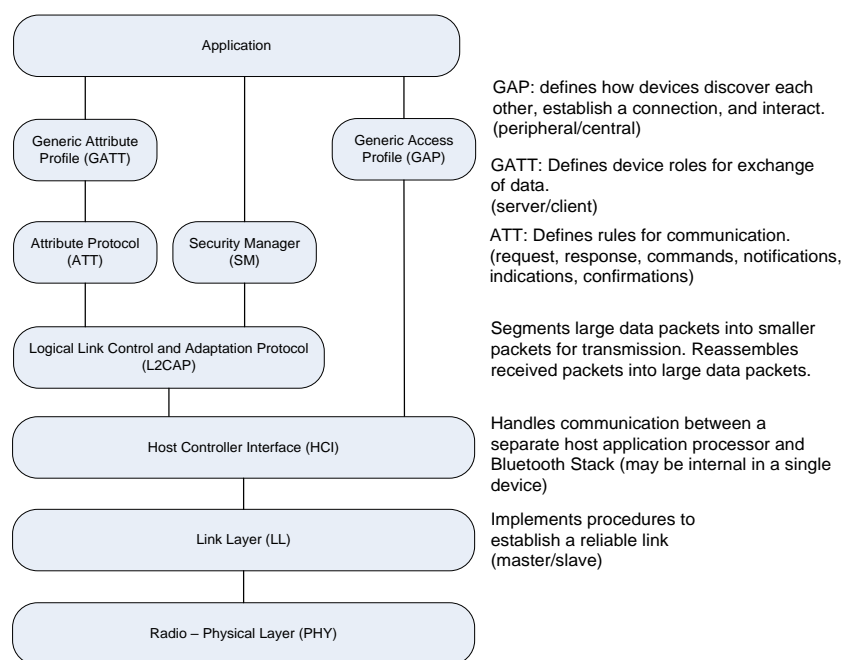
Convention	Usage	Example
Courier New	Displays code and text commands	CY_ISR_PROTO(MyISR) ; make build
<i>Italics</i>	Displays file names and paths	<i>sourcefile.hex</i>
[bracketed, bold]	Displays keyboard commands in procedures	[Enter] or [Ctrl] [C]
Menu > Selection	Represents menu paths	File > New Project > Clone
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the Debugger icon, and then click Next .

10.1 Arm debug port

Hardware debugging on Bluetooth® applications can be done using an OpenOCD supported probe just like on any other PSoC™ 6 application. However, if your device has an active Bluetooth® LE connection it will drop the connection when the CPU is halted in the debugger (after the connection timeout). Therefore, for Bluetooth® LE applications it is more common to use WICED HCI and an application like BTSPy for debugging rather than a hardware debugger. Unlike traditional hardware debugging, WICED HCI provides debug messages while the Bluetooth® connection stays active. That's what the rest of this chapter covers.

10.2 Host Controller Interface (HCI)

In many complicated systems, hierarchy is used to manage the complexity and Bluetooth® is one of them. The Bluetooth® stack is called a stack because it is a set of blocks that have well defined interfaces. Here is a simple picture of the software system that we have been using. You have been writing code in the block called "Application." You have made API calls and gotten events from the "Attribute Protocol" and you implemented the "Generic Attribute Profile" by building the GATT Database. Moreover, you advertised using GAP and you Paired and Bonded by using the Security Manager.

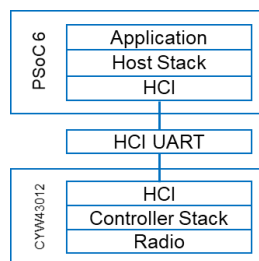


The next block to talk about is the "Host Controller Interface." In many Bluetooth® systems, the radio is a separate chip from the one that runs the upper levels of the stack and the application. The radio chip takes the name of controller because it contains the radio and radio controller while the chip running the upper level of the stack and the application is called the host because it hosts the application. The interface between the host and controller is therefore called the Host Controller Interface, or HCI for short.

By standardizing the HCI, it allows big application processors (like those existing in PCs and cellphones) to interface with Bluetooth®.

Sometimes host and controller are integrated into one chip but HCI persists even though both sides may be physically on the same chip. In that case, the HCI layer is essentially just a pass-through. Other times, the

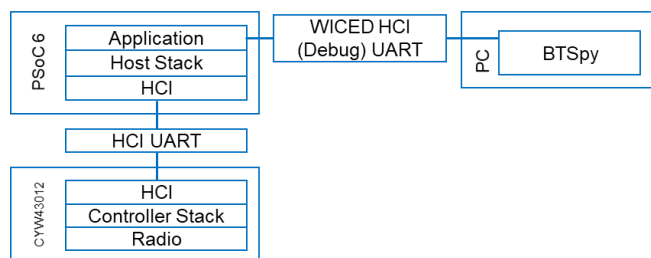
Bluetooth® stack can be split into separate host and controller parts. For example, the PSoC™ 6 and CYW43012 combo radio that we have been using is a 2-chip solution that looks like this:



In this case, a dedicated UART interface is used to send HCI messages back and forth between the PSoC™ 6 and CYW43012.

10.3 WICED HCI

The HCI concept was extended by the Infineon software team to provide a means of mirroring the HCI traffic between the devices to allow debugging of the Bluetooth® interface while it is running. This interface is called "WICED HCI." Normally you send WICED HCI messages to the debug UART interface that connects between the PSoC™ 6 and a PC. The result is that from a PC you can view your standard debug print messages along with all HCI messages between the host and controller devices.



10.3.1 Message Format

WICED HCI is a packet-based format for PC applications to interact with the application running on the host. The packets have 3 standard fields plus an optional number of additional bytes for payload. The packet looks like this:

- 0x19 – the initial byte to indicate a WICED HCI packet
- 2-byte little endian Opcode consisting of a 1-byte Control Group (a.k.a. Group Code) and a 1-byte Sub-Command (a.k.a. Command Code)
- A 2-byte little endian length of the additional bytes
- An optional number of additional bytes for payload

The Control Group is one of a predefined list of categories of transactions including `Device=0x00`, `BLE=0x01`, `GATT=0x02`, etc. These groups are defined in `mtb_shared/btstack/<version>/wiced_include/hci_control_api.h`. Each Control Group has one or more optional Sub-Commands. For instance, the Device Control Group has Sub-Commands `Reset=0x01`, `Trace Enable=0x02`, etc.

The Control Group plus the Sub-Command together is called a "Command" or an "Opcode" and is a 16-bit number. For example, Device Reset = 0x0001. In the actual data packet, the Opcode is represented little endian. For example, the packet for a Device Reset = 19 01 00 00 00.

10.4 WICED HCI debug firmware architecture

In order to enable WICED HCI debug traces, there are just three things that need to be done in the application: (1) enable the WICED HCI logging feature in the stack; (2) include the appropriate header file; and (3) setup the debug UART to send messages in the WICED HCI format.

10.4.1 Enable logging

The first step is to enable BTSpY logging from the stack. This is done by setting a define in the application's *Makefile*:

```
DEFINES=ENABLE_BT_SPY_LOG
```

10.4.2 Include WICED HCI header file

The following file must be included to get access to the WICED HCI functions:

```
#include "cybt_debug_uart.h"
```

10.4.3 Setup debug UART

When using WICED HCI, the debug UART needs to be initialized using the function `cybt_debug_uart_init`. This can be done in initialization immediately after calling `cybt_platform_config_init`. The first argument is a pointer to a configuration structure of type `cybt_debug_uart_config_t` where you specify the pins, baud rate, and flow control setting. The second argument is an optional callback function in case you want to do your own processing when a debug message is received.

You can add the new code conditionally so that the debug UART uses the standard Retarget-IO functionality if BTSpY logging is not enabled in the *Makefile*.

```
#ifdef ENABLE_BT_SPY_LOG
    cybt_debug_uart_config_t config =
    {
        .uart_tx_pin = CYBSP_DEBUG_UART_TX,
        .uart_rx_pin = CYBSP_DEBUG_UART_RX,
        .uart_rts_pin = CYBSP_DEBUG_UART_RTS,
        .uart_cts_pin = CYBSP_DEBUG_UART_CTS,
        .baud_rate = 3000000,
        .flow_control = true,
    };
    cybt_debug_uart_init(&config, NULL);
#else
    cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
        CY_RETARGET_IO_BAUDRATE);
#endif
```

Note: The BTSpy application supports baud rates of 115200 or 3000000 but the higher value is recommended.

Optionally, you can call the function `cybt_platform_set_trace_level` to change which blocks and which severity of messages are printed (e.g. info, warning, error). The available trace IDs and levels can be found in `mtb_shared/bluetooth-freertos/<version>/platform/include/cybt_platform_trace.h`. The default values are `CYBT_TRACE_ID_STACK` and `CYBT_TRACE_LEVEL_DEBUG`.

If you are going to call `cybt_platform_set_trace_level`, you must include an additional header file:

```
#include "cybt_platform_trace.h"
```

10.5 BTSpy Application

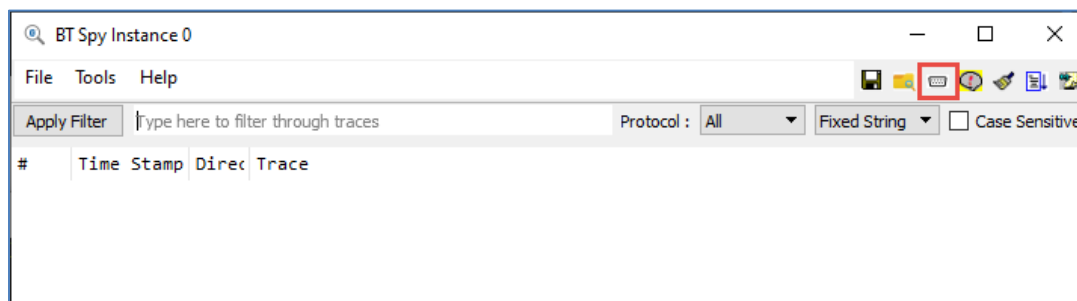
The BTSpy application is available in a Bluetooth® utilities library on GitHub. To get the library, open a terminal (e.g. `modus-shell` on Windows) and execute this command from the directory where you want the library to be downloaded:

```
git clone https://github.com/Infineon/btsdk-utils.git
```

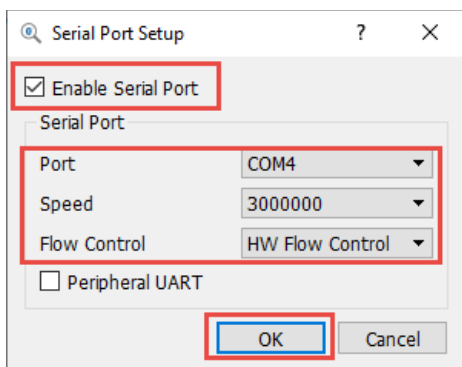
Once it is cloned, go to the directory `btstk-utils/BTSpy`. You will see subdirectories for Linux64, OSX, and Windows versions of the tool. The screen shots below are from the Windows version.

In the *Windows* directory, the executable is `BTSPy.exe`. It can be run directly from the terminal (`./BTSPy.exe`) or by double-clicking on it from Windows explorer.

Once it starts up, you will see a window like this:







The first thing to do is to setup the UART connection. Click the **Serial Port Setup** button (see the red box in the figure above) or use the menu item **Tools > Serial Port Setup**. Check the box to **Enable Serial Port**, select the correct port for your kit, set the baud rate, and enable hardware flow control. Click **OK** when you are done.



Note: To disconnect from the kit, you must open the serial port setup window again, uncheck the Enable Serial Port button, and click OK.

As messages are received from the kit, you will see green messages for incoming Bluetooth® traffic, yellow for outgoing Bluetooth® traffic, and white for non-HCI messages.

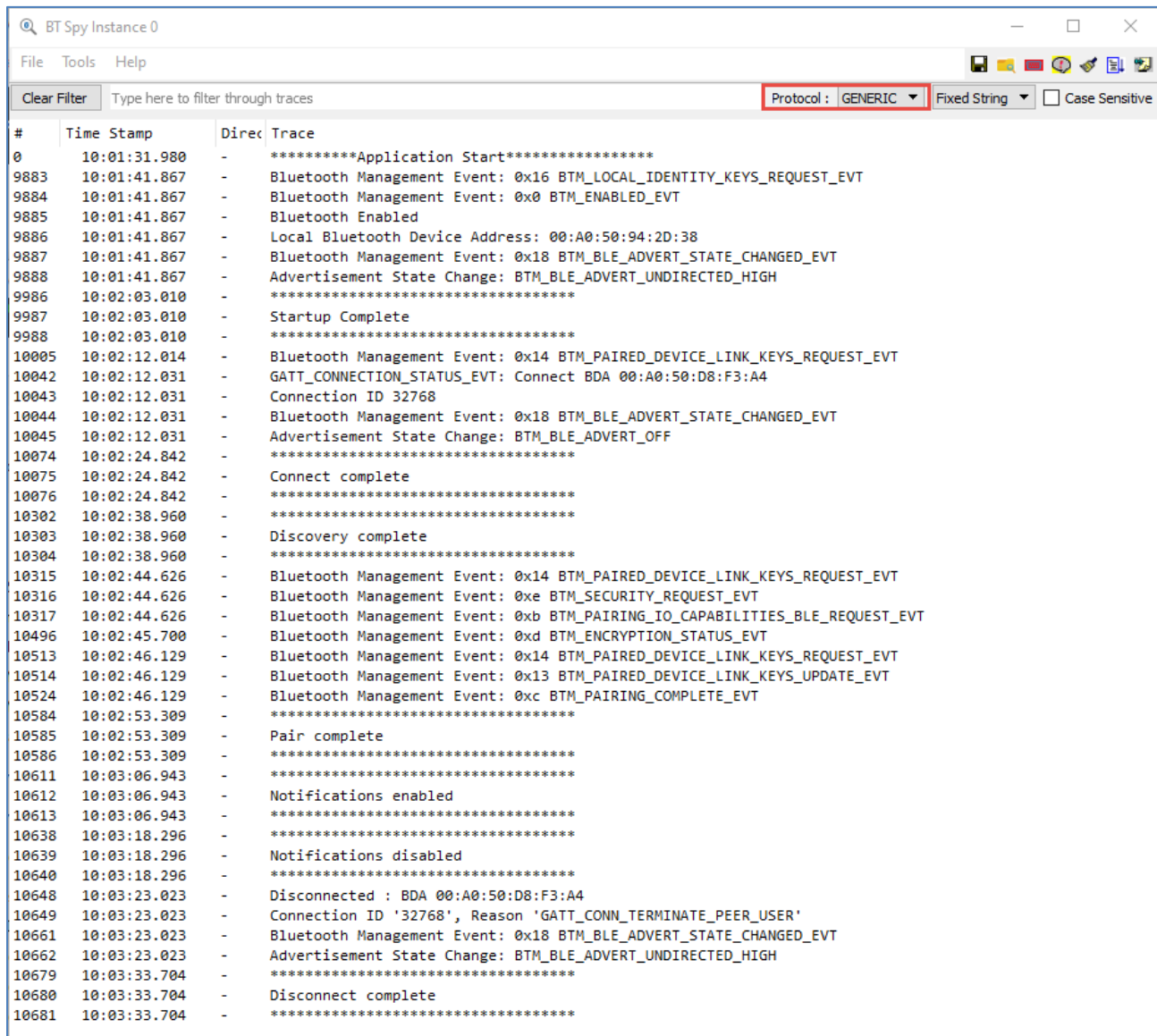
A few other controls of interest are:

Command	Icon	Use
Clear Trace		Clear the log history
Note		Allows you to enter a note that shows up in the log
File Logging Options		Setup and start/stop logging messages to a file
Open Logged File		Open a previous log file in the tool
Protocol	Protocol : <input type="text" value="All"/>	Allows you to filter out messages of certain types (ALL, HCI, GENERIC, etc.)

The following figure shows an example of a log that was created for a Bluetooth® peripheral that allows pairing and has notifications. The lines shown with stars above/below were added using the Note function (the stars are added automatically by the tool). As you can see, after pairing is complete, the user enables notifications from the central. To do this, the central sends a GATT write request (1) to handle 0x000C (which is the handle for the CCCD) and a value of 01 00 (which is the code to enable notifications). The peripheral responds to the write request (2). Next, the central sends a read request (3) to the same handle and the peripheral sends back a read response (4) with the new value of the CCCD of 01 00. This allows you to see exactly what is being sent back and forth across the Bluetooth® link which is extremely helpful during debugging.

10583	10:02:46.129	D+	Status : Success (0x00)
10584	10:02:53.309	-	*****
10585	10:02:53.309	-	Pair complete
10586	10:02:53.309	-	*****
10587	10:02:57.845	D+	RCVD - HCI_RAW ACL Data :
10588	10:02:57.845	D+	0000: 05 00 04 00 12 0c 00 01 00
10589	10:02:57.845	D+	RCVD [0] ACL Data from HCI. Handle: 0x040 Boundary: 2 Brdcst: 0 Len: 9 Data: 0x05 0x00 0x04 ...
10590	10:02:57.845	D+	ATT RECV Command. len:5 Name: Write Request (0x12)
10591	10:02:57.845	D+	Attr Handle : 12 (0x000c)
10592	10:02:57.845	D+	Attr Value : 01 00
10593	10:02:57.845	D+	SENT - HCI_RAW ACL Data :
10594	10:02:57.845	D+	0000: 01 00 04 00 13
10595	10:02:57.845	D+	SENT [0] ACL Data to HCI. Handle: 0x040 Boundary: 0 Brdcst: 0 Len: 5 Data: 0x01 0x00 0x04 ...
10596	10:02:57.845	D+	ATT SEND Command. len:1 Name: Write Response (0x13)
10597	10:02:57.889	D+	RCVD - HCI_RAW ACL Data :
10598	10:02:57.889	D+	0000: 03 00 04 00 0a 0c 00
10599	10:02:57.889	D+	RCVD [0] ACL Data from HCI. Handle: 0x040 Boundary: 2 Brdcst: 0 Len: 7 Data: 0x03 0x00 0x04 ...
10600	10:02:57.890	D+	ATT RECV Command. len:3 Name: Read Request (0x0a)
10601	10:02:57.890	D+	Attr Handle : 12 (0x000c)
10602	10:02:57.890	D+	SENT - HCI_RAW ACL Data :
10603	10:02:57.890	D+	0000: 03 00 04 00 0b 01 00
10604	10:02:57.890	D+	SENT [0] ACL Data to HCI. Handle: 0x040 Boundary: 0 Brdcst: 0 Len: 7 Data: 0x03 0x00 0x04 ...
10605	10:02:57.890	D+	ATT SEND Command. len:3 Name: Read Response (0x0b)
10606	10:02:57.890	D+	Attr Value : 01 00
10607	10:02:57.897	D+	HCI_RAW Event Data:
10608	10:02:57.897	D+	0000: 01 40 00 02 00
10609	10:02:57.897	D+	RCVD [0] Event from HCI. Name: HCI_Number_Of_Completed_Packets (Hex Code: 0x13 Param Len: 5)
10610	10:02:57.897	D+	0 : 64 (0x0040) - 2
10611	10:03:06.943	-	*****
10612	10:03:06.943	-	Notifications enabled
10613	10:03:06.943	-	*****
10614	10:03:10.707	D+	RCVD - HCI_RAW ACL Data :

If you change the protocol filter from ALL to GENERIC, you will see just the messages that are printed from the application along with the notes that were added manually.



#	Time Stamp	Dir	Trace
0	10:01:31.980	-	*****Application Start*****
9883	10:01:41.867	-	Bluetooth Management Event: 0x16 BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT
9884	10:01:41.867	-	Bluetooth Management Event: 0x0 BTM_ENABLED_EVT
9885	10:01:41.867	-	Bluetooth Enabled
9886	10:01:41.867	-	Local Bluetooth Device Address: 00:A0:50:94:2D:38
9887	10:01:41.867	-	Bluetooth Management Event: 0x18 BTM_BLE_ADVERT_STATE_CHANGED_EVT
9888	10:01:41.867	-	Advertisement State Change: BTM_BLE_ADVERT_UNDIRECTED_HIGH
9986	10:02:03.010	-	*****
9987	10:02:03.010	-	Startup Complete
9988	10:02:03.010	-	*****
10005	10:02:12.014	-	Bluetooth Management Event: 0x14 BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT
10042	10:02:12.031	-	GATT_CONNECTION_STATUS_EVT: Connect BDA 00:A0:50:D8:F3:A4
10043	10:02:12.031	-	Connection ID 32768
10044	10:02:12.031	-	Bluetooth Management Event: 0x18 BTM_BLE_ADVERT_STATE_CHANGED_EVT
10045	10:02:12.031	-	Advertisement State Change: BTM_BLE_ADVERT_OFF
10074	10:02:24.842	-	*****
10075	10:02:24.842	-	Connect complete
10076	10:02:24.842	-	*****
10302	10:02:38.960	-	*****
10303	10:02:38.960	-	Discovery complete
10304	10:02:38.960	-	*****
10315	10:02:44.626	-	Bluetooth Management Event: 0x14 BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT
10316	10:02:44.626	-	Bluetooth Management Event: 0xe BTM_SECURITY_REQUEST_EVT
10317	10:02:44.626	-	Bluetooth Management Event: 0xb BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT
10496	10:02:45.700	-	Bluetooth Management Event: 0xd BTM_ENCRYPTION_STATUS_EVT
10513	10:02:46.129	-	Bluetooth Management Event: 0x14 BTM_PAIRING_DEVICE_LINK_KEYS_REQUEST_EVT
10514	10:02:46.129	-	Bluetooth Management Event: 0x13 BTM_PAIRING_DEVICE_LINK_KEYS_UPDATE_EVT
10524	10:02:46.129	-	Bluetooth Management Event: 0xc BTM_PAIRING_COMPLETE_EVT
10584	10:02:53.309	-	*****
10585	10:02:53.309	-	Pair complete
10586	10:02:53.309	-	*****
10611	10:03:06.943	-	*****
10612	10:03:06.943	-	Notifications enabled
10613	10:03:06.943	-	*****
10638	10:03:18.296	-	*****
10639	10:03:18.296	-	Notifications disabled
10640	10:03:18.296	-	*****
10648	10:03:23.023	-	Disconnected : BDA 00:A0:50:D8:F3:A4
10649	10:03:23.023	-	Connection ID '32768', Reason 'GATT_CONN_TERMINATE_PEER_USER'
10661	10:03:23.023	-	Bluetooth Management Event: 0x18 BTM_BLE_ADVERT_STATE_CHANGED_EVT
10662	10:03:23.023	-	Advertisement State Change: BTM_BLE_ADVERT_UNDIRECTED_HIGH
10679	10:03:33.704	-	*****
10680	10:03:33.704	-	Disconnect complete
10681	10:03:33.704	-	*****

10.6 Exercises

Exercise 1: Add WICED HCI traces to pairing application and use BTSpy to test

In this exercise, you will start from the Bluetooth® pairing exercise from a previous chapter and you will add in WICED HCI trace capability. You will then use BTSpy to look at the application messages and HCI messages.



1. Create a new application for the BSP you are using.

On the application template page, use the **Browse** button to start from the completed application for the *ch05_ex01_pair* exercise. If you did not complete that exercise, the solution can be found in *Projects/key_ch05_ex01_pair*.

Name the new application *ch10_ex01_btspy*.



2. Open the Bluetooth® configurator and on the GAP settings tab change the device name to `<init>_btspy`.



3. Follow the instructions in section 10.4 to enable WICED HCI logging.



4. Program your application to the kit.



5. If you have a UART terminal open, close it.



6. If you don't already have it, download the *btsdk-utils* repo from GitHub as described in section 10.5.



7. Run the BTSpy application. Configure the serial port for your kit and enable it.



8. Clear the log and then reset the kit to observe the messages.



9. When startup has completed (advertising has started) enter a note that says "Startup Complete."



10. Open AIROC™ Connect and connect to your device.



11. Once the connection has been made, enter a note in BTSpy that says "Connect complete."



12. Click the GATT DB widget followed by the Unknown Service and then the Characteristic that has Read & Notify properties. This is the Counter Characteristic that counts button presses.



13. Accept the pairing request on the mobile device. Once it completes, enter a note in BTSpy that says "Pairing Complete."



14. Click the **Notify** button in AIROC™ Connect. Once it completes, enter a note in BTSpy that says "Notifications Enabled."



15. Scroll through the BTSpy log messages to see what is going on during each operation. You will see the notes that you added to be able to tell when each operation starts/stops.



16. Use the **Protocol** dropdown to select **GENERIC**. Now you will see just the informational messages from your application along with the notes that you added.



17. Disconnect and close AIROC™ Connect. Close BTSpy.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2023 Infineon Technologies AG.
All Rights Reserved.

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.