Chapter 9: Exercises

Objective: This chapter is a compilation of all exercises by chapter with additional exercises at the end.

TABLE OF CONTENTS

9.1 EXERCISES FOR TOUR – MODUS

9.1-1 CREATE A FORUM ACCOUNT AND DOWNLOAD THE LATEST BT SDK

1. Go to https://community.cypress.com/welcome
2. Click “Log in” from the top right corner of the page and login to your Cypress account. If you do not have an account, you will need to create one first.
3. Once you are logged in, click the “Wireless” icon and then explore.
4. Go to “Wireless”, “ModusToolbox Bluetooth” and download the latest ModusToolbox BT SDK from the Downloads table.
5. Unzip the BT SDK file once it has been downloaded.

9.1-2 START MODUSTOOLBOX IDE, EXPLORE THE DOCUMENTATION, AND UPDATE TO THE LATEST BT SDK

1. Run ModusToolbox IDE and create a new workspace.
2. Select the Documents tab in the lower-left panel.
3. Explore the different documents available such as the *ModusToolbox IDE Help*, *Quick Start Guide, User Guide*, *Eclipse IDE Survival Guide* and *WICED API Reference*
4. Go to “Help > Update ModusToolbox SDKs…”
5. Click “OK” if you see a message that says “Unable to find installable SDKs,…”.

6. It the latest version of the SDK is not already installed, Click “Install Custom SDK” and browse to the location of the SDK file to install the latest version.
7. Optional: You can uninstall any older version(s) of the BT SDK if desired.

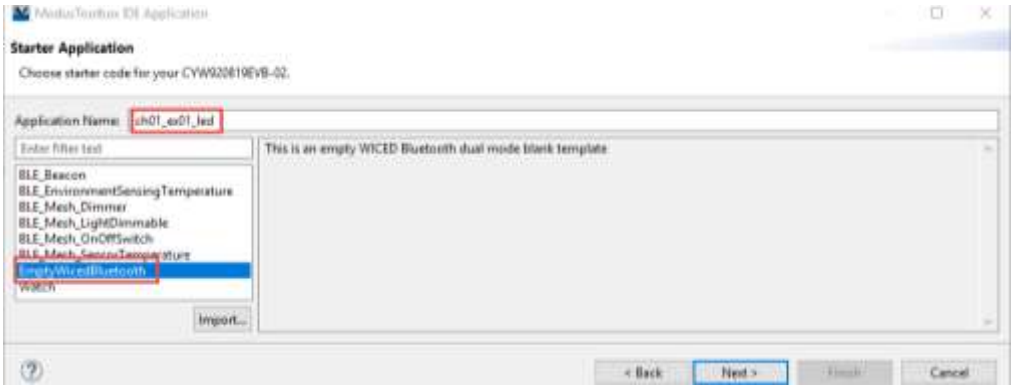
9.1-2A QUESTIONS

1. Where is the WICED API documentation for the PWM located?

9.1-3 PROGRAM A SIMPLE APPLICATION

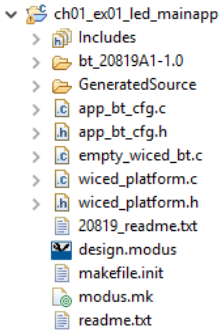
In this exercise, you will create a new application in ModusToolbox IDE and will program it to a kit.

1. In the Quick Panel tab click "New Application".
2. Select the CYW920819EVB-02 kit and click "Next >".
3. Select "EmptyWicedBluetooth".
4. Change the application name to **ch09_ex01_led**.



Note: you can call the application anything you like but it will really help if you maintain an alphabetically sortable naming scheme – you are going to create quite a few projects.

5. Click "Next ". Verify the presented device, board, and example, then click "Finish". When the application has been created, the Project Explorer window in Eclipse should look like the following screenshot (ModusToolbox IDE adds "_mainapp" to the project name because some applications contain more than one project):



6. Open the top-level C file which in this case is called empty_wiced_bt.c.
7. Add the following line in the application_start function:

```
wiced_hal_gpio_set_pin_output(WICED_GPIO_PIN_LED_1,
GPIO_PIN_OUTPUT_LOW);
```

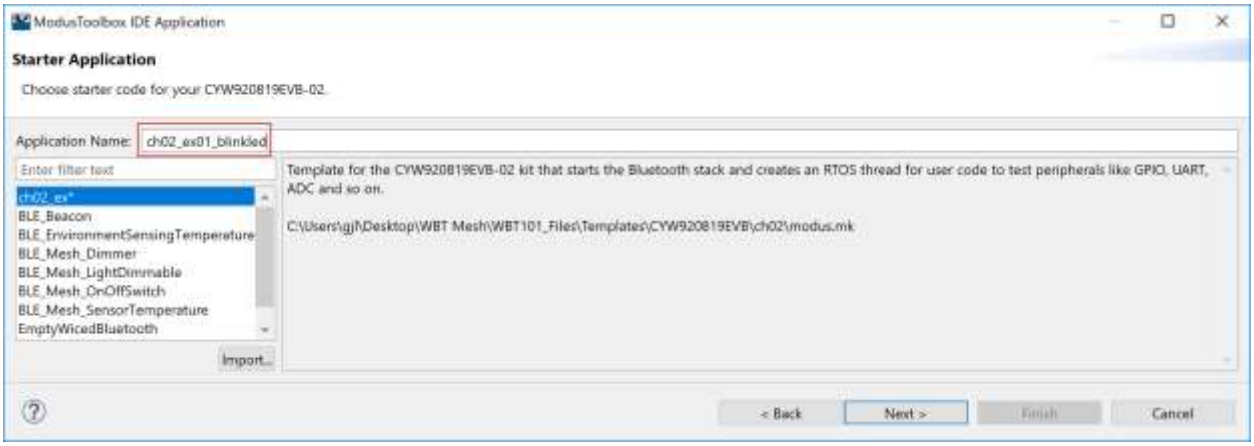
8. Connect your CYW920819EVB-02 kit to a USB port on your computer.
9. In the Quick Panel, look in the "Launches" section and press the "ch01_ex01_led Build + Program" link.
10. Once the build and program operations are done, you should see "Download succeeded" in the Console window and LED1 (yellow user LED) should be on.

9.2 EXERCISES FOR PERIPHERALS

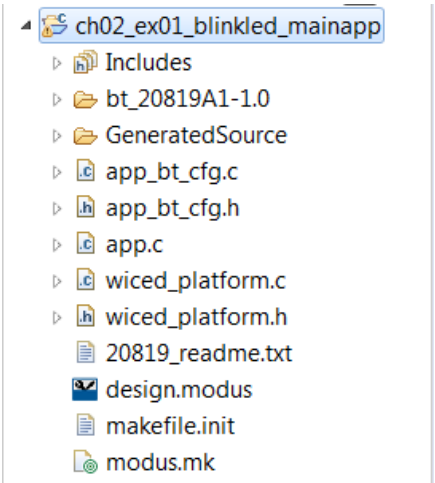
9.2-1 (GPIO) BLINK AN LED

In this exercise, you will create an application to blink LED_2 on the kit at 2 Hz. This material is covered in **Error! Reference source not found.**

1. In the Quick Panel click "New Application".
2. Select the CYW920819EVB-02 kit and press "Next >".
3. Press the "Import…" button and navigate to the course materials "templates/CYW920819EVB/ch02" folder.
4. Double-click on the modus.mk file to import the template and return to the wizard.
5. The new template will now be selected, but you may need to wait a few seconds for the dialog to refresh.



6. Change the application name to **ch03_ex01_blinkled**.
   Note: you can call the application anything you like but it will really help if you maintain an alphabetically sortable naming scheme – you are going to create quite a few projects in this course.
7. Click "Next ". Verify the presented device, board, and example, then click "Finish". When you finish, the Project Explorer window in Eclipse should look like the following screenshot:



8. Examine app.c to make sure you understand what it does.
   All WICED BLE applications are multi-threaded (the BLE stack requires it). There is an operating system (RTOS) that gets launched from the device startup code and you can use it to create your own threads. Each thread has a function that runs almost as though it is the only software in the

system – the RTOS allocates time for all threads to execute when they need to. This makes it easier to write your programs without a lot of extra code in your main loop. The details of how to use the RTOS effectively are covered in the next chapter but, in these exercises, we will show you how to create a thread and associate it with a function for the code you will write (look in app_bt_management_callback()).

9. Add code in the app_task thread function to do the following:
    a. Read the state of WICED_GPIO_PIN_LED_1
        i. Hint: Go back to the section on GPIOs if you need a reminder on using pins.
    b. Drive the state of WICED_GPIO_PIN_LED_1 to the opposite value.
10. In the Quick Panel, look in the "Launches" section and press the "ch02_ex01_blinkled Build + Program" link.

9.2-1A QUESTIONS

1. What is the name of the first user application function that is executed? What does it do?

2. What is the purpose of the function app_bt_management_callback? When does the BTM_ENABLED_EVT case occur?

3. What controls the rate of the LED blinking?

9.2-2 (GPIO) ADD DEBUG PRINTING TO THE LED BLINK PROJECT

For this exercise, you will add a message that is printed to a UART terminal each time the LED changes state. This material is covered in **Error! Reference source not found.**

1. Repeat steps 2 through 8 from the previous exercise to create a new application called **ch03_ex02_blinkled_print**.
    Note: It is possible to copy-paste the prior project and rename it. The new application will build without error. However, Eclipse does not re-create programming or debugging configurations in a copied project and so you would need to manually configure a GDB connection to an OpenOCD server. Accordingly, we do not recommend copying whole projects unless you are an Eclipse and OpenOCD expert who can set up debugging configurations faster than it takes to create a new project!
2. Copy the app.c file from the previous application.
    Note: The safe and quick way to do this is to highlight the file you want in the Eclipse Project Explorer, copy (Control-C), select the newly created application, and paste (Control-V). Eclipse will ask if you wish to overwrite the file (if it does not, you are probably copying to the wrong location). You can also use drag & drop but be sure to press and hold the control key so that you copy, not move, the file.
3. Add WICED_BT_TRACE calls to display “LED LOW” and “LED HIGH” at the appropriate times.
    a. Hint: Go back to the section on Debug Printing if you need a refresher.
    b. Hint: Remember to set the debug UART to WICED_ROUTE_DEBUG_TO_PUART. Although you will see comments in the template code encouraging you to put initialization code in the app_bt_management_callback() function so that it runs when the BLE stack starts up, we recommend doing it in application_start() instead. This is

because the PUART is a special type of peripheral and you may want to print messages before even trying to start the stack!

c. Hint: Remember to use \n\r to create a new line so that information is printed on a new line each time the LED changes.

4. Program your application to the board.
5. Open a terminal window (e.g. PuTTY or TeraTerm) with a baud rate of 115200 and observe the messages being printed.
    a. Hint: The PUART will be the larger number of the two WICED COM ports.
    b. Hint: if you don't have terminal emulator software installed, you can use putty.exe which is included in the class files under "Software_tools". To configure putty:
        i. Go to the Serial tab, select the correct COM port (you can get this from the device manager under "Ports (COM & LPT)" as *"WICED USB Serial Port"*), and set the speed to 115200.
        ii. Go to the session tab, select the Serial button, and click on "Open".
        iii. If you want an automatic carriage return with a line feed in putty (i.e. add a \r for every \n) check the box next to "Terminal -> Implicit CR in every LF"

9.2-3 (GPIO) READ THE STATE OF A MECHANICAL BUTTON

In this exercise, you will control an LED by monitoring the state of a mechanical button on the kit. This material is covered in **Error! Reference source not found.**

1. Create another new application called **ch03_ex03_button** with the same template.
2. Again, copy your code that toggles the LED from the **ch03_ex01_blink** application
3. In the C file:
    a. Change the thread sleep time to 100ms.
    b. In the thread function, check the state of mechanical button input (use WICED_GPIO_PIN_BUTTON_1). Turn on WICED_GPIO_PIN_LED_1 if the button is pressed and turn it off if the button is not pressed.
4. Program your project to the board and test it.

9.2-4 (GPIO) USE AN INTERRUPT TO TOGGLE THE STATE OF AN LED

In this exercise, rather than polling the state of the button in a thread, you will use an interrupt so that your firmware is notified every time the button is pressed. In the interrupt callback function, you will toggle the state of the LED. This material is covered in **Error! Reference source not found.**

1. Create another new application called **ch03_ex04_interrupt** with the same template. There is no need to copy source code from another application in this exercise.
2. In the app.c file:
    a. Remove the calls to wiced_rtos_create_thread() and wiced_rtos_init_thread().
    b. Delete the thread function.
2. In the BTM_ENABLED_EVT, set up a falling edge interrupt for the GPIO connected to the button and register the callback function.
    a. Hint: You will need to call wiced_hal_gpio_register_pin_for_interrupt and wiced_hal_gpio_configure_pin.
3. Create the interrupt callback function so that it toggles the state of the LED each time the button is pressed.
4. Program your application to the board and test it.

9.2-5 (TIMER) USE A TIMER TO TOGGLE AN LED

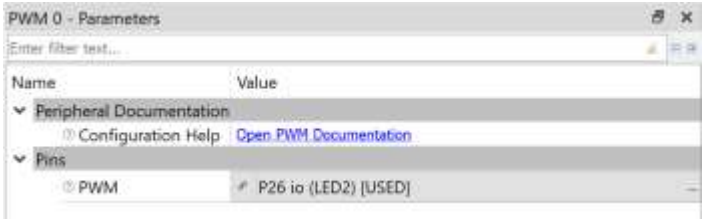
In this exercise, you will replace the thread and instead use a timer to blink an LED. This material is covered in **Error! Reference source not found.**

1. Create another new application called **ch03_ex05_timer** with the same template.
2. Copy app.c from **ch03_ex04_interrupt**.
3. In the C file:
    a. Add an include for wiced_timer.h.
    b. In the BTM_ENABLED_EVT, add calls to initialize and start a periodic timer with a 250ms interval.
    c. Modify the interrupt callback function to serve as the timer callback.
        i. Hint: the body of the timer function is the same as the code you wrote for the interrupt callback, but the function argument list is slightly different.
4. Program your application to the board and test it.

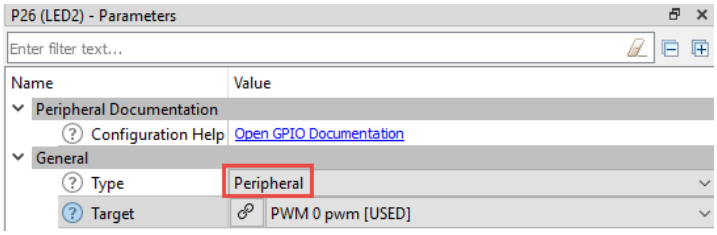
9.2-6 (PWM) LED BRIGHTNESS

In this exercise, you will control an LED using a PWM instead of a GPIO. The PWM will toggle the LED too fast for the eye to see, but by controlling the duty cycle you will vary the apparent brightness of the LED. This material is covered in **Error! Reference source not found.**

1. Create a new application called **ch03_ex06_pwm** using the modus.mk file in the original location of templates/CYW920819EVB/ch02.
2. Click Configure Device from the Quick Panel or double-click the design.modus file to open it in the Device Configurator.
    a. Find PWM0 in the list of peripherals and click the checkbox to enable it.
    b. In the Parameters panel use the PWM drop-down menu to choose P26 (LED2) for the output.
    c. Find P26 (LED2) in the list of peripherals.
    d. Hint: the PWM Parameter panel should look like this. You can jump straight to the Parameters for P26 by clicking the “link” button to the left of the drop-down menu.



    e. Change the type of the pin from LED to Peripheral.
    f. Hint: The Pin Parameter panel should look like this. Note that the PWM output is already selected.



    g. Save the changes and close the configurator.

   h. Hint: The PWM routing can be done in code rather than using the configurator. In fact, the solution project does not use the configurator method.
3. In the app.c file:
   a. Add a #include for the PWM functions – "wiced_hal_pwm.h"
   b. Configure PWM0 with an initial period of 100 and a duty cycle of 50%.
   c. Hint: Use LHL_CLK as the source clock since the exact period of the PWM doesn't matter as long as it is faster than the human eye can see (~50 Hz).
4. Update the duty cycle in the thread function so that the LED gradually cycles through intensity values from 0 to 100%.
   a. Hint: Change the delay in the thread function to 10ms so that the brightness changes relatively quickly.
5. Program the application to the board and test it.

9.2-7 (PWM) LED TOGGLING AT SPECIFIC FREQUENCY AND DUTY CYCLE

In this exercise, you will use a PWM with a period of 1 second and a duty cycle of 10% so that the LED will blink at a 1 Hz rate but will only be on for 100ms each second. In the prior exercise you configured the PWM and then the pin. This time do it by configuring the pin first. This material is covered in **Error! Reference source not found.**

1. Create a new application called **ch03_ex07_pwm_blink** using the modus.mk file in templates/CYW920819EVB/ch02.
2. Click Configure Device from the Quick Panel or double-click the design.modus file to open it in the Device Configurator.
   a. Find P27 in the list of Pins and change its type to Peripheral.
   b. Choose PWM1 from the Target drop-down menu
   c. Click the link button to jump to the PWM – notice that it is already enabled and configured.
   d. Save the file and close the Device Configurator.
3. In the app.c file:
   a. Hint: Don't forget to include the header files for the PWM and ACLK functions.
   b. Initialize the aclk with a frequency of 1 kHz.
   c. Change the PWM1 configuration to use PMU_CLK as the source. Set the period to 1000 and set the duty cycle to 10%.
4. As you did in ex05, remove or comment out the calls to wiced_rtos_create_thread(), wiced_rtos_init_thread() and the thread function because, even though it would not affect the program functionally, it is not a good practice to write to pins that are driven by a peripheral Program the application to the board and test it.

9.2-8 (I2C) READ MOTION SENSOR DATA

In this exercise, you will use the I2C master to read 3-axis acceleration data from the LSM9DS1 motion sensor that is included on the kit. The values will be printed to the PUART. This material is covered in **Error! Reference source not found.**

1. Create a new application called **ch03_ex08_i2c_motion** using the modus.mk file in templates/CYW920819EVB/ch03_ex08_i2c_motion.
2. In app.c, look for TODO comments to configure the motion sensor and read values using I2C.

   a. Hint: If you use the wiced_hal_i2c_combined_read() function, which writes an offset and then reads data from that location, it expects the read arguments first and the wrote arguments second.

3. The LSM9DS1 motion sensor on the kit has the following properties:
   a. 0x6A: device I2C address
   b. 0x20: address of configuration register to enable the accelerometer
   c. 0x40: value for configuration register to provide 2g acceleration data at 50Hz
   d. 0x28: address of first acceleration data register. The values in order are:
      i. 0x28: X_LSB
      ii. 0x29: X_MSB
      iii. 0x2A Y_LSB
      iv. 0x2B Y_MSB
      v. 0x2C: Z_LSB
      vi. 0x2D: Z_MSB
   e. Hint: Search online for the LSM9DS1 datasheet if you want to explore other settings.
4. Program your application to the board and test it.

9.2-9 (ADVANCED) (NVRAM) WRITE AND READ DATA IN THE NVRAM

In this exercise, you will store a 1-byte value in the NVRAM. Mechanical button MB1 (WICED_GPIO_PIN_BUTTON_1) will increment the value each time it is pressed and print the new value. This material is covered in **Error! Reference source not found.**

1. Create another new application called **ch03_ex09_nvram** using the template in templates/CYW920819EVB/ch03_ex09_nvram (NOT the I2C template).
2. Add a #include for the NVRAM API functions.
3. Call wiced_set_debug_uart() to use the PUART.
4. In the interrupt callback function:
    a. Read the value from NVRAM at location WICED_NVRAM_VSID_START.
    b. Print out the value, the number of bytes read, and the status of the read operation (not the return value).
    c. Increment the value.
    d. Write the new value into NVRAM at location WICED_NVRAM_VSID_START.
    e. Print out the value, the number of bytes written, and the status of the write operation.
5. Open a terminal window and program the kit. Wait a few seconds and then press Button 1 a few times to observe the results.
6. Reset the kit. Notice that the previously stored value is retained.
7. Unplug the kit, plug it back in, and reset the terminal. Notice that the previously stored value is retained.

9.2-9A QUESTIONS

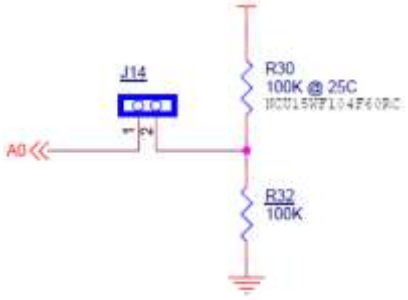
1. How many bytes does the NVRAM read function get when you press the button the first time?

2. What is the return status value when you press the button the first time?

3. What does the return value mean?

9.2-10 (ADVANCED) (ADC) CALCULATE THE RESISTANCE OF A THERMISTOR

In this exercise you will measure the voltages, in millivolts, of Vddio (supply) and across the thermistor balance resistor (100kOhm). Then you will use that data to calculate the resistance of the thermistor. For reference, here is the thermistor circuit. This material is covered **Error! Reference source not found.**



The thermistor shares pin P08 (CYW20819EVB) or P10 (CYBT213043-MESH) with A0 on the Arduino header. You need to make sure the THERMISTOR_ENABLE jumper is attached to read the voltage.

1. Create a new application called **ch03_ex10_adc** using the modus.mk file in templates/CYW920819EVB/ch02.
2. Add in the include file for the ADC functions.
3. Add the call to wiced_set_debug_uart() to enable printing to the PUART.
4. In the management callback function, initialize the ADC.
5. In the application thread function, use the ADC to read Vddio and the voltages across the balance resistor.
    a. Hint: The ADC has a dedicated channel to read the supply voltage called ADC_INPUT_VDDIO.
6. Calculate and print the resistance of the thermistor periodically.
    a. Hint: The formula is as follows.
    $$Rtherm = \frac{(Vddio - Vmeas) * BALANCE\_RESISTANCE}{Vmeas}$$
7. Program the board and open a terminal window with a baud rate of 115200. Alternately place and remove your finger from the thermistor (next to the THERMISTOR_ENABLE jumper) and observe the value displayed in the terminal.

9.2-11 (ADVANCED) (UART) SEND A VALUE USING THE STANDARD UART FUNCTIONS

In this exercise, you will use the standard UART functions to send a value to a terminal window. The value will increment each time a mechanical button on the kit is pressed. This material is covered in **Error! Reference source not found.**

1. Create a new application called **ch03_ex11_uartsend** using the modus.mk file in templates/CYW920819EVB/ch02.
2. Copy the app.c file from the **ch03_ex04_interrupt** application. Remember to copy, not move!
3. In app.c, remove the call to wiced_set_debug_uart() if your interrupt exercise used the PUART.
4. Initialize the UART with Tx enabled, baud rate of 115200, and no flow control.
5. Modify the interrupt callback so that each time the button is pressed a variable is incremented and the value is sent out over the UART. For simplicity, just count from 0 to 9 and then wrap back to 0 so that you only have to send a single character each time.

   a. Hint: Use a “static” variable in the callback function to remember the last number printed.
6. Program the board and open a terminal window with a baud rate of 115200. Press the button and observe the value displayed in the terminal.

9.2-12 (ADVANCED) (UART) GET A VALUE USING THE STANDARD UART FUNCTIONS

In this exercise, you will learn how to read a value from the UART rather than sending a value like in the previous exercise. The value entered will be used to control an LED on the board (0 = OFF, 1 = ON). This material is covered in **Error! Reference source not found.**

1. Create a new application called **ch03_ex112_uartreceive** using the modus.mk file in templates/CYW920819EVB/ch02.
2. Copy the app.c file from the **ch03_ex11_uartsend** project. Remember to copy, not move!
3. Update the code to initialize the UART with Rx enabled, baud rate of 115200, no flow control, and an interrupt generated on every byte received.
   a. Hint: you can remove the code for the button press and its interrupt, but you will need to register a UART Rx interrupt callback instead.
4. In the interrupt callback, read the byte. If the byte is a 1, turn on LED_2. If the byte is a 0, turn off the LED. If you wish you can also print LED_ON/OFF messages. Ignore any other characters.
5. Program your application to the board.
6. Open a terminal window with a baud rate of 115200.
7. Press the 1 and 0 keys on the keyboard and observe the LED turn on/off.

9.2-13 (ADVANCED) (RTC) DISPLAY TIME AND DATE DATA ON THE UART

In this exercise you will set the time and date after reset and thereafter display a clock on the UART. This material is covered in **Error! Reference source not found.**

1. Create a new application called **ch03_ex13_rtc** using the modus.mk file in templates/CYW920819EVB/ch03_ex13_rtc.
2. In app.c, look for TODO comments and add code to control the RTC as follows.
    a. Initialize the RTC.
    b. Set the time and date. Make sure you understand how the getDateTimeEntry() function works and gets used.
    Hint: the template code includes ring_pop() and ring_push() functions that implement a ring buffer for the UART – the UART interrupt code pushes received characters into the buffer and the code that sets the time and date pulls characters from it.
    c. Display the clock with a precision of 1s.
3. Program your application to the board and test it.

9.3 (ADVANCED) EXERCISES FOR RTOS

9.3-1 SEMAPHORE

Create a program where an interrupt looks for a button press then sets a semaphore to communicate to the toggle LED thread.

1. Create a new application called **ch09_ex01_semaphore** using the template in the "templates/CYW920819EVB/ch03" folder.
2. Create a new semaphore pointer as a global variable, then create and initialize the semaphore when the Bluetooth stack is enabled.
    a. Hint: You need both a create function call and an initialize function call.
    b. Hint: Be sure to create and initialize the semaphore before starting the LED thread or the interrupt (added in the next step) since they use the semaphore.
3. Use the provided interrupt callback function to look for a button press and set the semaphore.
    a. Hint: Refer to the interrupt exercise from the peripherals chapter.
4. Get the semaphore inside the LED thread so that it waits for the semaphore forever and then toggles the LED rather than blinking constantly.
    a. Hint: Use WICED_WAIT_FOREVER so that the thread will wait until the button is pressed. The definition for this can be found at the top of wiced_rtos.h.

9.3-1A QUESTIONS

1. Do you need wiced_rtos_delay_milliseconds() in the LED thread? Why or why not?

9.3-2 (ADVANCED) MUTEX

An LED may behave strangely if two threads try to blink it at the same time. In this exercise we will use a mutex to lock access.

1. Create a new application called ch09_ex02_mutex using the template in the "templates/CYW920819EVB/ch09_ex02_mutex" folder.

2. The project has 2 threads – one thread blinks the LED at a rate of 2Hz and the other thread blinks the same LED at a rate of 5Hz when the button is being pressed.
3. Program the project as it is to your kit. What happens when you hold down the button? Does the LED blink at 5Hz?
4. Look at the TODO comments to add a mutex to the two threads so that each thread prevents the other from blinking the LED when it needs access.
5. Program the project to your kit. Now what happens when you hold down the button?

9.3-2A QUESTIONS

1. Before you added the mutex, how did the LED behave when you pressed the button?

2. What changed when you added the mutex?

3. What happens if you forget to unlock the mutex in one of the threads? Why?

9.3-3 (ADVANCED) QUEUES

Use a queue to send a message to indicate the number of times to blink an LED.
Create a new application called ch09_ex03_queue using the template in the "templates/CYW920819EVB/ch03" folder (not the mutex template).

1. The queue API uses memory from the buffer pools that are defined in app_bt_cfg.c. By default, there are insufficient pools to support queues and so you need to modify the value of wiced_bt_cfg_settings.max_number_of_buffer_pools to allocate more memory. You should add a buffer pool for each queue that your application will create (in this case, increase from 4 to 5).
2. In app.c, create a global pointer to a queue and, when the stack gets enabled, create and initialize the queue
    a. Hint: Use a message size of 4 bytes (room for one uint32_t) and a queue length of 10 so you can push messages while the thread is blinking without causing the queue to overflow.
3. Add a static variable to the interrupt callback that increments each time the button is pressed. Push the value onto the queue to give the LED thread access to it.
4. In the LED thread, pop the value from the queue to determine how many times to blink the LED.
    a. Hint: Add a longer delay (e.g. 1 second) after the LED blinks the specified number of times so that you can tell the button press sequences apart.
5. Program your project to the board. Press the button a few times to see how the number of blinks is increased with each press. Note that you can press the button while it is currently blinking, and the new press will be added to the queue (provided that the queue is large enough).

9.3-4 (ADVANCED) PRINT THE THREAD STACK USAGE

Add a function to determine the amount of stack used by a thread and print it to the UART.

1. Create a new application called ch09_ex04_stack_size using the template in the "templates/CYW920819EVB/ch03" folder.
2. Copy app.c from ch09_ex01_semaphore.
3. Configure the debug uart to output to the PUART.
    a. Hint: See exercise 2 from the peripherals chapter if you need a reminder.
4. Add a function to calculate the max stack used by the LED thread.
5. Call the stack size function and print the value to the UART each time the LED is toggled (i.e. whenever the semaphore get function returns).

9.4 EXERCISES – BLE BASICS (COPIED FROM WBT101-04A)

9.4-1 CREATE A BLE PROJECT WITH A MODUSLED SERVICE

Use the template in folder "templates/CYW920819EVB/ch04a" to create a project called **ch04a_ex01_ble**.

Follow the instructions in section **Error! Reference source not found.**to use the Bluetooth Configurator to set up a Service called ModusLED with a Characteristic called LED that allows an LED on the kit to be controlled from your phone using CySmart.
Hint: The template app.c file has comments marked with “TODO” for locations that need changes for exercises 1 and 2.

9.4-2 IMPLEMENT A CONNECTION STATUS LED

9.4-2A INTRODUCTION

In this exercise, you will implement a connection status LED that is:

- Off – when the device is not advertising
- Blinking – when the device is advertising
- On – when there is a connection

Hint: Use LED_1 for the connection status since LED_2 is used by other projects for the LED Characteristic.

9.4-2B PROJECT CREATION

1. Use the ch04a template to create a project called **ch04a_ex02_status**.
2. Launch the Device Configurator.
    a. Enable PWM0.
    b. Connect PWM0 output to LED_1.
    c. Change the type of LED_1 from “LED” to “Peripheral”.
    d. Save your edits.
    e. Note: In the solution file this step is accomplished with a line of code in the app.c file.
3. Launch Bluetooth Configurator.
    a. Set the device name to <inits>_status. Make sure you hit enter or click in a field outside the Value. If not, the new value may not be saved.
    b. Save your changes and close the configurators. We are only using the GATT database to handle connections in this exercise and so there is no need to add a service.
4. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
5. Hint: The template app.c file has comments marked with "TODO" for locations that need changes for exercises 1 and 2.
6. Open app.c and #include "GeneratedSource/cycfg_gatt_db.h"
    a. Hint: If you don't see cycfg_gatt_db.h in the GeneratedSource folder, right click on the folder and select "Refresh".
7. #include "wiced_hal_pwm.h"
8. #include "wiced_hal_aclk.h"
9. In the BTM_ENABLED_EVT case, start the ACLK and PWM but set the compare value to match the maximum value so that the LED is always off.

```
/* Start the PWM in the LED always off state */
wiced_hal_aclk_enable( PWM_FREQUENCY, ACLK1, ACLK_FREQ_24_MHZ );
wiced_hal_pwm_start( PWM0, PMU_CLK, PWM_ALWAYS_OFF, PWM_INIT, 0 );
```

    a. Hint: Use the pre-defined macros for PWM_ALWAYS_ON, PWM_ALWAYS_OFF, and PWM_TOGGLE from the template to make the PWM code easier to write.

b. Hint: You may see some items underlined in red before you build – this are things that are in the new includes that you added.

10. In the BTM_ENABLED_EVT case, enable the GATT database and disallow pairing.

```
/* Configure the GATT database and advertise for connections */
wiced_bt_gatt_register( app_gatt_callback );
wiced_bt_gatt_db_init( gatt_database, gatt_database_len );
wiced_bt_set_pairable_mode( WICED_FALSE, WICED_FALSE );
```

11. Declare a global uint16_t variable to keep track of the connection ID and initialize to 0.
    a. Hint: This will be needed so that when advertisements stop, you will know if the LED should be turned ON (connected) or OFF (not connected).
12. Set/clear the connection ID variable at the appropriate places.
    a. Hint: Look in the GATT connect callback function.
        i. For a connection: connection_id = p_conn->conn_id;
        ii. For a disconnection: connection_id = 0;
13. Turn the LED ON or OFF when advertising stops based on the connection ID.
    a. Hint: In the BTM_BLE_ADVERT_STATE_CHANGED_EVT case, create a switch statement for p_event_data->ble_advert_state_changed that handles the following cases; BTM_BLE_ADVERT_OFF (sets the LED on or off based on connection_id), BTM_BLE_ADVERT_UNDIRECTED_HIGH and BTM_BLE_ADVERT_UNDIRECTED_LOW (both set the PWM to toggle).
        i. Hint: use wiced_hal_pwm_change_values and make use of the macros PWM_ALWAYS_OFF, PWM_ALWAYS_ON, and PWM_TOGGLE that are provided in the template.
14. Note: If you are impatient, like us, you can speed up your testing by changing the values of high_duty_duration and low_duty_duration to 15 in app_bt_cfg.c, which will make the stack change advertising state much faster.

9.4-2C TESTING

1. Program the application to your kit.
2. Use the PC version of CySmart to connect to the kit. Observe the state of LED_1 when not advertising, when advertising and when a connection is active.
    a. Hint: you must have a CY5677 CySmart BLE USB dongle connected to your PC to run CySmart.
    b. Hint: Don't forget to update the scan interval and window to 1000 and 100 ms respectively.
    c. Hint: You will have to wait for the advertising timeout while not connected to see the first case.

9.4-3 CREATE A BLE ADVERTISER

9.4-3A INTRODUCTION

In this exercise, you will create a project that will send out advertisement packets but will not allow any connections. This is common for devices like beacons or locator tags. The advertisement packet will include the flags, complete name, appearance and three bytes of manufacturer specific data. Each time

a button is pressed on the kit, the value of the manufacturer data will be incremented, and advertisements will be re-started.

Below is a table showing the events that occur during this exercise. Arrows indicate the cause/effect of the stack events.

External Event	BLE Stack Event	Action
Board reset →	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT →	Not used yet
	BTM_ENABLED_EVT →	Initialize application, start the button interrupt
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_NONCONN_HIGH)	← Start advertising
Scan for devices in CySmart PC application. Look at advertising data.		
Press MB1.	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_NONCONN_HIGH)	← Update information in the advertising packet and restart advertising
Re-start scan in CySmart. Look at new advertising data.		
Wait for timeout. →	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_NONCONN_LOW)	Stack switches to lower advertising rate to save power

9.4-3B PROJECT CREATION

1. Use the template to create a project called **ch04a_ex03_adv**.
2. Launch the Device Configurator and open Bluetooth Configurator.
    a. Set the device name to <inits>_adv. Make sure you hit enter or click in a field outside the Value. If not, the new value may not be saved.
    b. In the “Appearance” characteristic, choose the “Generic Tag” type.
    c. Save your changes and close the configurators.
3. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
4. Hint: The template app.c file does NOT have TODO comments for the locations to change for the remaining exercises.
5. Open app.c and #include "GeneratedSource/cycfg_gatt_db.h"
6. Locate the line in the main C file that starts advertisements. Change the advertisement type to *BTM_BLE_ADVERT_NONCONN_HIGH* because we don't want the device to be connectable.
    a. Hint: Right click on the existing advertisement type and select *Open Declaration* to see all the available choices.
7. Locate the function that sets up the advertisement data and add a new element to send Cypress’ unique manufacturer ID and a count value.
    a. Hint: Create a global *uint8_t* array of size three. Set the first two values equal to 0x31 and 0x01. The third value will hold the count value.

i. The Cypress manufacturer ID assigned by the Bluetooth SIG is 0x0131. The value is little endian in the advertising packet which is why the first two bytes are 0x31 and 0x01.

b. Hint: The advertisement type for this element should be *BTM_BLE_ADVERT_TYPE_MANUFACTURER*.

c. Hint: don't forget to increase the number of elements in the advertising data array.

8. Configure Button1 for a falling edge interrupt in the BTM_ENABLED_EVT. Add a button interrupt callback to do the following:
   a. Clear the pin interrupt.
   b. Increment the third byte of the array holding the manufacturer's data (i.e. the count value).
   c. Update the advertisement packet data array
      i. Hint: you can just call the function that sets up the advertising packet again.

9.4-3C TESTING

1. Program the project to the board and use the PC version of CySmart to examine the advertisement packets. Start scanning and the stop once you see your device listed. Then click on your device to see its advertisement data. Press the button, re-start/stop the scan, and look at your device's scan response to see that the value has incremented.
   a. Hint: Don't forget to update the scan interval and window to 1000 and 100 ms respectively.

9.4-3D QUESTIONS

1. How many bytes is the advertisement packet?

9.4-4 CONNECT USING BLE

9.4-4A INTRODUCTION

In this exercise, you will create a project that will have a custom Service called "Modus101" containing two Characteristics:

1. A Button characteristic with the state of the button on the kit
2. An LED Characteristic to control an LED.

You will monitor the button on the kit board and update its state in a GATT Characteristic so that a client can read the value. The LED Characteristic will behave like the LED in exercise 01 – you will be able to Read and Write the LED state from a client to control the LED on the board.

Below is a table showing the events that occur during this exercise. Arrows indicate the cause/effect of the stack events. New events introduced in this exercise are highlighted.

External Event	BLE Stack Event	Action
Board reset →	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT →	Not used yet
	BTM_ENABLED_EVT →	Initialize application
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_ UNDIRECTED _HIGH)	← Start advertising

CySmart will now see advertising packets		
Connect to device from CySmart →	GATT_CONNECTION_STATUS_EVT →	Set the connection ID and enable pairing
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_OFF)	
Read Button characteristic while touching buttons →	GATT_ATTRIBUTE_REQUEST_EVT, GATTS_REQ_TYPE_READ →	Returns button state
Disconnect →	GATT_CONNECTION_STATUS_EVT →	Clear the connection ID and re-start advertising
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_UNDIRECTED_HIGH)	
Wait for timeout. →	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_ UNDIRECTED _LOW)	Stack switches to lower advertising rate to save power
Wait for timeout. →	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_OFF)	Stack stops advertising.

9.4-4B PROJECT CREATION

1. Use the template to create a project called **ch04a_ex04_con**.
2. Launch the Device Configurator
    a. Enable PWM0 and connect it to LED1.
    b. Set LED1 type to Peripheral.
    c. Save your changes.
3. From the Device Configurator, open the Bluetooth Configurator.
    a. Set the device name to <inits>_con. Make sure you hit enter or click in a field outside the Value. If not, the new value may not be saved.
    b. Add a new custom service to the GATT server and rename it “Modus101”.
        i. Note: Right click on the service and choose "rename" to rename it.
    c. Rename the custom characteristic to “LED”
    d. Configure LED to be a uint8 that is initially 0 and enable both read and write.
    e. Add a new custom characteristic alongside LED.
    f. Rename it to “Button”.
    g. Configure Button to be a uint8_t that is initially 0 but only enable it for read.
4. Save your changes and close both the configurators.
5. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
6. Open app.c and #include "GeneratedSource/cycfg_gatt_db.h"
7. #include "wiced_hal_pwm.h"
8. #include "wiced_hal_aclk.h"
9. You are going to re-create the advertising status LED (on LED1) behavior from ex02_status

a. Create a global uint16_t for the connection ID.
b. Add code to set and clear the variable in the GATT_CONNECTION_STATUS_EVT event in the GATT callback handler.
c. Start the PWM when the BLE stack comes up (ALWAYS_OFF)
d. Add code to change the blink behavior in the BTM_BLE_ADVERT_STATE_CHANGED_EVT event in the management callback function.

10. Enable GATT connections and disallow pairing.

```
/* Configure the GATT database and advertise for connections */
  wiced_bt_gatt_register( app_gatt_callback );
  wiced_bt_gatt_db_init( gatt_database, gatt_database_len );

  /* Disable pairing */
  wiced_bt_set_pairable_mode( WICED_FALSE, WICED_FALSE );
```

11. Add the code to the app_gatt_set_value function from exercise 01 to set the state of LED_2 based on the LED Characteristic value.
    a. Hint: The name of the case (i.e. handle for the LED characteristic value) and the array will be different because the project name and service names are different. The names can be found in the GATT definitions and the GATT Initial Value Arrays section of the GATT database code.
12. You may want to build/program/test to make sure everything works up until this point. You should be able to connect and see one service with two Characteristics. The Read/Write Characteristic should still control LED_2.
13. Configure the button for an interrupt on both edges. In the interrupt callback, save the current state of the button to the appropriate GATT array.
    a. Hint: On the CYW920819EVB-02 kit, invert the value before storing it in the array since the button is active low and we want the button Characteristic value to be high when the button is pressed. This is not necessary on the CYBT-213043-MESH kit because the button is active high.
    b. Hint: You can find the name of the array in the GATT Initial Value Arrays section of the GATT database code.

9.4-4C TESTING

1. Program the project to the board.
2. Open the mobile CySmart app.
3. Connect to the device.
4. Open the GATT browser widget and then open the Modus101 Service followed by the Button Characteristic.
5. Read the value while both pressing and not pressing the button to see the values.
6. Switch to the LED characteristic and verify that it still works to turn the LED ON/OFF.
7. Disconnect from the mobile CySmart app and start the PC CySmart app.
8. Start scanning and then connect to your device.
9. Click on "Discover all Attributes".
10. Read the button value in CySmart by clicking on the Characteristic and then clicking the "Read Value" button. Continue reading as you press and release the button and verify that the value is correct.

11. Click "Disconnect".

9.4-4D QUESTIONS

1. What function is called when there is a Stack event? Where is it registered?

2. What function is called when there is a GATT database event? Where is it registered?

3. Which GATT events are implemented? What other GATT events exist? (Hint: right click and select Open Declaration on one of the implemented events)

4. In the GATT "GATT_ATTRIBUTE_REQUEST_EVT", what request types are implemented? What other request types exist?

9.5 EXERCISES – ADVANCED BLE PERIPHERALS (COPIED FROM WBT101-04B)

9.5-1 SIMPLE BLE PROJECT WITH NOTIFICATIONS USING BLUETOOTH CONFIGURATOR

In this exercise you will create a simple example of notifications without pairing the phone and kit (so the notifications are not authenticated).

1. Create a new application called **ch04b_ex01_ctr** using the modus.mk file in templates/CYW920819EVB/**ch04b**.
2. Follow the instructions in section **Error! Reference source not found.**to use the Bluetooth Configurator to create a project with a Service called Modus and a Characteristic called Counter that will keep track of how many times mechanical button 1 has been pressed and will send a notification if it is enabled by the client.

   Hint: Remember to use your initials in the device name so that you can find it in the list of devices that will be advertising.

   Hint: Remember to set the BT_DEVICE_ADDRESS = random to avoid advertising the same address as other students.

9.5-2A INTRODUCTION

In this exercise, you will add Pairing and Security (Encryption) to the previous project.
Below is a table showing the events that occur during this exercise. Arrows indicate the cause/effect of the stack events. New events introduced in this exercise are highlighted.

External Event	BLE Stack Event	Action
Board reset →	BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT →	Not used yet.
	BTM_ENABLED_EVT →	Initialize application.
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_ UNDIRECTED _HIGH)	← Start advertising
CySmart will now see advertising packets		
Connect to device from CySmart →	GATT_CONNECTION_STATUS_EVT →	Set the connection ID and enable pairing
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_OFF)	
Pair →	BTM_SECURITY_REQUEST_EVT →	Grant security
	BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT→	Capabilities are set
	BTM_ENCRYPTION_STATUS_EVT	Not used yet
	BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT	Not used yet
	BTM_PAIRING_COMPLETE_EVT	Not used yet
Read Button characteristic while pressing button →	GATT_ATTRIBUTE_REQUEST_EVT, GATTS_REQ_TYPE_READ →	Returns button state
Read Button CCCD →	GATT_ATTRIBUTE_REQUEST_EVT, GATTS_REQ_TYPE_READ →	Returns button notification setting
Write 01:00 to Button CCCD →	GATT_ATTRIBUTE_REQUEST_EVT, GATTS_REQ_TYPE_WRITE →	Enables notifications
Press button →		Send notifications
Disconnect →	GATT_CONNECTION_STATUS_EVT →	Clear the connection ID
	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_UNDIRECTED_HIGH)	Re-start advertising
Wait for timeout →	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_ UNDIRECTED _LOW)	Stack switches to lower advertising rate to save power
Wait for timeout →	BTM_BLE_ADVERT_STATE_CHANGED_EVT (BTM_BLE_ADVERT_OFF)	Stack stops advertising

9.5-2B PROJECT CREATION

1. Create a new application called **ch04b_ex02_pair** using the modus.mk file in templates/CYW920819EVB/**ch04b_ex02_pair**. This template is the same as project ex01_ctr that you just completed. If you prefer to build on your own code, use the modus.mk file from that project instead of the template.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device Configurator and then the Bluetooth Configurator.
    a. Change the Device Name to <inits>_pair.
    b. In the Counter characteristic set the Read (authenticated) permission, which will make the peripheral reject read requests unless the devices are paired.
    c. Update the Client Characteristic Configuration descriptor to require authenticated read and write. This will cause the application to require pairing to view or change the notification settings.
    d. Save the edits and close the configurators.
4. In app.c, look for the call to `wiced_bt_set_pairable_mode()` mode and set the first argument to WICED_TRUE to allow pairing.
5. In the `BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT` management case tell the central that you require MITM protection but the device has no IO capabilities.

```
p_event_data->pairing_io_capabilities_ble_request.auth_req =
BTM_LE_AUTH_REQ_SC_MITM_BOND;
p_event_data->pairing_io_capabilities_ble_request.init_keys =
BTM_LE_KEY_PENC|BTM_LE_KEY_PID;
p_event_data->pairing_io_capabilities_ble_request.local_io_cap =
BTM_IO_CAPABILITIES_NONE;
```

6. In the `BTM_SECURITY_REQUEST_EVT` management case grant the authorization to the central by using the following code:

```
wiced_bt_ble_security_grant( p_event_data-
>security_request.bd_addr, WICED_BT_SUCCESS );
```

9.5-2C TESTING

1. Build and program the project to the board.
2. Open the mobile CySmart app.
3. Connect to the device.
4. Open the GATT browser, navigate to the Counter characteristic, press Descriptor and then the Client Characteristic Configuration.
5. If requested, accept the invitation to pair the devices.
6. Return to the Counter, enable notifications and observe the Counter value as you press the button on the kit.
7. Disconnect from the mobile CySmart app.
8. Go to the phone's Bluetooth settings and remove the <inits>_pair device from the paired devices list. This is necessary so that when you re-program the kit the phone won't have stale bonding information stored which could prevent you from re-connecting. In the next exercise we'll store bonding information on the WICED device so that you will be able to leave the devices paired if you desire.
9. Start the PC CySmart app. Scan for your device. Once it appears in the list, stop scanning and connect to it.
10. Click on "Discover all Attributes" and then on "Enable Notifications". Notice that you will get an authentication error. Click "OK" to close the error dialog.
11. Try reading the Counter Characteristic Value manually. Notice that you again get an authentication error. Click "OK" to close the error window.
12. Click on "Pair" and click "No" if you are asked if you want to add the device to the resolving list since we haven't yet enabled privacy.
13. Click on "Enable All Notifications" again. Now when you press the button you will see the characteristic value change.
14. Click on "Disable All Notifications" and then read the Button Characteristic Value manually. It should now work.
15. Click "Disconnect".
16. From the Device List, select a Device Address and select "Clear -> All" since we have not stored bonding information in the device yet.

9.5-2D QUESTIONS

1. How long does the device stay in high duty cycle advertising mode? How long does it stay in low duty cycle advertising mode? Where are these values set?

9.5-3 (ADVANCED) SAVE BLE PAIRING INFORMATION (I.E. BONDING) AND ENABLE PRIVACY

9.5-3A INTRODUCTION

The prior exercise has been modified for you to save and restore bonding information to NVRAM. You will create the application from a template, program it to your kit, experiment with it, and then answer questions about the stack events that occur.

By saving Bonding information on both sides (i.e. the client and the server) future connections between the devices can be established more quickly with fewer steps. This is particularly useful for devices that require a pairing passkey (which will be added in the next exercise) since saving the bonding information means the passkey doesn't have to be entered every time the device connects.

Moreover, since the keys are saved on both devices, they don't need to be exchanged again. This means that after the first connection, there is no possibility of a MITM attack since the keys are not sent out over the air.

The firmware has two "modes": *bonding mode* and *bonded mode*. After programming, the kit will start out in bonding mode. LED1 will blink slowly (1 sec duty cycle) to indicate that the kit is waiting to be Paired/Bonded. Once a Client connects to the kit and pairs with it, the Bonding information will be saved in non-volatile memory. The LED will be ON since the kit is connected. The only Client that will be allowed to pair with the kit is the one that is bonded (the firmware only allows 1 bonded device at a time for now). If the Bonding information is removed from the Client, it will no longer be able to Pair/Bond with the kit without going through the Paring/Bonding process again.

When you disconnect, LED1 will flash rapidly (200ms duty cycle) to indicate that it is bonded. To remove Bonding information from the kit and return bonding mode, press 'e' in the UART terminal window. This will erase the stored bonding information and put the kit back into Bonding mode. LED1 will now go back to a slow flashing rate. When you reconnect, the key must be entered again to connect. This allows you to Pair/Bond from a Client that has "lost" the bonding information or to Pair/Bond with a new device without having to reprogram the kit.

9.5-3B PROJECT CREATION

1. Create a new application called **ch04b_ex03_bond** using the modus.mk file in templates/CYW920819EVB/**ch04b_ex03_bond**.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device Configurator and then the Bluetooth Configurator.
    a. Change the Device Name to <inits>_bond.
    b. Save the edits and close the configurators.
4. All the code for this exercise has already been implemented for you.
5. Build and program the kit.

9.5-3C TESTING

1. Open a UART terminal window to the PUART.
2. Build the project and program it to the board.
3. Open the CySmart PC application and connect to the dongle.
4. Click ‘Configure Master Settings’ and, under ‘Privacy 1.2’, change the Address Generation Interval to match the *rpa_refresh_timeout* in *app_bt_cfg.c*.

5. If there is anything listed in the "Device List" near the bottom of the screen, click on any device from the list and choose "Clear > All". This will remove any stored bonding information from CySmart so that it will not conflict with your new firmware. It is necessary to do this each time you re-program the kit so that the old information is not used.
6. Start scanning. Once you see your device in the list stop scanning. Note that your device shows up with a Random Bluetooth address now since privacy is enabled.
7. Connect to your device.
8. Click on "Discover all Attributes".
9. Click on "Pair" and click "Yes" when asked if you want to add the device to the resolving list so that the privacy keys will be remembered by CySmart.
    a. Note down the Bluetooth Stack events that occur during pairing. This information is displayed in the UART.
10. Click on "Enable All Notifications". Press the button and observe the characteristic value changes.
11. Click "Disconnect". Do NOT remove the device from the Device List this time – we want bonding information retained.
12. Start a new scan and stop when your device appears in the list.
13. Notice how the Address is now listed as a Public Identity Address rather than Random in the table of discovered devices. Look at the Device List window at the bottom; both the Device Address and the Public Identity Address are listed. If you click on 'View …', some Details concerning the device appear including the Identity Resolving Key. The IRK is used to map the Private Random Address to the Public Identity Address.
14. Re-connect to your device.
15. Click on "Discover all Attributes" and "Pair".
    a. Once again note down the Bluetooth Stack events that occur during pairing. You will notice that fewer steps are required this time.
16. Press the button and observe that notifications are already enabled since they were enabled when you disconnected. This information was retained in NVRAM.
    a. Note: If you use the mobile CySmart app, notifications will be disabled when you reconnect because the app automatically disables notifications before disconnecting.
17. Disconnect again.
18. Reset or power cycle the board.
    a. Hint: If you power cycle the board, you will need to either reset or re-open the UART terminal window.
19. Start scanning, stop when your device appears, and then connect to your device for a third time.
20. Click on "Discover all Attributes" and "Pair".
    a. Note down the Bluetooth Stack events that occur this time during pairing. Compare to the previous two connections.
21. Note that notifications are still enabled.
22. Disconnect again.
23. Clear the Device List at the bottom of the CySmart window (i.e. click on any device listed and do "Clear -> All".
24. Start scanning and stop when your device appears. Notice that it again has a Random address type.

25. Connect to your device, "Discover all Attributes" and then try to "Pair". Note that pairing will not complete because CySmart no longer has the required keys to use.
    a. Hint: If you look in the UART window you will see a message about the security request being denied.
    b. Hint: It will take a while before pairing times out.
26. Click on Disconnect and close the Authentication failed message window.
27. Press "e" in the UART window and note that LED1 begins flashing slowly. This indicates that the bonding information has been cleared from the device and it will now allow a new connection.
28. Connect, Discover Attributes, and Pair again. This time it should work.
29. Note the steps that the firmware goes through this time.
30. Disconnect a final time and clear the Device List so that the saved boding information won't interfere with the next exercise.
    a. Hint: You should clear the bonding information from CySmart anytime you are going to reprogram the kit since it will no longer have the bonding information on its side.

9.5-3D OVERVIEW OF CHANGES

1. A structure called "hostinfo" is created which holds the BD_ADDR of the bonded device and the value of the Button CCCD. The BD_ADDR is used to determine when we have reconnected to the same device while the CCCD value is saved so that the state of notifications can be retained across connections for bonded devices.
2. Before initializing the GATT database, existing keys (if any) are loaded from NVRAM. If no keys are available this step will fail so it is necessary to look at the result of the NVRAM read. If the read was successful, then the keys are copied to the address resolution database and the variable called "bonded" is set as TRUE. Otherwise, it stays FALSE, which means the device can accept new pairing requests.
3. In the BTM_SECURITY_REQUEST_EVENT look to see if bonded is FALSE. Security is only granted if the device is not bonded.
4. In the Bluetooth stack event *BTM_PAIRING_COMPLETE_EVT* if bonding was successful write the information from the hostinfo structure into the NVRAM and set bonded to TRUE.
    a. This saves hostinfo upon initial pairing. This event is not called when bonded devices reconnect.
5. In the Bluetooth stack event *BTM_ENCRYPTION_STATUS_EVT*, if the device is bonded (i.e. bonded is TRUE), read bonding information from the NVRAM into the hostinfo structure.
    a. This reads hostinfo upon a subsequent connection when devices were previously bonded.
6. In the Bluetooth stack event BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT, save the keys for the peer device to NVRAM.
7. In the Bluetooth stack event BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT, read the keys for the peer device from NVRAM.
8. In the Bluetooth stack event BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT, save the keys for the local device to NVRAM.
9. In the Bluetooth stack event BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT, read the keys for the local device from NVRAM.
10. In the GATT connect callback:

   a. For a connection, save the BD_ADDR of the remote device into the hostinfo structure. This will be written to NVRAM in the BTM_PAIRING_COMPLETE_EVT.
   b. For a disconnection, clear out the BD_ADDR from the hostinfo structure and reset the CCCD to 0.
11. In the GATT set value function, save the Button CCCD value to the hostinfo structure whenever it is updated and write the value into NVRAM.
12. The UART is configured to accept input. The rx_cback function looks for the key "e". If it has been sent, it sets bonded to FALSE, removes the bonded device from the list of bonded devices, removes the device from the address resolution database, and clears out the bonding information stored in NVRAM.
13. The PWM that blinks the LED during advertising has two different rates. The PWM is started during initialization, and then stopped/restarted when a disconnect happens or when bonding information is removed. The blink rate is set depending on the value of bonded.
14. Finally, privacy is enabled in wiced_bt_cfg.c by updating the rpa_refresh_timeout to *WICED_BT_CFG_DEFAULT_RANDOM_ADDRESS_CHANGE_TIMEOUT*.

9.5-3E QUESTIONS

1. What items are stored in NVRAM?

2. Which event stores each piece of information?

3. Which event retrieves each piece of information?

4. In what event is the privacy info read from NVRAM?

5. Which event is called if privacy information is not retrieved after new keys have been generated by the stack?

9.5-4 (ADVANCED) ADD A PAIRING PASSKEY

9.5-4A INTRODUCTION

In this exercise, you will modify the previous exercise to require a Passkey to be entered to pair the device the first time. The Passkey will be randomly generated by the stack and you will send it to the UART. The Passkey will need to be entered in CySmart on the PC or in your Phone's Bluetooth connection settings before Pairing/Bonding will be allowed.

9.5-4B PROJECT CREATION

1. Create a new application called **ch04b_ex04_pass** using the same modus.mk file as in the previous exercise - templates/CYW920819EVB/ch04b_ex03_bond.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device Configurator and then the Bluetooth Configurator.
   a. Change the Device Name to <inits>_pass
   b. Save the edits and close the configurators.

4. In app.c, change the pairing_io_capabilities_ble_request.local_iop_cap from `BTM_IO_CAPABILITIES_NONE` to `BTM_IO_CAPABILITIES_DISPLAY_ONLY`
5. Create a new event case statement for `BTM_PASSKEY_NOTIFICATION_EVT` (it is not already in the template code) in `app_management_callback()` and print the value of the Passkey to the UART.
    a. Hint: Make sure you print something (e.g. asterisks) around the value so that it is easy to find in the terminal window.
    b. Hint: The Passkey must be 6 digits so print leading 0's if the value is less than 6 digits. (i.e. use %06d).
    c. Hint: The key is passed to the callback event as:
    ```
    p_event_data->user_passkey_notification.passkey
    ```
6. Build and program the kit.

9.5-4C TESTING

1. Open a UART terminal window.
2. Open the mobile CySmart app.
3. Attempt to connect to the device and then navigate down to the button characteristic in the GATT browser. You will see a notification from the Bluetooth system asking for the Passkey to be entered. Find the Passkey on the UART terminal window and enter it into the device.
4. Once Pairing and Bonding completes, verify that the application still works.
5. Disconnect and reconnect. Observe that the key does not need to be entered to Pair this time.
6. Disconnect, then remove the bonding information from the phone's Bluetooth settings.
7. Press 'e' in the UART terminal to put the kit into Bonding mode (i.e. erase the stored bonding information) and then reconnect. Observe that the key must be entered again to connect.
8. Disconnect again and remove the bonding information from the phone's Bluetooth settings.
9. Now try the same thing using the PC version of CySmart. It will pop up a window when the Passkey is needed.
    a. Hint: Remember to put the kit into Bonding mode first to remove the phone's Bonding information from the kit. This is necessary since we only allow bonding information from one device to be stored in our firmware. The final exercise will fix that.

9.5-4D QUESTIONS

1. Other than BTM_IO_CAPABILITIES_NONE and BTM_IO_CAPABILITIES_DISPLAY_ONLY, what other choices are available? What do they mean?

2. What additional stack callback event occurs compared to the previous exercise? At what point does it get called?

9.5-5 (ADVANCED) ADD NUMERIC COMPARISON

9.5-5A INTRODUCTION

In this exercise, you will modify the previous exercise to require the user to compare a 6-digit number on both devices to pair the first time. After comparing that both numbers are the same, the user needs to click "Yes" in CySmart on the PC or in your Phone's Bluetooth connection settings and enter "Y" in the UART terminal window for the kit before Pairing/Bonding will be allowed.
Note that the passkey code is kept since it is up to the central to decide which method to use. That is, we have the device capabilities set to BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT. Since the device has both a display and Yes/No input, the central can choose to use either passkey or numeric comparison.

9.5-5B PROJECT CREATION

1. Create a new application called **ch04b_ex05_num** using the modus.mk file in templates/CYW920819EVB/**ch04b_ex05_num**. This is a copy of the solution project for the passkey exercise and so, if you prefer, you can use the your own modus.mk file from the previous project.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device Configurator and then the Bluetooth Configurator.
    a. Change the Device Name to <inits>_num
    b. Save the edits and close the configurators.
4. In app.c, change the pairing_io_capabilities_ble_request.local_iop_cap to `BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`.
5. Create a new event case statement for `BTM_USER_CONFIRMATION_REQUEST_EVT` (it is not already in the template code) in `app_management_callback(`In the event, print the value for the user to confirm with this code.

```
WICED_BT_TRACE("\r\n********************\r\n" );

WICED_BT_TRACE( "\r\nNUMERIC = %06d\r\n\n", p_event_data-
>user_confirmation_request.numeric_value );
WICED_BT_TRACE("\r\n********************\r\n\n" );
wiced_bt_dev_confirm_req_reply( WICED_BT_SUCCESS,
p_event_data->user_confirmation_request.bd_addr );
```

6. Build the application and program the kit.

9.5-5C TESTING

1. Open a UART terminal window.
2. Open the PC CySmart app.
3. Attempt to Connect and then Pair to the device. You will see a notification from CySmart asking for you to verify the number printed by both devices is the same. Find the number on the UART terminal window and click "Yes" if it matches.
4. Once Pairing and Bonding completes, verify that the application still works.
5. Disconnect and reconnect. Observe that the number does not need to be verified to Pair this time.
6. Disconnect, then clear the Device List in CySmart.
7. Enter "e" in the UART window to put the kit into Bonding mode and then reconnect. Observe that the comparison must be done again to connect.
8. Disconnect again and clear the Device List in CySmart.

9.5-6 (ADVANCED) ADD MULTIPLE BONDING CAPABILITY

9.5-6A INTRODUCTION

In this exercise, you will create an application from a template, and use it to bond to up to 4 different devices at one time. Note that this application stores bonding information for multiple devices, it does NOT allow multiple simultaneous BLE connections. That is also possible, but it is not demonstrated by this example.

9.5-6B PROJECT CREATION

1. Create a new application called **ch04b_ex06_mult** using the modus.mk file in templates/CYW920819EVB/**ch04b_ex06_mult**.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device Configurator and then the Bluetooth Configurator.
    a. Change the Device Name to <inits>_mult.
    b. Save the edits and close the configurators.
4. All the code for this exercise has already been implemented for you.
5. Build and program the kit.

9.5-6C TESTING

1. Open a UART terminal window.
2. The device starts out in bonding mode (LED_1 should be flashing slowly – once per second).
3. Open the mobile CySmart app.
4. Discover all attributes in the GATT database, and attempt to Pair with the device.
5. Once Pairing completes, verify that the application still works. The device will now be in “normal mode”.
6. Disconnect from the device. The LED will be flashing rapidly – once every 200ms. Keep the bonding information on the phone.
7. To allow bonding another device, press "e" in the UART terminal - the LED will flash slowly.
    a. Note: If you already have the max number of devices bonded and enter "e" in the UART, it will remove the oldest bonded device before going back into bonding mode.

8. Connect to the device using the PC version of CySmart and Pair with the device.
9. Verify that the application still works.
10. Enter "l" (lower-case letter L) in the UART terminal. You should see a list of the bonded devices on the terminal window.
11. Disconnect from the PC CySmart app, connect again from the phone, and verify that the application still works. It should connect and pair without requiring the passkey.
12. Disconnect from the device, re-connect from the PC, Pair, and verify that the application still works. Again, a passkey should not be required to Pair with the device.
13. Disconnect from the device.
14. Clear the bonding information from the phone and CySmart on the PC.
15. Note: you may not be able to bond to multiple computers running CySmart, but you can connect to a PC and a phone or multiple phones.

9.5-6D OVERVIEW OF CHANGES

1. A #define for BOND_MAX which is the maximum number of devices that can be bonded at a time (default is set to 4).
2. NVSRAM is organized so that we have:
    a. 1 VSID for the local keys (i.e. privacy keys).
    b. 1 VSID to keep track of how many bonded devices we have and the next one to be over-written.
    c. VSIDs to hold host information (i.e. BD_ADDR and CCCD values). There is one VSID for each bonded device so this is BOND_MAX VSIDs.
    d. VSIDs to hold encryption keys for each bonded host. There is one VSID for each bonded device so this is BOND_MAX VSIDs.
3. Update UART receive callback function so that it just toggles whether we are in bonding mode or not when "e" is pressed. Upon entering bonding mode, if the max number of devices is already bonded, it will remove the oldest information from the bonded device list, address resolution database, and NVRAM (hostinfo and paired device keys).
4. Update UART receive callback function that prints bonding information when "l" is pressed.
5. Update BTM_PAIRING_COMPLETE_EVT so that it stores the newly bonded device's host information into the correct VSID slot in NVRAM. That is, it needs to store the information in the first free location. This case also increments the number of bonded devices and increments the next free slot location since a new device has just completed bonding.
6. Update BTM_ENCRYPTION_STATUS_EVT so that it searches for the BD_ADDR of the device that was just paired. If it is found, then the device was previously bonded, so its host information can be read from NVRAM.
7. Update BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT so that it stores the newly bonded device's encryption key information into the correct VSID slot in NVSRAM. That is, it needs to store the information in the first free location.
8. Update BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT so that it searches through all bonded devices to determine if the device that is currently trying to pair already has bonding information. If the information is found, it is loaded from NVRAM. If the information is not found, this case returns WICED_BT_ERROR which causes the stack to generate new keys and then call BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT.

9. Update the GATT set_value function so that it stores any changes to the CCCD value to the proper NVRAM location. That is, the value must be stored in the location that is assigned to the currently connected host.

9.6 EXERCISES – BLE LOW POWER, BEACONS & OTA (COPIED FROM WBT101-04C)

9.6-1 BLE LOW POWER (EPDS)

9.6-1A INTRODUCTION

In this exercise, you will program and analyze a project that implements low power using ePDS. The project is based on the exercise that stores BLE bonding information in NVRAM (ch04b/ex03_bond). This material is covered in **Error! Reference source not found.**

9.6-1B PROJECT CREATION

1. Use the template in folder "templates/CYW920819EVB/ch04c_ex01_lp" to create a project called **ch04c_ex01_lp**.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device and Bluetooth configurators, change the name of the device to <*inits*>_lp, then save and close the configurators.
4. Review the file app.c to familiarize yourself with the way sleep is configured.

9.6-1C TESTING

1. Program the project onto the kit.
    a. Hint: If your device is in low power mode you will have to put it into Recovery mode first to program it. To enter Recovery mode:
        i. Press and hold the recovery button on the base board (left button)
        ii. Press and release the reset button (center button)
        iii. Release the recovery button
2. Open a UART terminal window. This will allow you to see sleep events and to determine how often the device wakes up and goes back to sleep during different operations.
    a. Hint: You may see strange (non-ASCII) characters in the UART window when the device wakes from sleep. This is because the PUART was left active during ePDS. If the debug print was removed from the low power sleep callback function, you would not see these characters. In a real-life low power application, the PUART would not be left active but we want to see what's going on.
3. Open the PC CySmart app. Start scanning and then stop once your device appears.
4. Observe the UART while the device is in high duty cycle and low duty cycle advertising.
5. Connect to the device in CySmart. You will see a notification asking to confirm the connection parameters. Select 'Yes'.
6. Observe the UART once the connection is established.
7. Discover all attributes in the GATT database, and Pair with the device.
    a. Hint: attribute discovery and paring will take longer (~5 to 10 sec) since they each take multiple wakeup cycles to complete.
8. Observe the UART when pairing completes.
9. Enable all notifications.

10. Press and release the button repeatedly so that notifications are sent.
11. Observe the UART while notifications are being sent.
12. Disconnect and clear the Device Information from the Device List in CySmart.

9.6-1D QUESTIONS

1. Which lines in the code are used to configure and initialize sleep?

2. When in the code is sleep configured (i.e. after which event)?

3. What is used as a wakeup source?

4. What is the name of the sleep permit handler function?

5. When are the connection interval min, max, latency, and timeout values updated in the code and what values are used?

9.6-2 (ADVANCED) EDDYSTONE URL BEACON

9.6-2A INTRODUCTION

In this exercise, you will create an Eddystone beacon that will advertise the URL for https://www.cypress.com. From your phone you will be able to scan for the beacon (using a beacon scanner app) and then directly connect to the advertised website. This material is covered in **Error! Reference source not found.**.

9.6-2B PROJECT CREATION

1. Create a new application called **ch04c_ex02_eddy** using the modus.mk file in templates/CYW920819EVB/ch04c_ex02_eddy. This is a very simple application with no GATT support. All it does is advertise.
2. The beacon_lib library is already added as middleware and wiced_bt_beacon.h is included in app.c in the template.
3. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
4. In app_set_advertisement_data() create an advertising packet with three elements.
    a. BTM_BLE_ADVERT_TYPE_FLAG
        i. This is the same element you have used in chapter 4A.
    b. BTM_BLE_ADVERT_TYPE_16SRV_COMPLETE
        i. Two-byte Eddystone Service UUID (0xFEAA). Note that this is little-endian so the LSB must be the first element in the array.
    c. BTM_BLE_ADVERT_TYPE_SERVICE_DATA
        i. Eddystone Service UUID (again little-endian).
        ii. Eddystone frame type for URL (see EDDYSTONE_FRAME_TYPE_URL in wiced_bt_beacon.h).
        iii. Transmit power (use 0xF0)
        iv. URL Scheme Prefix (see EDDYSTONE_URL_SCHEME_1 in wiced_bt_beacon.h)
        v. Data (the URL itself as a list of characters)
            1. Hint: 'c', 'y', 'p', 'r', 'e', 's', 's'
        vi. Eddystone suffix for .com (see the Eddystone website).
5. In app_bt_cfg.c, change the value of wiced_bt_cfg_settings.ble_advert_cfg.high_duty_duration to 0, which indicates to the stack that advertising should never time out.
6. Build and program the kit.

9.6-2C TESTING

1. On your phone, install a beacon scanner app. For Android, there is an app called "Beacon Scanner" written by Nicolas Bridoux that works well (although it does not recognize EID frames). Similar apps exist for iOS.
2. Look for your Bluetooth Device address in the UART terminal window to find the correct beacon in the list.
    a. Note: In iOS the Bluetooth Device address can't be identified (Apple doesn't allow it) so you will not be able to identify your specific device.
3. Open the URL for www.cypress.com from the beacon app.
4. If you don't see your device in the beacon app it most likely means your packet isn't correct. Using the CySmart PC application, scan for your device and look at its advertising packet to determine what's wrong.

9.6-3 (ADVANCED) USE MULTI-ADVERTISING ON A BEACON

9.6-3A INTRODUCTION

In this exercise you will use multi-advertising to send UID, URL and TLM frames to the listening devices. This material is covered in **Error! Reference source not found.**.

9.6-3B PROJECT CREATION

1. Create a new application called **ch04b_ex03_multi** using the modus.mk file in templates/CYW920819EVB/ch04b_ex03_multi. This template already changes the advertising timeout, so you will not need to edit app_bt_cfg.c again.
2. The beacon_lib library is already added as middleware and wiced_bt_beacon.h is included in app.c in the template.
3. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
4. The code in app_set_advertisement_data() shows how to set up the URL advertising using multi-advertising. It uses the library function wiced_bt_eddystone_set_data_for_url() which is included in the beacon_lib middleware library.
5. Find the global that defines the advertising parameters and speed up the advertising rate by changing the value of adv_int_max from BTM_BLE_ADVERT_INTERVAL_MAX (0x4000) to 100.
6. Create a global array for the new packet (e.g. tlm_packet[]) which will look similar to the url_packet array.
7. Make a copy of the four lines of code that create the URL packet and edit them to create a TLM packet.
    a. Instead of wiced_bt_eddystone_set_data_for_url(), call the function for TLM (unencrypted).
    b. Hint: Use 0 for the vbatt, temp, adv_cnt and sec_cnt arguments.
    c. Hint: You can re-use the packet_len argument.
    d. Hint: Use the provided #define BEACON_EDDYSTONE_TLM to set up the second packet.
8. Repeat the above two steps for a UID packet. That is:
    a. Start by creating another packet array (e.g. uid_packet).
    b. Make another copy of the four lines of code and edit them to create a UID packet.
        i. Instead of wiced_bt_eddystone_set_data_for_url(), call the function for UID.

    Hint: Use 0 for the ranging data argument.

ii. Modify the bytes in the provided uid_namespace and uid_instance arrays so that you will be able to pick out your beacon in the app.

Hint: The namespace is 10 bytes and the instance is 6 bytes.

Hint: Use the provided #define BEACON_EDDYSTONE_UID to set up the third packet.

9. The template includes a timer that fires every 100ms. Use the provided callback function to increment the "seconds" parameter in the TLM advertising packet.

Note: the parameter is actually tenths of a second even though some scanner apps say seconds.

Hint: In the callback just reuse two lines of code from app_set_advertisement_data() to re-generate the packet and re-set the advertising data.

Hint: Provide the tenths variable value in when you generate the packet instead of a value of 0. Remember that the value needs to be sent little endian – you can use the macro SWAP_ENDIAN_32(tenths) to do the swapping for you.

9.6-3C TESTING

1. Build and program the application to your kit.
2. On your phone, open the beacon scanner app.
3. Look for your Bluetooth Device address in the UART terminal window to find the correct beacons in the list.
4. You should see two beacons with your address: one that shows URL and TLM information and the other that shows UID and TLM information. Notice how the TLM Uptime value increases every second.
5. It should look something like this:

9.6-4 (ADVANCED) ADVERTISE MANUFACTURING DATA AND USE SCAN RESPONSE FOR THE UUID

9.6-4A INTRODUCTION

In this project, you will take a project that advertises the manufacturer ID for Cypress and a product ID of 0x2A and you will add a scan response packet that sends the Service UUID for the Modus Service. This material is covered in 4C.3

9.6-4B PROJECT CREATION

1. Create a new application called **ch04c_ex04_scan** using the modus.mk file in templates/CYW920819EVB/ch04c_ex04_scan. This template is a solution to exercise ch04b/ex02_ntfy.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device and Bluetooth configurators, change the name of the device to <*inits*>_scan, then save and close the configurators.
4. Copy the app_set_advertisement_data function to a new function called app_set_scan_response_data.
    a. Hint: Don't forget to add function prototype at the top of the file.
5. Update the scan response packet to send the 128-bit service UUID.

a. Hint: there is a macro for the service UUID in cycfg_gatt_db.h that you can use in the array that you set up.
b. Hint: Remember, the Scan response packet doesn't have flags.

6. At the end of your new function, call the function to set the raw scan response data instead of the raw advertisement data.
7. Call your new function before starting advertising.

9.6-4C TESTING

1. Build and program the project to your kit.
2. Using the CySmart mobile app, scan for devices and note that it reports “1 Service Advertised”.
   a. Note: This feature is only on the iOS version of CySmart. It shows services advertised instead of the Bluetooth Device Address
3. Open the PC version of CySmart and scan for your device. Stop scanning once you see it.
4. Click on your device and examine the Advertisement data packet to verify it is as expected.

Click the tab above the Advertisement data that says Scan response data and verify it is as expected.

9.6-5 (ADVANCED) OTA FIRMWARE UPGRADE (NON-SECURE)

9.6-5A INTRODUCTION

In this exercise, you will modify a project that counts button presses to add OTA firmware upgrade capability. Once OTA support is added, you will modify the project to decrement the count instead of incrementing and you will upload the new firmware using OTA. This material is covered in **Error! Reference source not found.**

9.6-5B PROJECT CREATION

1. Create a new application from templates/CYW920819EVB/ch04c_ex05_ota.
   a. Hint: The template is just the solution project for the pairing exercise that counts button presses with OTA_FW_UPGRADE=1 added to the modus.mk file to get the build settings updated for OTA.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Middleware selector and enable "fw_upgrade_lib".
4. Use the Bluetooth Configurator to:
   a. Change the device name *to <inits>_ota*.
   b. Add the OTA Service and Characteristics as shown in **Error! Reference source not found.**.
5. Save and close the Configurators.
6. Edit cycfg_gatt_db.h:
   a. Add `#include "wiced_bt_ota_firmware_upgrade.h"`
   b. Replace the Service and Characteristic handle defines with the ones provided in the **Error! Reference source not found.** section of this document.
7. Edit app.c:
   a. Add `#include "wiced_bt_ota_firmware_upgrade.h"`
   b. Add `wiced_ota_fw_upgrade_init(NULL, NULL, NULL)` after the GATT database is initialized.

c. Add `wiced_ota_fw_upgrade_connection_status_event(p_conn)` in the GATT connection status event (for both connect and disconnect events).
d. Review the GATT Attribute Request Event handler code to understand how OTA GATT attribute request events are passed to the library.

9.6-5C TESTING

1. Build the project and program it to your kit.
2. Look at the PUART window during initialization and write down your kit's Bluetooth Device Address. You will need this to identify the correct device when you perform OTA upgrade.
3. Use CySmart to make sure the project functions as expected. Press the button and notice that the counter characteristic increases each time the button is pressed
    a. Hint: The Service with a single Characteristic is the Counter Service and the Service with two Characteristics is the OTA Service.
4. Disconnect from the kit in CySmart.
5. Unplug the kit from your computer. This will ensure that OTA is used instead of regular programming to update the firmware.
6. Update the project so that each button press decrements the value instead of incrementing it. Build the project without programming.
    a. Hint: In Quick Panel, use the link "Build ch04c_ex05_ota Application".
7. Connect your kit directly to a power outlet using a USB charger.
8. Use OTA to update your kit. You can use either the Windows or the Android app.
    a. Hint: The Windows application only works on Windows 10 or later since earlier versions of Windows do not support BLE.
    b. Hint: Don't forget to copy over the *.bin file from the Debug folder every time you re-build the project so that you are updating the latest firmware.
    c. Hint: If the OTA process fails on Windows, try resetting the kit and trying again. If that still fails, try using the Android version since it is more robust.
9. Once OTA upgrade is done, reset the kit then connect using CySmart and verify that the new firmware functionality is working.

9.6-6 (ADVANCED) OTA FIRMWARE UPGRADE (SECURE)

9.6-6A INTRODUCTION

In this exercise, you will update the previous OTA exercise to use Secure OTA firmware upgrade. This material is covered in **Error! Reference source not found.**

9.6-6B PROJECT CREATION

1. Create a new application from templates/CYW920819EVB/ch04c_ex06_ota_sec.
    a. Hint: The template is just the solution project for ch04c_ex05_ota but without the manual edits to the cycfh_gatt_db.h file.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Use the Bluetooth Configurator to:
    a. Change the name to <inits>_*otas.*
    b. Change the OTA Service UUID to the one for Secure OTA.
        i. Hint: The Secure OTA UUID is: C7261110-F425-447A-A1BD-9D7246768BD8
4. Save and close the configurators.
5. Re-do the edits to cycfg_gatt_db.h (add the include and replace the defines for the OTA Service and Characteristic handles).
6. Generate keys and update app.c as described in the **Error! Reference source not found.** section of this manual.

9.6-6C TESTING

1. Build the project and program it to your kit.
2. Use CySmart to make sure the project functions as expected.
3. Disconnect from the kit in CySmart.
4. Unplug the kit from your computer. This will ensure that OTA is used instead of regular programming to update the firmware.
5. Make the same change as the previous exercise to count down instead of up on each button press.
6. Build the project.
7. Connect your kit directly to a power outlet using a USB charger.
8. Use OTA to update your kit. You can use either the Windows or the Android app.
    a. Hint: Don't forget that every time you re-build the project you must sign the bin file and copy the resulting *.bin.signed to the Windows OTA folder or Android device. Remember, instructions on signing the file are in **Error! Reference source not found.**
    b. Hint: If the OTA process fails on Windows, try resetting the kit and trying again. If that still fails, try using the Android version since it is more robust.
9. Once OTA upgrade is done, connect to the kit using CySmart and verify that the new firmware functionality is working.
10. Try doing OTA with the unsigned image. Notice that it will fail after loading the image. The original firmware will be retained in this case.

9.7 EXERCISES – BLE CENTRALS (COPIED FROM WBT101-04D)

The exercises in this chapter require two kits - a peripheral and a central. You can use another CYW920819EVB-02 kit (Arduino style) or CYBT-213043-MESH kit (custom form factor) for the PERIPHERAL device. Always use an EVB for the central.

The peripheral application is a combination of the exercises from previous chapters. It advertises the device name and a service UUID. Once connected there is an LED characteristic that can be read and written without authentication. Note that on the MESH kit the LED is an RGB LED that takes a value from 0 (off) to 7 (white), while the EVB kit only has a single-color LED and any non-zero value turns it on. A counter characteristic, which is controlled by the button on the peripheral, requires pairing to be read and supports notifications.

9.7-1 MAKE AN OBSERVER

In this exercise you will build an observer that will listen to all the BLE devices that are broadcasting and will print out the BD Address of each device that it finds. To create this project, follow these steps:

1. Begin by programming your peripheral kit with the provided application. If you don't already have a 2nd kit to use as the peripheral, get one from an instructor.
    a. Make sure only the kit that you intend to use as a peripheral is plugged into your computer.
    b. Create a new application called **ch4d_ex00_peripheral** from the modus.mk files in either templates/CYW92019EVB/ch4d_ex00_peripheral_mesh or templates/CYW92019EVB/ch4d_ex00_peripheral_evb.
        i. Hint: If you use the Mesh application, you must select the CYBT-213043-MESH kit during creation.
    c. Use the Application Settings to set BT_DEVICE_ADDRESS to random.
    d. Use the Bluetooth Configurator to change the device name to <inits>_peri.
    e. Build and program the kit.
    g. Use the PC version of CySmart to test the application.
        i. Connect and open the GATT database browser.
        ii. Write a number between 0 and 7 to control the RGBLED.
            Note: If you are using a CYW920819EVB-02 for the peripheral, there is no RGB and numbers 1 through 7 just turn on the LED.
        iii. Check that you can't enable notifications (not authenticated).
        iv. Pair the device.
        v. Enable notifications and check that the button changes the Counter.
2. Unplug your Peripheral so that you don't accidentally re-program it with the Central project.
    a. Hint: You should not need to re-program this kit again and so, if possible, plug it into a charger instead of your computer.
3. Create a new application called **ch4d_ex01_observer** from the modus.mk file in templates/CYW92019EVB/ch4d_ex01_observer.
4. In app.c, make a function to process the scanned advertising packets which just prints out the BD Address of the remote. This function declaration should look like this:
   ```
   void myScanCallback( wiced_bt_ble_scan_results_t *p_scan_result, uint8_t *p_adv_data );
   ```
   Hint: Remember you can use the format code %B in WICED_BT_TRACE to print a BD Address.
5. Add a call to wiced_bt_ble_scan in the BTM_ENABLED event with a function pointer to your advertiser processing function. Enable filtering so that each new device only shows up once – otherwise you will see a LOT of packets from everyone's Peripheral.

6. Build and Program to your kit.
7. Open a terminal window.
8. If it is not already powered from a charger, plug in the kit programmed with your Peripheral application.
9. Open CySmart on your PC or Android Phone to find the BD Address of your Peripheral. You can also look at the UART terminal of the peripheral to find its address.
10. Look for the same BD Address in the list of devices printed by your Central.

9.7-2 READ THE DEVICE NAME TO SHOW ONLY YOUR PERIPHERAL DEVICE'S INFO

In this exercise you will extend the previous exercise so that it examines the advertising packet to find the device name. It will only list your device instead of all devices. It will also look for the Modus Service UUID.

1. If necessary, unplug your Peripheral so that you don't accidentally re-program it.
2. Create a new application called **ch4d_ex02_observer_mydev** from the modus.mk file in templates/CYW92019EVB/ch4d_ex02_observer_mydev or from the modus.mk file in your previous exercise.
3. Change the scanner callback function so that it looks at the advertising packet. Find and print out the BD Address only of devices that match your Peripheral's Device Name.
    a. Hint: You can use the function wiced_bt_ble_check_advertising_data to look at the advertising packet and find fields of type wiced_bt_ble_advert_type_t. If there is a field that matches it will return a pointer to those bytes and a length.
    b. Hint: You can use memcmp to see if the fields match what you are looking for.
4. For the advertising packet from your peripheral, search for a Service UUID and print it's value to the terminal.
    a. Hint: Remember the function WICED_BT_TRACE_ARRAY can be used to print out the value of an array much easier than using WICED_BT_TRACE.
5. Program your project to your Central.
6. Plug in your Peripheral and check to see if it is found. It should print:
    a. Device Name of your Peripheral
    b. BD Address
    c. UUID for the Modus Service – check that this matches the value of __UUID_SERVICE_MODUS in cycfg_gatt_db.h in the ch4d_ex00_peripheral project.
7. Use CySmart on your PC or your phone to verify that the BD Address matches.

9.7-3 UPDATE TO CONNECT TO YOUR PERIPHERAL DEVICE & TURN ON/OFF THE LED

In this exercise, you will modify the previous exercise to connect to your Peripheral once it finds it. We will decide what to connect to based on your Peripheral's Device Name. In real-world applications, it would be more common to search for a Service UUID or Manufacturer Data but during class you will need to use the Device Name since there will be a lot of devices with the same Service and Manufacturer Data.

Once connected, you will be able to send numbers to the LED Characteristic to turn the LED off/on.

To simplify the project, you will hardcode the handle, but in the upcoming exercises you will add Service discovery.

The steps are as follows:

1. Create a new application called **ch4d_ex03_connect** from the modus.mk file in templates/CYW92019EVB/ch4d_ex03_connect.

2. The template includes a keyboard interface called uart_rx_callback() that reads in and handles single key commands. Uncomment the code in the BTM_ENABLED_EVT to install the PUART receive handler. The callback function handles these key presses:
    a. ?: print out help
    b. s: turn on scanning
    c. S: turn off scanning
3. Update the UART Rx callback function to add the appropriate wiced_bt_ble_scan commands to start/stop scanning.
4. Remove the call to wiced_bt_ble_scan from the BTM_Enabled event.
5. Test 1: Unplug your Peripheral (if needed), program your Central, and plug your Peripheral back in.
6. Verify that the keyboard interface works and that s/S turns scanning on and off.
7. In the app_bt_cfg.c file, find the GATT configuration entry for client_max_links. Change the value from 0 to 1. If you don't change this setting, the connection will not be allowed.
8. Update your scan callback function to connect to the Peripheral when it finds one that it recognizes by calling wiced_bt_gatt_le_connect.
    a. Note: As mentioned previously, you will need to use the Device Name of the Peripheral here so that it will connect to your device. If you used the UUID or Manufacturer Data, it would connect to the first matching device it found which would almost certainly belong to another student.
9. After making the connection, turn off scanning.
10. The template also includes a simple GATT callback function. Register this callback function in the BTM_ENABLED event with function wiced_bt_gatt_register.
11. Add a global variable uint16_t to save the connection id.
12. In the GATT callback when you get a GATT_CONNECTION_STATUS_EVT figure out if it is a connect or a disconnect. Save the connection id to your global variable on a connect and set it to 0 on a disconnect. Print out a message when a connect or disconnect happens.
13. Add a ‘d’ command to the UART interface to call the disconnect function.
    a. Hint: remember that the disconnect function does not have "le" in its name.
14. In the management callback add a case for BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT. Because we are not pairing set the status variable to return WICED_BT_ERROR.
15. Test 2: Unplug your Peripheral (if needed), program your Central, and plug your Peripheral back in.
16. Test the following:
    a. Start Scanning (s)
    b. Verify that it finds and then connects to your Peripheral
    c. Disconnect from your Peripheral (d)
17. Create a new global uint16_t variable called ledHandle. HARDCODE its initial value to the handle of your LED Characteristic's Value. You won’t change the variable in this exercise, but in a future one you will find the handle via a Service Discovery.
    a. Hint: Make sure you use the Handle for the Characteristic's Value, not the Characteristic declaration.
18. Create a new function to write the LED Characteristic Value with a uint8_t argument that represents the state of the LED (on CYW920819EVB-02 this is either 0 or 1, while on CYBT-

213043-MESH it is a value between 0 and 7). If there is no connection or the ledHandle is 0 then you should just return. Then call the GATT write function.

   a. Hint: Don't forget to include wiced_memory.h to get access to the get and free buffer functions.

19. In the UART interface call write LED function when values from 0 – 7 are entered.
20. Test 3: Unplug your Peripheral (if needed), program your Central, and plug your Peripheral back in.
21. Test the following:
    a. Start Scanning (s)
    b. Verify that it finds and then connects to your Peripheral
    c. Change the LED color by entering values from 0 to 7.
    d. Disconnect from your Peripheral (d)

9.7-4 (ADVANCED) ADD COMMANDS TO TURN THE CCCD ON/OFF

In this project, you will set up the CCCD to turn on/off Notifications for the button Characteristic and print out messages when Notifications are received.

1. Create a new application called **ch4d_ex04_connect_notify** from the modus.mk file in templates/CYW92019EVB/ch4d_ex04_connect_notify or from the modus.mk file in your previous exercise.
2. After the connection is made using wiced_bt_gatt_le_connect, initiate pairing by calling wiced_bt_dev_sec_bond. This is necessary because the CCCD permissions for your Peripheral are set such that it cannot be read or written unless the devices are paired first (i.e. Authenticated).
    a. Hint: Do this in the GATT callback for the connected event, not right after calling wiced_bt_gatt_le_connect since you need to wait until the connection is up.
3. Add a new uint16_t global variable called cccdHandle to hold the handle of the CCCD. Set its initial value to HDLD_MODUS_COUNTER_CLIENT_CHAR_CONFIG from your peripheral application (this is hardcoded for now, but we will fix that in the next exercise).
4. Create a new function to write the CDDD for the Button Characteristic with a uint8_t argument to turn Notifications off or on. If there is no connection or the cccdHandle is 0 then you should return.
    a. Hint: You can use the function wiced_bt_util_set_gatt_client_config_descriptor.
    b. Hint: add the gatt_utils_lib library to your project using the Middleware selector.
5. Add cases ‘n’ and ‘N’ to set and unset the CCCD using the function you just created.
6. Add a case for GATT_OPERATION_CPLT_EVT into the GATT callback. This case should print out the connection id, operation, status, handle, length and the raw bytes of data. This is how you will see the notification values sent back from the peripheral once notifications are enabled.
    a. Hint: These values are all under the operation_complete structure in the event data. Some of them are 2 levels down under operation_complete.response_data or 3 levels down under operation_complete.response_data.att_value.
    b. Hint: Use the function WICED_BT_TRACE_ARRAY to print out the data array.
7. Unplug your Peripheral (if needed), program your Central, and plug your Peripheral back in.
8. Test the following:
    a. Start Scanning (s)
    b. Verify that the connection is made
    c. Press the button on the peripheral. Notifications are not enabled so you should not see a response.
    d. Enable Notifications (n)
    e. Press the button on the peripheral. You should see a response.
    f. Disable Notifications (N)
    g. Press the button. You should not see a response.
    h. Disconnect (d)

9.7-5 (ADVANCED) MAKE YOUR CENTRAL DO SERVICE DISCOVERY

In this exercise, instead of hardcoding the Handle for the LED and Button CCCD, we will modify our program to do a Service discovery. Instead of triggering the whole process with a state machine, we will use keyboard commands to launch the three stages.

The three stages are:

1. 'q'= Service discovery with the UUID of the Modus Service to get the start and end Handles for the Service Group.
2. 'w'= Characteristic discovery with the range of Handles from step 1 to discover all Characteristic Handles and Characteristic Value Handles.
3. 'e' = Descriptor discovery of the button Characteristic to find the CCCD Handle.

To create the project:

1. Create a new application called **ch4d_ex05_discover** from the modus.mk file in templates/CYW92019EVB/ch4d_ex05_discover or from the modus.mk file in your previous exercise.
    a. Note: The gatt_util_lib is included automatically for you.
2. Open the file cycfg_gatt_db.h from the peripheral application and copy over the macros for the following into the app.c file for this exercise:
    a. __UUID_SERVICE_MODUS
    b. __CHARACTERISTIC_MODUS_RGBLED
    c. __CHARACTERISTIC_MODUS_COUNTER
    d. __UUID_DESCRIPTOR_CLIENT_CHARACTERISTIC_CONFIGURATION

    Hint: This guarantees that the Central uses the same UUIDs that are used in the Peripheral.
3. Create variables to save the start and end Handle of the Modus Service and create a variable to hold the Modus Service UUID (if you don't already have one).

```
static const uint8_t modusServiceUUID[] = { __UUID_SERVICE_MODUS };
static uint16_t serviceStartHandle = 0x0001;
static uint16_t serviceEndHandle   = 0xFFFF;
```

4. Make a new structure to manage the discovered handles of the Characteristics:

```
typedef struct {
    uint16_t startHandle;
    uint16_t endHandle;
    uint16_t valHandle;
    uint16_t cccdHandle;
} charHandle_t;
```

5. Create a charHandle_t for the LED and the Counter as well as the Characteristic UUIDs to search for.

```
static const uint8_t ledUUID[] = { __CHARACTERISTIC_MODUS_LED };
static charHandle_t  ledChar;

static const uint8_t counterUUID[] = { __CHARACTERISTIC_MODUS_COUNTER };
static charHandle_t  counterChar;
```

6. Create an array of charHandle_t to temporarily hold the Characteristic Handles. When you discover the Characteristics, you won't know what order they will occur in, so you need to save the Handles temporarily to calculate the end of group Handles.

```
#define MAX_CHARS_DISCOVERED (10)
static charHandle_t charHandles[MAX_CHARS_DISCOVERED];
static uint32_t charHandleCount;
```

Service Discovery

7. Add a function to launch the Service discovery called “startServiceDiscovery”. This function will be called when the user presses ‘q’. Instead of finding all the UUIDs you will turn on the filter for just the Modus Service UUID.
    a. Setup the wiced_bt_gatt_discovery_param_t with the starting and ending Handles set to 0x0001 & 0xFFFF.
    b. Setup the UUID to be the UUID of the Modus Service.
    c. Use memcpy to copy the Service UUID into the wiced_bt_gatt_discovery_param_t.
    d. Set the discovery type to GATT_DISCOVER_SERVICES_BY_UUID and launch the wiced_bt_gatt_send_discover.
8. Add the case GATT_DISCOVERY_RESULT_EVT to your GATT Event Handler. If the discovery type is GATT_DISCOVER_SERVICES_BY_UUID then update the serviceStart and serviceEnd Handle with the actual start and end Handles (remember GATT_DISCOVERY_RESULT_SERVICE_START_HANDLE and GATT_DISCOVERY_RESULT_SERVICE_END_HANDLE.

Characterisic Discovery

9. Add a function to launch the Characteristic discovery called “startCharacteristicDiscovery” when the user presses ‘w’.
    a. Setup the wiced_bt_gatt_discovery_param_t start and end Handle to be the range you discovered in the previous step.
    b. Call wiced_bt_gatt_send_discover with the discovery type set to GATT_DISCOVER_CHARACTERISTICS.
10. In the GATT_DISCOVERY_RESULT_EVT of your GATT Event Handler add an "if" for the Characteristic result. In the “if” you need to save the startHandle and valueHandle in your charHandles array. Set the endHandle to the end of the Service Group (assume that this Characteristic is the last one in the Service). If this is not the first Characteristic, then set the previous Characteristic end handle:

```
if( charHandleCount != 0 )
{
   charHandles[charHandleCount-1].endHandle =
charHandles[charHandleCount].endHandle-1;
}
charHandleCount += 1;
```

The point is to assume that the Characteristic ends at the end of the Service Group. But, if you find another Characteristic, then you know that the end of the previous Characteristic is the start of the new one minus 1.
Then you want to see if the Characteristic is the Counter or LED Characteristic. If it is one of those, then also save the start, end and value Handles.

Descriptor Discovery

11. Add a function to launch the Descriptor discovery called “startDescriptorDiscovery” when the user presses ‘e’. The purpose of this function is to find the CCCD Handle for the counter Characteristic.
    a. It will need to search for all the Descriptors in the Characteristic Group.
    b. The start will be the Counter Value Handle + 1 to the end of the group handle.
    c. Once the parameters are setup, launch wiced_bt_gatt_send_discover with GATT_DISCOVER_CHARACTERISTIC_DESCRIPTORS.

12. In the GATT_DISCOVERY_RESULT_EVT of your GATT Event Handler add an "if" for the Descriptor result. If the Descriptor UUID is _UUID_DESCRIPTOR_CLIENT_CHARACTERISTIC_CONFIGURATION then save the Button CCCD Handle.

13. The last change you need to make in the GATT callback is for the GATT_DISCOVERY_CPLT_EVT. You should check to see if it is a Characteristic discovery. If it is, then you will be able figure out the endHandle for each of the LED and Button Characteristic by copying them from the charHandles array. Your code could look something like this:

```
// Once all characteristics are discovered... you need to setup the end handles
if( p_event_data->discovery_complete.disc_type == GATT_DISCOVER_CHARACTERISTICS
)
{
  for( int i=0; i<charHandleCount; i++ )
  {
    if( charHandles[i].startHandle == ledChar.startHandle )
      ledChar.endHandle = charHandles[i].endHandle;

    if( charHandles[i].startHandle == counterChar.startHandle )
       counterChar.endHandle = charHandles[i].endHandle;
  }
}
```

14. Add the function calls for ‘q’, ’w’, and ‘e’. Also add those keys to the help print out.
15. Unplug your Peripheral (if needed), program your Central, and plug your Peripheral back in.
16. Test the following:
    a. Start Scanning (s)
    b. Discover the Modus Service (q)
    c. Discover the Counter and LED Characteristics (w)
    d. Discover the Button CCCD (e)
    e. Turn on the LED (1...7)
    f. Turn off the LED (0)
    g. Turn on notification (n)
    h. Press the button on the Peripheral and make sure it works
    i. Turn off notification (N)
    j. Press the button on the Peripheral to make sure the notification is off

9.7-6 (ADVANCED) RUN THE ADVERTISING SCANNER

In this exercise you will experiment with a full-function advertising scanner project.

1. Create a new application called **ch4d_ex06_AdvScanner** from the modus.mk file in templates/CYW92019EVB/ch4d_ex06_AdvScanner.
2. Open a UART terminal window.
3. Unplug your Peripheral (if needed), program your Central, and plug your Peripheral back in.
4. Once the project starts running press "?" to get a list of available commands.
5. Use 's' and 'S' to enable/disable scanning
6. Use 't' to print a single line table of all devices that have been found
    a. This table will show raw advertising data
7. Use 'm' to print a multi-line table of all devices
    a. This table will show both raw advertising data and decoded data
    b. Use '?' to get a list of table commands
    c. Use '<' and '>' to change pages
    d. Use 'ESC' to exit the table
8. To filter on a specific device, enter its number from the table
9. Use 'r' to list recent packets from the filtered device
    a. Use '?' to get a list of table commands
    b. Use '<' and '>' to change pages
    c. Use 'ESC' to exit the table

9.8 EXERCISES – USING THE DEBUGGER (COPIED FROM WBT101-05)

9.8-1 RUN BTSPY

In this project you will use BTSpy to look at Bluetooth protocol trace messages.

9.8-1A PROJECT CREATION

1. Create a new application called **ch09_ex01_btspy** using the modus.mk file in templates/CYW920819EVB/ch09_ex01_btspy.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device Configurator and then the Bluetooth Configurator.
    a. Change the device name to <inits>_btspy.
    b. Save edits and close the configurators.
4. Review the steps in the **Error! Reference source not found.** section of this chapter and add in the necessary code to support WICED HCI.
    a. Hint: Add includes for the two header files, create a variable for transport_pool, create/setup the transport_cfg structure, and in application_start, initialize transport and create buffer pools.
    b. Hint: You can specify NULL for the receive data handler since we won't deal with any incoming HCI commands in our application.
5. Create a BTSpy callback function and register it in the BTM_ENABLED_EVT event.
6. Route the debug messages to WICED_ROUTE_DEBUG_TO_WICED_UART instead of WICED_ROUTE_DEBUG_TO_PUART. This will mean that Bluetooth trace messages and WICED_BT_TRACE messages will go to the HCI UART using WICED HCI formatting.

7. In the app_bt_cfg.c file, change the low duty cycle advertisement duration to 0 so that advertising doesn't stop after 60 seconds.

9.8-1B PROGRAMMING AND SETUP

1. Build and download the application to the kit.
2. Press **Reset** (SW2) on the WICED evaluation board to reset the HCI UART after downloading and before opening the port with *ClientControl*. This is necessary because the programmer and HCI UART use the same interface.
3. Run the *ClientControl* utility from the Quick Panel. Select the HCI UART port, and set the baud rate to the baud rate used by the application for the HCI UART (the default baud rate is 3000000). Do NOT open the port yet.
4. Run the BTSpy utility from the Quick Panel.
    a. Hint: If your computer has a Broadcom WiFi chip, it will show lots of messages that are from your computer instead of from your kit. To disable these messages, you will need to temporarily disable Bluetooth on your computer.
5. In the ClientControl utility, open the HCI UART COM port by clicking on "Open Port".

9.8-1C TESTING

1. If you want to capture the log from BTSpy to a file, click on the "Save" button (it looks like a floppy disk). Click Browse to specify a path and file name, click on "Start Logging", and then click on "OK".
    a. Hint: If you want to include the existing log window history in the file (for example if you forgot to start logging at the beginning) check the box "Prepend trace window contents" before you start logging.
2. Use CySmart to connect to the device and observe the messages in the BTSpy window.
3. Once paring is complete, click the "Notes" button (it looks like a Post-It note), enter "Pairing Completed" and click OK. Observe that a note is added to the trace window.
4. Read the Button Characteristic and observe the messages.
5. Add a note that says " Button Characteristic Read Complete"
6. Disconnect from the device.
7. Once you are done logging, click on the "Save" button, click on "Stop Logging", and then click on "OK".

9.8-2 USE THE CLIENT CONTROL UTILITY

9.8-2A INTRODUCTION

In this project, you will add the ability to start and stop advertising using WICED HCI messages from the Client Control utility.

9.8-2B PROJECT CREATION

1. Create a new application called **ch09_ex02_hci** using the modus.mk file in templates/CYW920819EVB/ch09_ex02_hci.
2. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS = random.
3. Open the Device Configurator and then the Bluetooth Configurator.
    a. Change the device name to <inits>_hci.
    b. Save edits and close the configurators.
4. Review the steps in the **Error! Reference source not found.** section of this chapter and add in the necessary code to support WICED HCI.
5. Create an RX handler and add a case to the opcode switch statement to handle the *_ADVERTISE command.
    a. Hint: Refer to the file include/common/hci_control_api.h to find the full name of the opcode for the *_ADVERTISE command.
    b. Hint: The payload will be 0 to stop advertisements and 1 to start advertisements.
    c. Hint: Use the function wiced_bt_start_advertisements with either BTM_BLE_ADVERT_OFF or BTM_BLE_ADVERT_NONCONN_HIGH depending on whether the first byte of the payload is 0 or 1.
6. (Advanced): Update the Bluetooth Stack Management callback to send a WICED HCI status message when the advertising state changes.
    a. Hint: Use wiced_transport_send_data with the same ADVERTISE code used above. Note that you must send a value of either 0 (stopped) or 1 (started) even though the callback will send other values depending on the type of advertising (e.g. 3 for high duty connectable). If you send a value other than 0 or 1, the Client Control utility will crash.

9.8-2C TESTING

1. Program the project to the kit.
    a. Hint: If you have the port open in Client Control, you will have to close the port before being able to program again because programming and Client Control both use the WICED HCI port.
2. Open the Client Control program from the Quick Panel and connect to the WICED HCI port.
3. Open CySmart and scan for devices. Note that your device does not appear.
4. In Client Control, switch to the GATT tab and click on Start Adverts. Look at the return message if you implemented the status message.
5. Verify that your device now appears in CySmart.
6. In Client Control click on Stop Adverts and look at the return message. Stop and Re-start the scan in CySmart. Note that your device no longer appears.

9.8-3 (ADVANCED) RUN THE DEBUGGER

In this exercise you will setup and run the debugger using a MiniProg4, Olimex, or J-Link debug probe. You will then use the debugger to change the value of a variable that controls an LED on the shield.

1. Read the "Hardware Debugging for CYW207xx and CYW208xx" guide.
    a. Note: To access the hardware debugging guide follow one of the following paths
    Quick Panel > Documents Tab > ModusToolbox Documentation Index
    Help > ModusToolbox General Documentation > ModusToolbox Documentation Index
2. **MiniProg4 or Olimex**: In the file: <install_folder>/ModusToolbox_1.1/libraries/bt_sdk1.1/makefiles/platforms/20819/mainapp.mk. move the “#” comment character from the line with “bluetooth.openocd.launch.xml” to the line with “bluetooth.jlink.launch.xml” to switch from J-Link to OpenOCD launch configuration.
    a. Note: This step is described in section 5.2 of the Hardware Debugging Guide.
3. **Olimex or J-Link**: Follow the instructions in sections 2 and 3 of the Hardware Debugging Guide to install drivers for the debug probe.
4. Create a new application called **ch09_ex03_debug** using the modus.mk file in templates/CYW920819EVB/ch09_ex03_debug.
5. Open the "Change Application Settings" dialog and verify that ENABLE_DEBUG is set to 1. Click "OK" to close the dialog.
6. Open the configurator and:
    a. Enable SWD
    b. Assign the SWD pins as follows:
        i. SWD_CLK: P02
        ii. SWD_IO: P03
    c. Verify that the pins are configured with a Type of "Peripheral".
        i. Hint: You can use the "chain links" icon to jump to the pin configuration.
    d. Save changes and close the configurator.
7. Add two pins (PLATFORM_GPIO_12 AND PLATFORM_GPIO_13) to the wiced_platform_pins list in the wiced_platform.h file.
    a. Hint: This is necessary because we added 2 new pins to our platform, but the platform configuration file does not define that many GPIOs by default.
8. Verify that the two switches in SW9 are in the OFF position on your kit.
    a. Hint: SW9 is the 2 position DIP switch near the 10-pin debug header. The 2 switches should be pushed to the side next to the debug header.
9. **MiniProg4** : In the file <install_folder>/ModusToolbox_1.1/libraries/bt_sdk-1.1/components/BT-SDK/208XX-A1_Bluetooth/platforms/CYW20819A1_openocd.cfg file put a “#” character in front of lines 6 and 7 and on line 8 add “source [find interface/kitprog3.cfg]” as shown here:

```
…
# default adapter
#source [find interface/ftdi/olimex-arm-usb-tiny-h.cfg]
#source [find interface/ftdi/olimex-arm-jtag-swd.cfg]
source [find interface/kitprog3.cfg]

set CHIPNAME CYW20819A1
…
```

10. If you have not already, connect your kit and debugger to each other and then to USB ports on your computer.

11. Program the application to the kit and launch the debugger (<appname> Build + Program + Debug).
    a. Note: You can also program using (<appname> Build + Program) and then start the debugger using the menu item "Run -> Debug Configurations -> <appname> Debug Attach".
12. Place a break point at the wiced_rtos_delay_milliseconds call in the app_task thread in app.c.
    a. Hint: To start the debugger click “Debug Attach” in the Quick Panel Launch Menu
13. Pause execution. You should be in the BUSY_WAIT_TILL_MANUAL_CONTINUE_IF_DEBUG_ENABLED() loop.
14. Set the value of *spar_debug_contine* to 1 to get out of the initial loop.
15. Execute a few more times and notice that execution suspends at the breakpoint you added.
16. Observe that the LED does not turn on because the variable "led" never changes.
17. Change the value of the variable "led" and then re-start execution.
    a. Hint: The LED is active low.
18. Note that the LED now turns on.
19. Stop the debugger by pressing the stop button.
20. Use the menu item Window > Perspective > Reset Perspective.

9.9 EXERCISES – BT CLASSIC BASICS (COPIED FROM WBT101-06A)

9.9-1 CREATE A SERIAL PORT PROFILE PROJECT

9.9-1A PROJECT CREATION

For this example, you will need to:

1. Use the SPP code example from the GitHub example repository CYW920819EVB02/apps/snip/bt/spp, to create a project called **ch09a_ex01_spp**.
2. Comment out lines 90 and 91 in spp.c to disable sending data on interrupts and on timer timeouts. For this exercise, we will only investigate using RX.
3. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS to random.
4. Change the name of your device by editing the variable BT_LOCAL_NAME in the wiced_bt_cfg.c file.
    a. Hint: The default name is “spp test”, remember to use your initials in the device name so that you can find it in the list of devices that will be advertising.
5. Review the file spp.c to familiarize yourself with the way that everything is configured.

9.9-1B TESTING

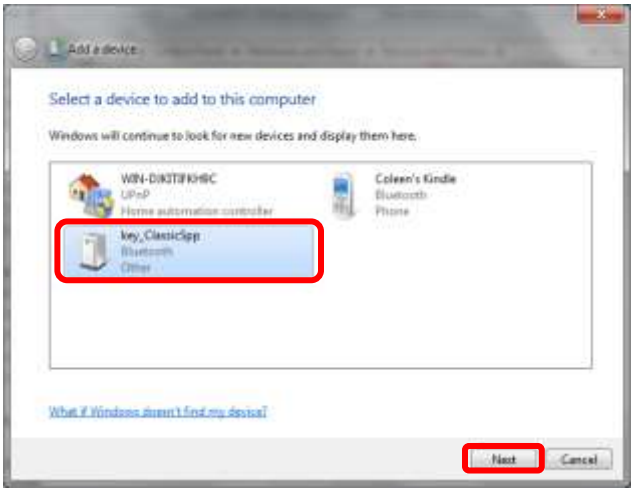
Once your application has been programmed to the kit, you can attach to it using Windows 7, Windows 10, MacOS or Android. Instructions for each are provided below.

Note that iOS does not support SPP directly, so you can't use an iPhone to test this project. Apple supports Classic Bluetooth with iAP2 (iPod Accessory Protocol) which works a bit differently than SPP and requires an MFi license.

9.9-1C PC INSTRUCTIONS (WINDOWS 7)

The first step is to pair your PC with the WICED Bluetooth device. Go to Control Panel -> Hardware and Sound -> Devices and Printers -> Add a Device.

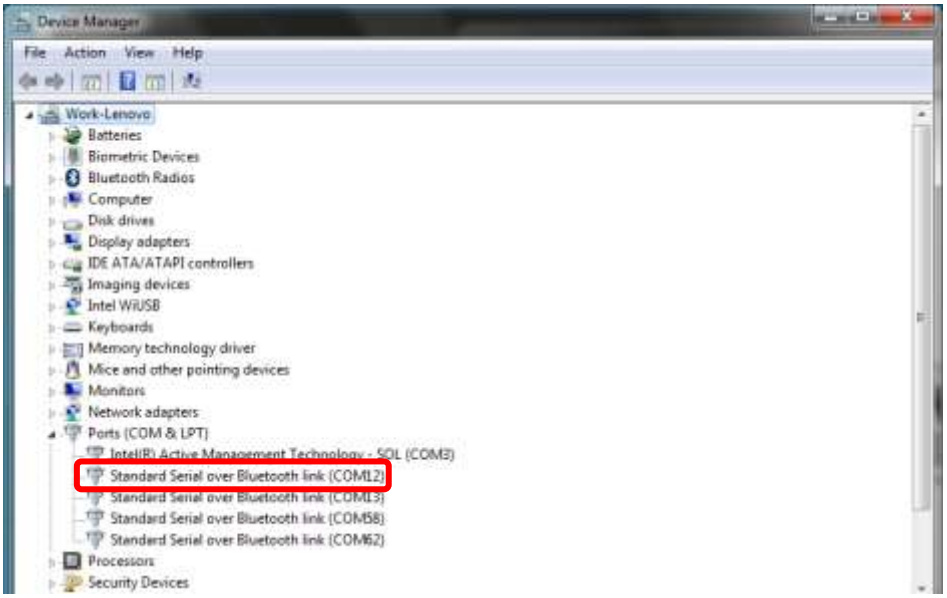
Wait until your device shows up in the list. Select it and click "Next".



Compare the 6-digit code with the one displayed on your UART terminal. If the two numbers match, make sure "Yes" is selected and click "Next".

Click "Close" once the device has been added. Your device will now show up in the list of devices and drivers will automatically install.

Go to the Device Manager and look under Ports (COM & LPT) to find the COM port for the SPP interface of your Bluetooth device. It will be listed as "Standard Serial over Bluetooth link". If you see multiple ports listed for Standard Serial over Bluetooth link, the lowest numbered port is the one you want to use.



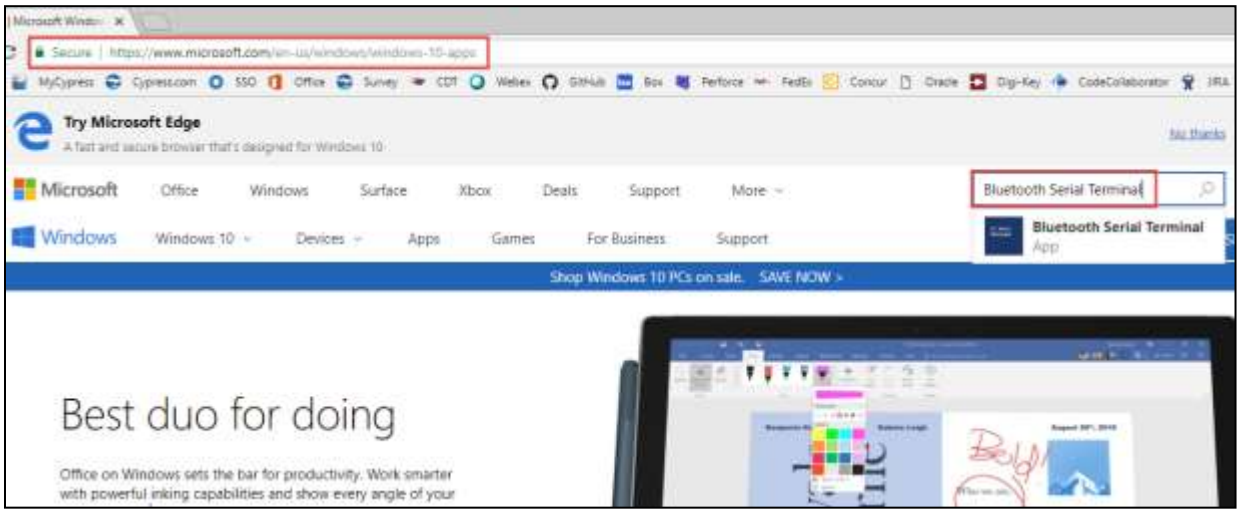
Open a serial terminal program of your choice (such as Putty) and connect to the SPP COM port. Now you can type in characters in the terminal window for the Bluetooth device and you will see them being received in the WICED kit by watching in terminal window connected to the kit's PUART.

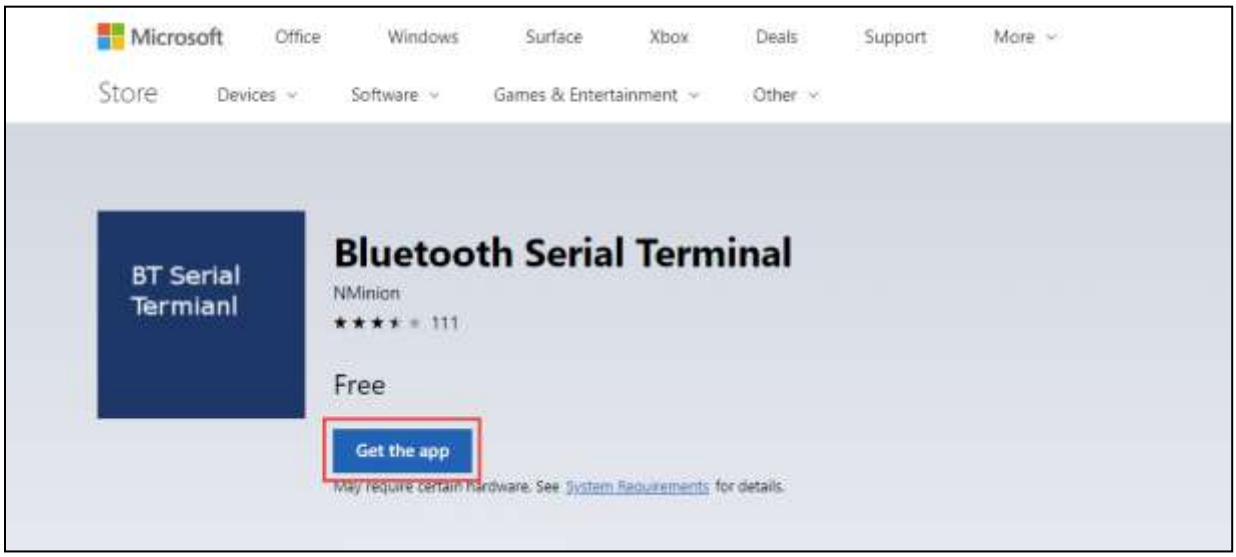
When you are done testing, close the Bluetooth SPP terminal window and then go to Control Panel -> Hardware and Sound -> Devices and Printers. Right click on the WICED Bluetooth device, select "Remove Device" and click "Yes" to remove the device's paring information.

9.9-1D PC INSTRUCTIONS (WIDOWS 10)

For Windows 10 you can use the same procedure as for Windows 7. Alternately, in Windows 10 you have the option to install the "Bluetooth Serial Terminal" from the Microsoft App Store which provides a "slick" interface. That option is discussed here.

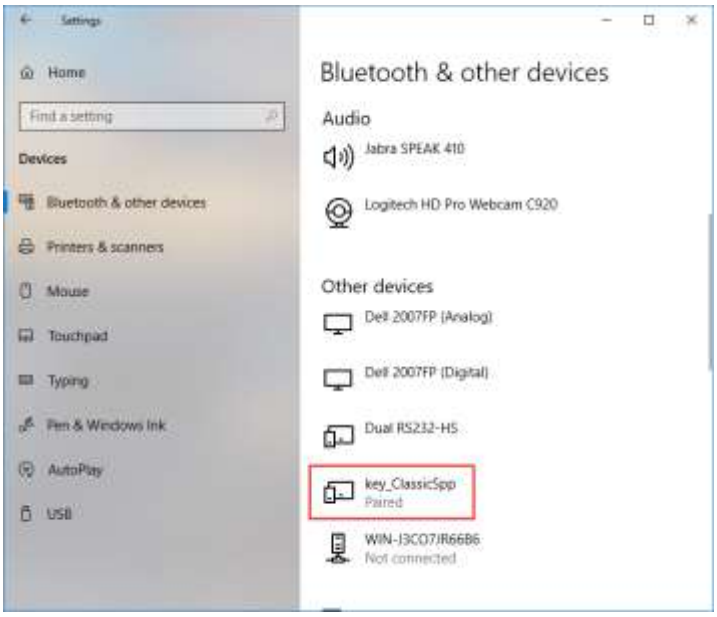
First, go to the Windows 10 Apps store (https://www.microsoft.com/en-us/windows/windows-10-apps), search for "Bluetooth Serial Terminal", and install it.



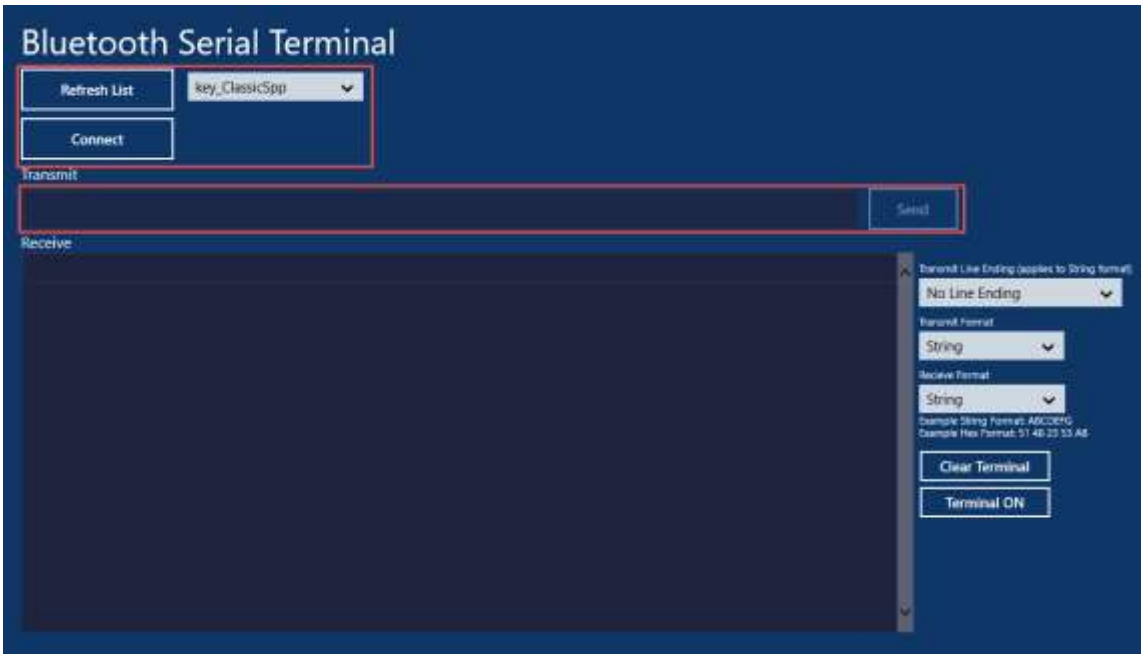


As with Windows 7, you need to pair with your device before it will show up as a serial port. To do this, go to *Settings -> Devices -> Add Bluetooth or other device -> Bluetooth*. When you see your device in the list, click on it. Compare the 6-digit code with the one displayed on your UART terminal. If the two

numbers match, click on "Connect". Click "Done". Your device should now show up in the list of devices as "Paired":

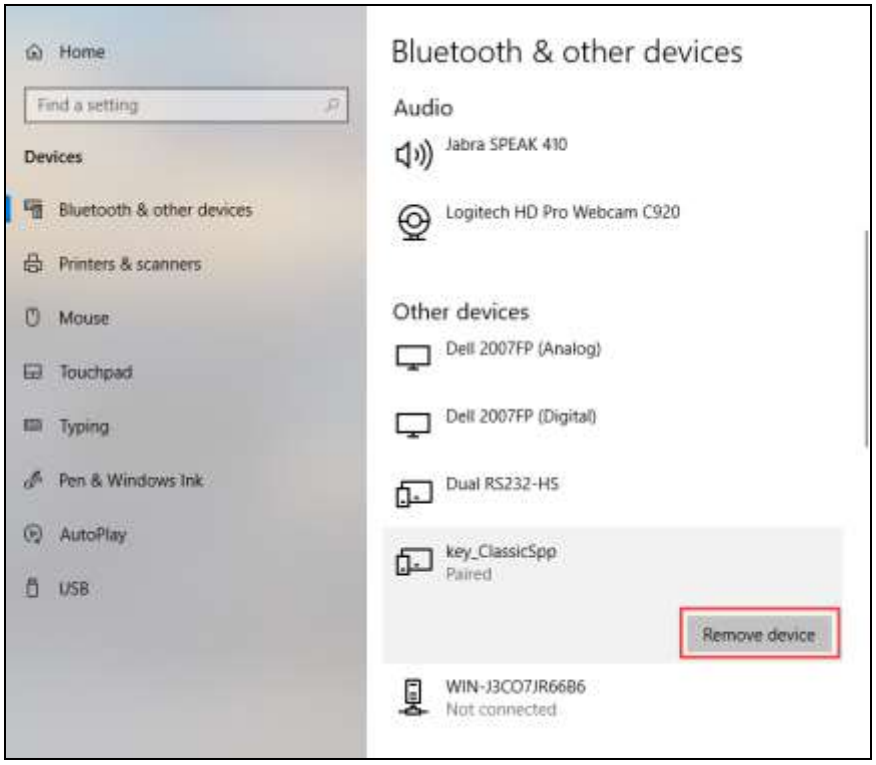


Now open the Bluetooth Serial Terminal app that you installed earlier. Your device should show up in the list. If not click "Refresh List".



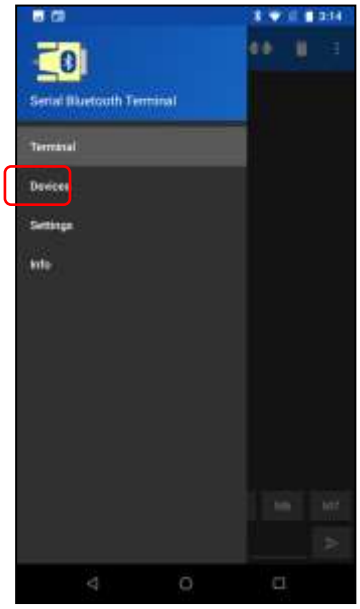
Once you see it in the list, click "Connect". Now you can type strings in the Transmit window and click "Send" to send them to the WICED SPP Project. Observe the strings being received in the WICED kit by watching in the UART terminal.

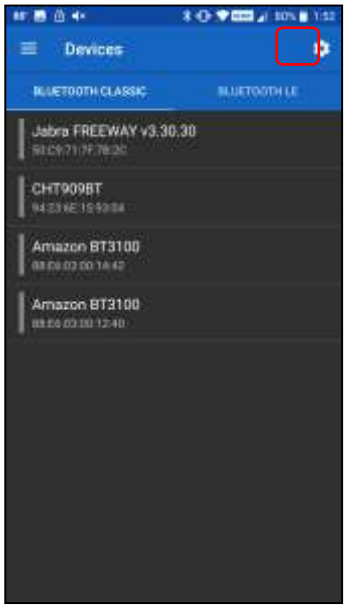
When you are done testing, click "Disconnect", close the Bluetooth Serial Terminal app and then go into the computer's Bluetooth settings to remove the device's paring information (Settings -> Devices -> <appname>_ClassicSpp -> Remove device -> Yes).



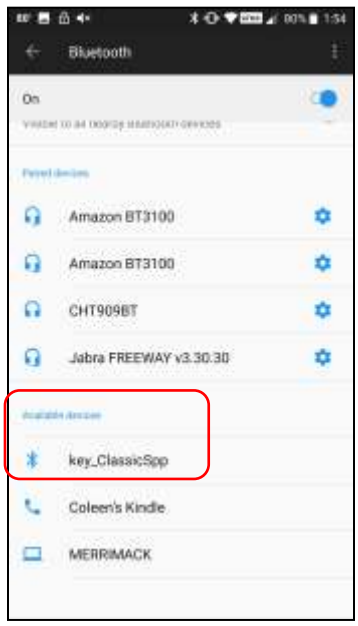
9.9-1E ANDROID INSTRUCTIONS

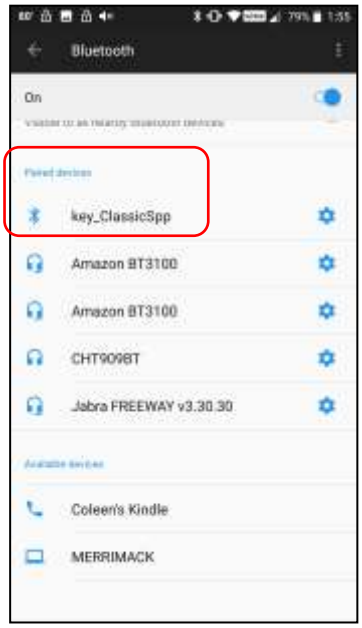
On an Android phone, you can install "Serial Bluetooth Terminal" from the Google Play Store. When you run the App, you will need to pair with your development kit. To do that open the menu (3 lines near the upper left corner) and tap on "Devices". From the Devices page, click on the "Gear" icon. This will take you to your phone's Bluetooth settings.



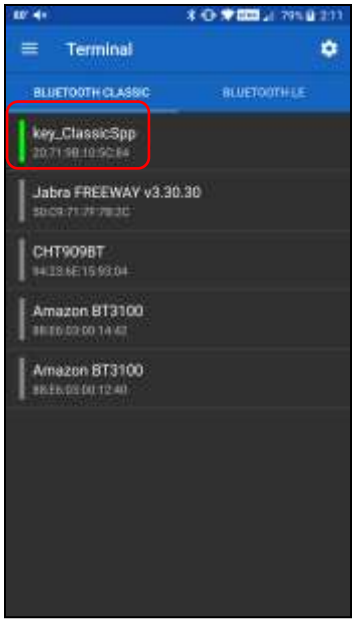


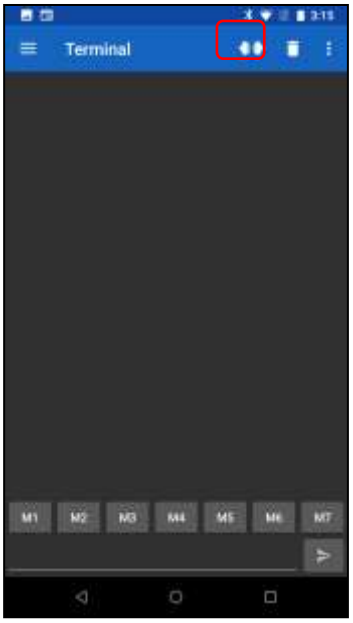
Find your device in the list and Pair with it (the exact procedure may be slightly different depending on the version of Android you are running).

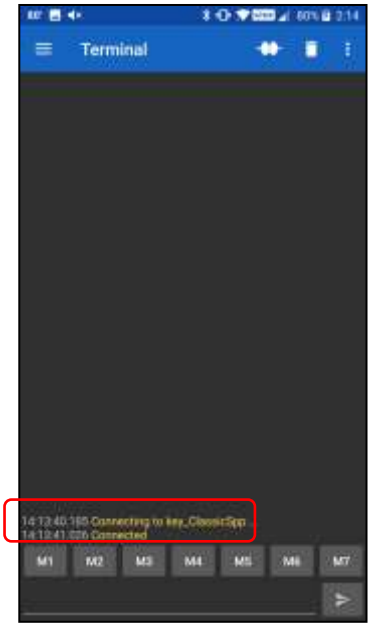




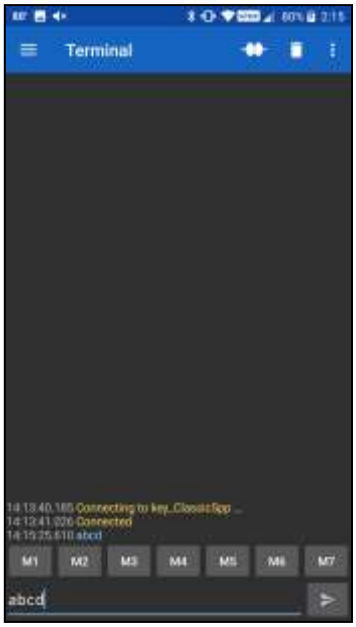
Once the device has been added, press the back arrow and you will see that your device appears in the Devices list. Tap on it to make it the active device (it will have a green bar to the left of the name when it is active). Then open the menu and select "Terminal" to see the blank terminal window. Next, tap the plug icon near the upper right corner to open the SPP server connection to your development kit. It will say "Connected" in the terminal window.



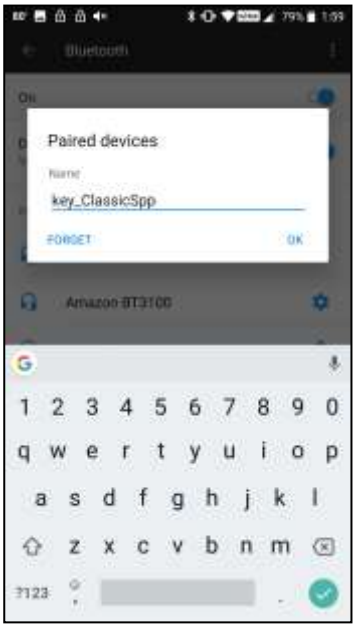




Now you can send data to the SPP server by entering it at the bottom of the window and clicking the Send arrow. You will see the data transmitted on the PUART terminal window for the kit.



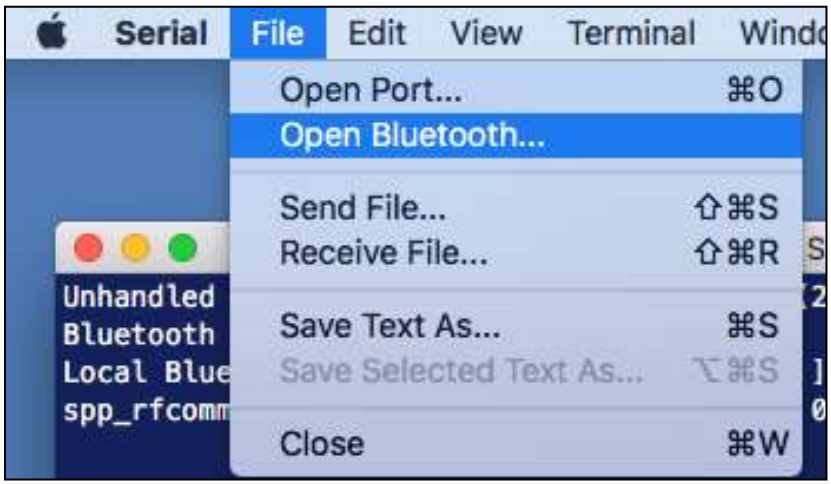
When you press the plug again, it will disconnect. You can then go back to the menu, select Devices, click on the Gear icon, and delete the Bonding information for your device (aka Forget) from the Bluetooth settings. Again, the exact procedure to forget the device will vary based on the version of Android you are running.



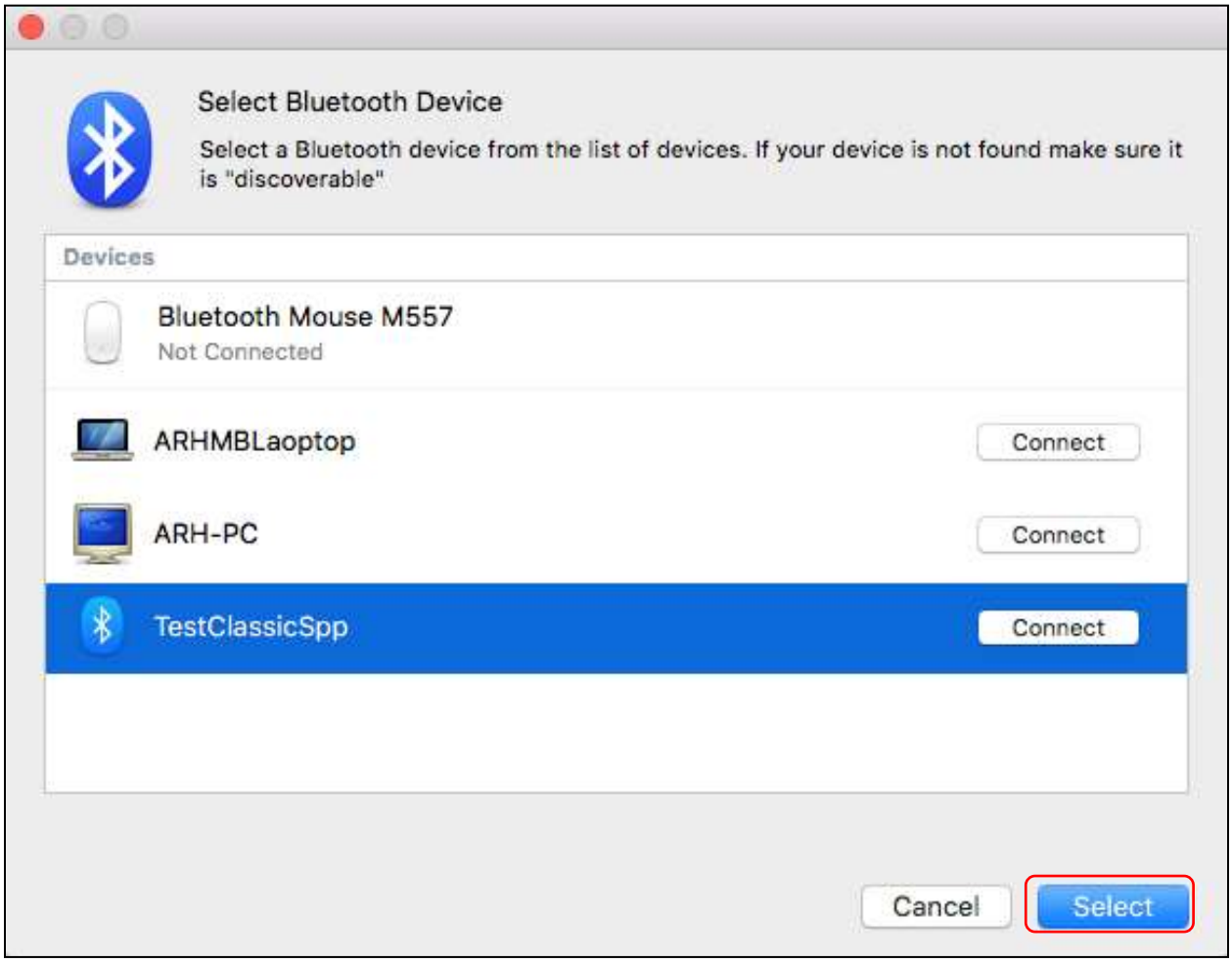
9.9-1F MAC INSTRUCTIONS

Install “Serial” from Decisive Tactics onto your Mac. You can get it in the App Store.

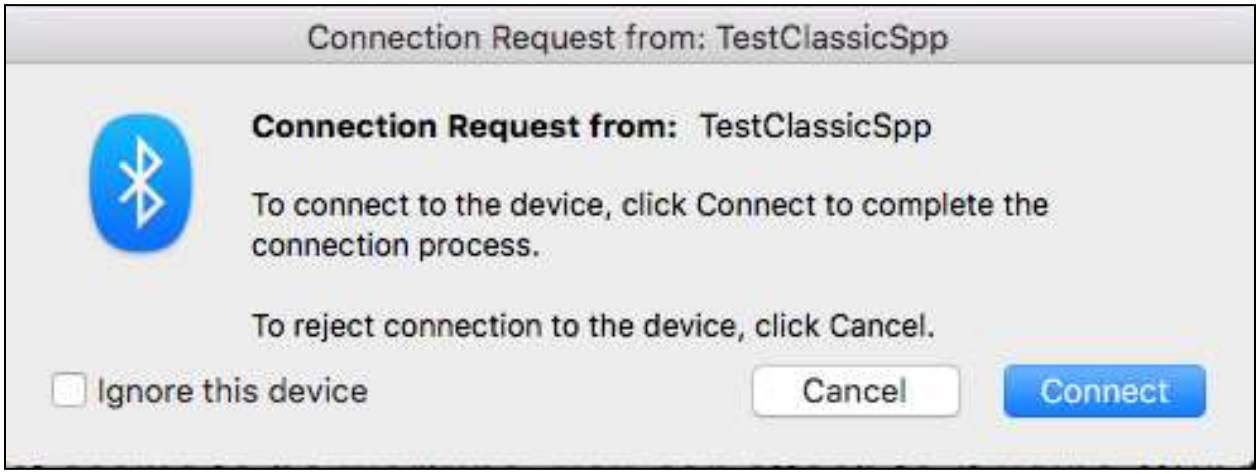
Once you have programmed the development kit you need to connect to it from the Mac. In the Serial program choose File → Open Bluetooth.



Then click on your project and press “Select”. This will pair to the development kit and open a window.



You will be asked to confirm the connection.

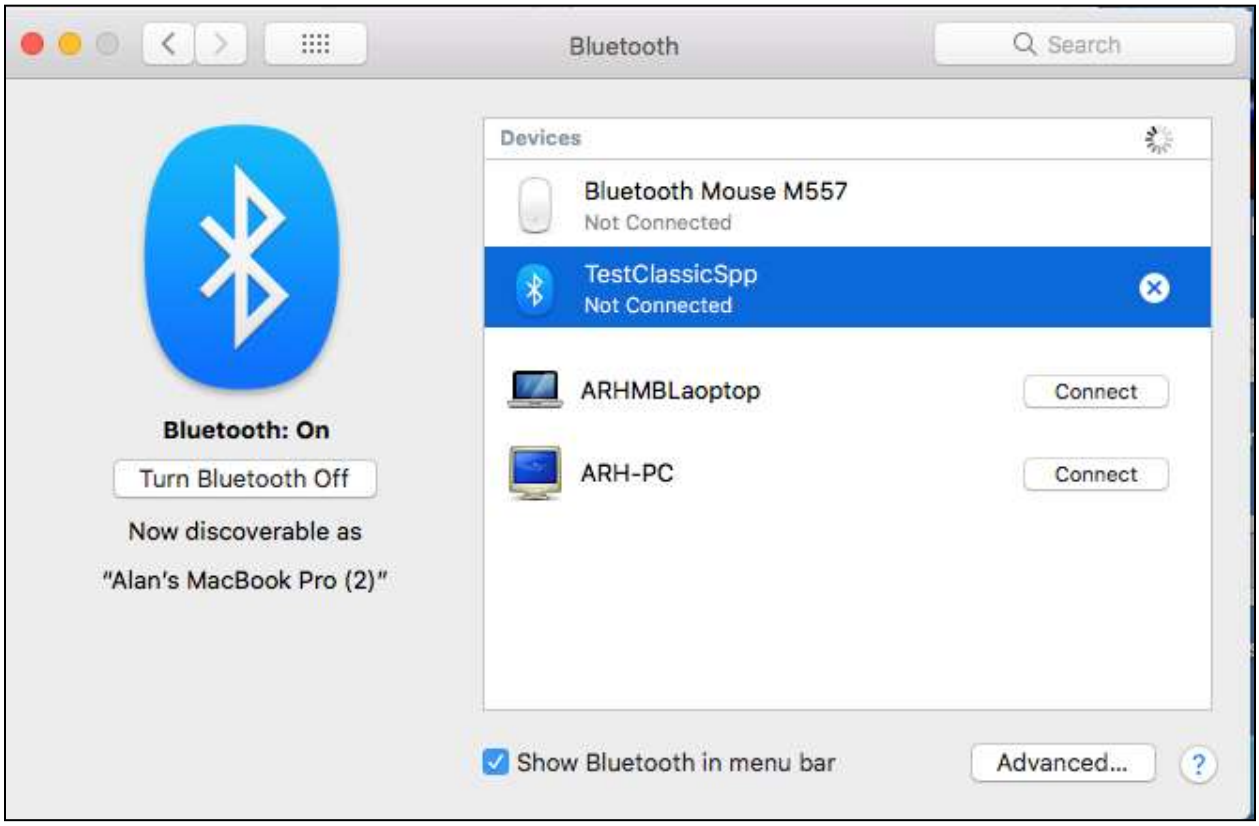


Once it is connected, everything you type will appear in the console window of the WICED Development kit. Below you can see that I typed “asdf”.

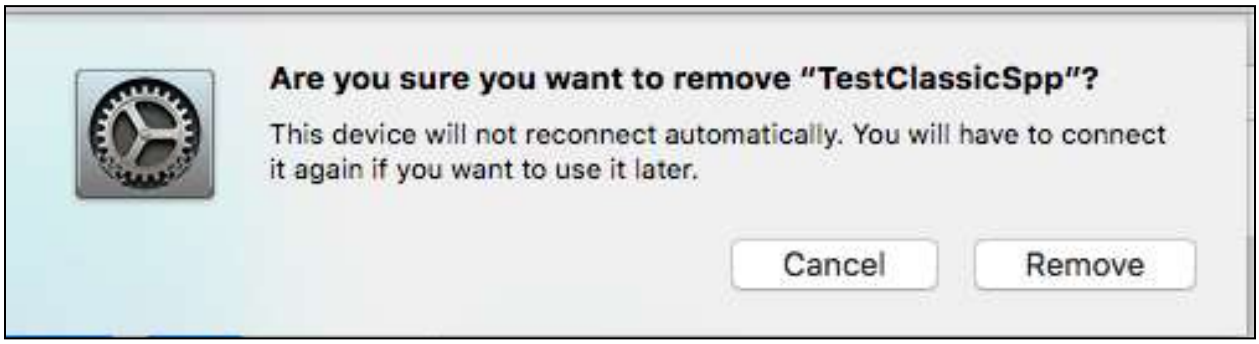
```
Local Bluetooth Address: [20 71 9b 17 19 a2 ]
spp_rfcomm_start_server: rfcomm_create Res: 0x0  Port: 0x0001
IO_CAPABILITIES_BR_EDR_RESPONSE peer_bd_addr: 78 4f 43 a2 64 f6 , peer_io_cap: 1, peer_oob_data: 0, peer_auth_req:
 2
BR/EDR Pairing IO cap Request
numeric_value: 664177
Pairing Complete 0.
BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT
NVRAM ID:512 written :136 bytes result:0
Encryption Status event: bd ( 78 4f 43 a2 64 f6  ) res 0
spp_rfcomm_control_callback : Status = 0, port: 0x0001  SCB state: 0  Srv: 0x0001  Conn: 0x0000
RFCOMM Connected  isInit: 0  Serv: 0x0001   Conn: 0x0001 78 4f 43 a2 64 f6
spp_connection_up_callback handle:1 address:78 4f 43 a2 64 f6
rfcomm_data: len:1 handle 1 data 61-61)
spp_session_data: len:1, total: 1 (session 1 data 61-61)
spp_rx_data_callback handle:1 len:1 61-61
a
rfcomm_data: len:1 handle 1 data 73-73)
spp_session_data: len:1, total: 2 (session 1 data 73-73)
spp_rx_data_callback handle:1 len:1 73-73
s
rfcomm_data: len:1 handle 1 data 64-64)
spp_session_data: len:1, total: 3 (session 1 data 64-64)
spp_rx_data_callback handle:1 len:1 64-64
d
rfcomm_data: len:1 handle 1 data 66-66)
spp_session_data: len:1, total: 4 (session 1 data 66-66)
spp_rx_data_callback handle:1 len:1 66-66
f
```

To unpair your development kit, select the Bluetooth symbol and pick “Open Bluetooth Preferences”

Select your device and click the “X”.



You will need to confirm that you want to remove the Bonding information from the Mac BT Stack.



9.9-2 ADD UART TRANSMIT

1. Use the example in the GitHub example repository CYW920819EVB02/apps/snip/bt/spp to create a project called **ch09a_ex02_spp_uart**.
2. Comment out lines 90 and 91 in spp.c to disable sending data on interrupts and on timer timeouts. We will use the PUART interface to send data to the kit that will then be transmitted over Bluetooth using the SPP send data function.
3. Launch the Change Applications Settings dialog and set BT_DEVICE_ADDRESS to random.
4. Change the name of your device by editing the variable BT_LOCAL_NAME in the wiced_bt_cfg.c file.
    a. Hint: The default name is “spp test”, remember to use your initials in the device name so that you can find it in the list of devices that will be advertising.
5. Modify spp.c to include a transmit function so that you can send data in both directions. You will read characters from the PUART terminal window so that when you type keys on your PC keyboard those values will be transmitted over Bluetooth to the Bluetooth Serial window.
    a. Hint: There is an example in the Peripherals chapter that receives characters from a PUART terminal window. Refer to that if you need help determining how to read characters from the PUART.
    b. Hint: Add a new function called *spp_tx_data* to *spp.c* to send the data. It will take a pointer to the data to send and the length as parameters and will call the *wiced_bt_spp_send_session_data* function to send the data. You will call the *spp_tx_data* function whenever a keystroke is received from the PUART.

9.9-3 (ADVANCED) IMPROVE SECURITY BY ADDING IO CAPABILITIES (YES/NO)

In this exercise, we are going change the previous exercise to add confirmation on the WICED device. As before, a numeric value will be displayed on both ends of the connection (WICED device and the phone or PC). The user will need to compare the two values and then perform the confirmation step on both devices before the connection is established.

Note: the phone or PC may or may not display the value and it may or may not require user input – this is up to the phone/PC application. In the case where the phone/PC automatically confirms the value you will still have to accept the connection on the WICED device end.

You will add the capability for the user to confirm that the numeric comparison value is correct on the WICED device using the “y” and “n” keys in the UART terminal.

To make this work you need to:

1. Use the template in folder “templates/CYW920819EVB/ch06a_ex03_spp_sec” to create a project called **ch09a_ex03_spp_sec**.
    a. Open the Change Application Settings dialog to verify that BT_DEVICE_ADDRESS is set to random.
    b. Change the device name in wiced_bt_cfg.c to include your initials.
    c. In spp.c, change the pairing_io_capabilities_ble_request.local_iop_cap to `BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT`.
    d. Add this code to the BTM_USER_CONFIRMATION_REQUEST_EVT it will print the value for the user to confirm.

```
WICED_BT_TRACE("\r\n********************\r\n" );
```

```
WICED_BT_TRACE( "\r\nNUMERIC = %06d\r\n\n",
p_event_data->user_confirmation_request.numeric_value
);
WICED_BT_TRACE("\r\n********************\r\n\n" );
```

e. Add a global variable called doCompare that is set when the user confirmation request is made and is reset after the user enters their Yes or No response.
f. Add code to the BTM_USER_CONFIRMATION_REQUEST_EVT that will set doCompare.
    i. Hint: This event will provide the BDADDR for the master that is trying to pair. You should save this value (hint: memcpy) to a global since you will need it when you reply in the interrupt.
g. In the spp_tx_data interrupt, if doCompare is set, look for "y" or "n" and then call wiced_bt_dev_confirm_req_reply() with the appropriate response. If doCompare is not set, then just send the value to the SPP interface as before.
    i. Hint: Look at the function declaration to determine what information you need to send with the reply based on which button was pressed.
    ii. Hint: Make sure you remove the existing wiced_bt_dev_confirm_req_reply() call from the BTM_USER_CONFIRMATION_REQUEST_EVT.

9.9-4 (ADVANCED) ADD MULTIPLE DEVICE BONDING CAPABILITY

In this exercise, you will add the capability to store bonding information from multiple devices. You will need to make several changes to your current SPP implementation:

1. Use the template in folder "templates/CYW920819EVB/ch06a_ex04_spp_mult" to create a project called **ch09a_ex04_spp_mult**.
2. Open the Change Application Settings dialog to verify that BT_DEVICE_ADDRESS is set to random.
3. Change the device name in wiced_bt_cfg.c to include your initials.
4. Handle saving multiple link keys into the NVRAM. Let's use 8 for the maximum number of saved link keys. Use one VSID to save a one-byte count of how many are being used. Then use VSID = VSID_Start+count to save each additional Address/Key Bonding pair.
5. Handle reading multiple link keys. When you get the event BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT, the event data will be a pointer to a wiced_bt_device_link_keys_t structure. That structure contains the BDADDR of the device that is trying to pair. You need to search through the VSIDs to find the BDADDR of the saved link keys. If you find one that matches return it. Otherwise return a WICED_ERROR so that a new device can be added.
6. Update BTM_ENABLED_EVENT to load the number of bonded devices and to overwrite the next slot when the max number of devices has been reached.
7. Update BTM_PAIRING_COMPLETE_EVT to save the BDADDR of the host to NVRAM
    a. Remember to increment the number of bonded devices and the next free slot as well as save this information to NVRAM
8. Update BTM_ENCRYPTION_STATUS_EVT to search for the BDADDR trying to connect in NVRAM. If found restore values to a current host structure.
9. Add BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT and write the code to save the new link keys to NVRAM.
10. Add BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT and write the code to search for the BD_ADDR in NVRAM. If not return WICED_BT_ERROR and have the stack generate new keys and call BTM_LINK_KEYS_UPDATE_EVT so that they are stored.
11. Add BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT to save local keys to NVRAM and BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT to read local keys from NVRAM.

Once you have the project completed, try bonding with two different devices (e.g. phone and PC). Connect and disconnect back and forth to verify that bonding information for both is retained and is used when reconnecting.

9.10 EXERCISES – MESH TOPOLOGY (COPIED FROM WBT101-07A)

9.10-1 CREATE NETWORK WITH A LIGHTDIMMABLE DEVICE

In this exercise you will create your own (very small) mesh network.

1. In ModusToolbox IDE, create a new application for:
    a. Target Hardware: CYBT-213043-MESH
    b. Starter Application: BLE_Mesh_LightDimmable

2. Open the file "light_dimmable.c" and find the "mesh_dev_name". Change the name so that it has your initials in it (e.g. "<Inits> Dimmable Light").
3. Program the project to one of the CYBT-213043-MESH kits.
    a. Hint: You should open a terminal window for the PUART to see messages. **The default PUART baud rate for the mesh applications is 921600**.
        i. Hint: If your terminal emulator does not support 921600, from ModusToolbox, open the file in libraries/mesh_app_lib/mesh_app_hci.c and search for 921600. Change that value to one that is supported and rebuild/reprogram.
4. Run the Mesh Lighting application to provision the device.
    a. Hint: If you don't have an Android smartphone, the instructor can provide a tablet with the app pre-installed.
    b. Hint: If you don't see any devices listed after ~10 seconds, exit the app, stop/restart BLE and then try again.

9.11 EXERCISES – MESH DETAILS (COPIED FROM WBT101-07B)

9.11-1 ADD MORE LIGHTS TO THE NETWORK AND CREATE/MODIFY GROUPS

In this exercise you will add additional lights to the network you created in the previous chapter. You will experiment with associating devices to different groups.

1. Program your LightDimmable application into 3 more mesh kits.
    a. Hint: Unplug the other kits while programming each one or move them to an alternate power source that isn't connected to your computer.
2. Provision the new LightDimmable kits.
3. Optional: Rename the new kits in the app so that you can distinguish them.
4. Create two Rooms (i.e. Groups) and add 2 of the lights to each Room.
    a. Hint: Leave all the lights in the group "All". That is, just add them to the new rooms, don't move them.
5. Experiment with controlling all lights at once (All), one room at a time, and individually.
6. Optional: experiment with other room configurations. For example:
    a. Light 1 is in Room 1 and All
    b. Light 2 is in Room 2 and All
    c. Light 3 is in Room 1, Room2 and All
    d. Light 4 is only in All

9.12 EXERCISES – MESH FIRMWARE (COPIED FROM WBT101-07C)

9.12-1 ADD AN ONOFF SWITCH TO YOUR NETWORK

In this exercise you will add an on/off switch to your mesh network that can control LEDs on the LightDimmable kit(s).

1. Remove one of your LightDimmable kits from your mesh network using the Android app.
2. Create a new application for:
    a. Target Hardware: CYBT-213043-MESH
    b. Starter Application: BLE_Mesh_OnOffSwitch
3. Open the file "on_off_switch.c" and find the "mesh_dev_name". Change the name so that it has your initials in it (e.g. "<Inits> Switch")
4. Program the project into the mesh kit.

   a. Hint: You may want to label the kits to keep track of which one is programmed with each project. Remember if you accidentally press the user button on the LightDimmable kit, it will perform a factory reset so that kit will need to be re-provisioned.
5. Provision the OnOff Switch kit to your network.
6. Press the user button on the OnOff Switch kit to toggle the LEDs on the LightDimmable kits.
7. Experiment with changing the Assignment for the OnOff Switch to control different lights or rooms.
   a. Hint: You can't move switches to different rooms, rather you Assign the device or rooms that the switch will control.
8. Note that you can still control the lights using the app.

9.12-2 ADD A DIMMER TO THE NETWORK

In this exercise you will add a dimmer device to your mesh network. This new device will be able to turn the LED on/off as well as control the brightness of the LED on the LightDimmable kit. The OnOff Switch from the previous exercise will be able to control the same LED.

1. Remove one of your LightDimmable kits from your mesh network using the Android app.
2. Create a new application for:
   a. Target Hardware: CYBT-213043-MESH
   b. Starter Application: BLE_Mesh_Dimmer
3. Open the file "dimmer.c" and find the "mesh_dev_name". Change the name so that it has your initials in it (e.g. "<Inits> Dimmer")
4. Program the project into the kit.
   a. Hint: You may want to label the kits to keep track of which one is programmed with each project. Remember if you accidentally press the user button on the LightDimmable kit, it will perform a factory reset.
5. Provision the Dimmer kit to your network.
6. Press the user button on the Dimmer kit to toggle the LEDs on the LightDimmable kits.
7. Press and hold the user button on the Dimmer kit to adjust the brightness of the LEDs.
   a. Hint: If you hold the button for longer than 15 seconds a factory reset will be performed and the Dimmer kit will no longer be associated with the mesh network.
8. Verify that the OnOff switch kit and the app can still control the LED.
9. Experiment with Assigning the Dimmer to different lights or rooms.

9.12-3 (ADVANCED) ADD A 2ND ELEMENT FOR THE GREEN LED TO LIGHTDIMMABLE

In this exercise, you will add a new element to the LightDimmable application so that you can control the Red and Green LEDs on the kit individually. To do this:

1. Use the app to remove one of the LightDimmable devices from your network.
2. Create a new application:
   a. Target Hardware: CYBT-213043-MESH
   b. Starter Application: BLE_Mesh_LightDimmable
   c. Application Name: BLE_Mesh_LightDimmable_ch9c_ex05
3. In the light_dimmable.c file:
   a. Add another element to the design. This new element will have one WICED_BT_MESH_MODEL_LIGHT_LIGHTNESS_SERVER model and no properties.
      i. Hint: You will need to create the mesh_element2_models array.

ii. Hint you will need to add a set of entries to the mesh_elements array for the new element.
iii. Hint: Make sure you set the number of models and number of properties in the mesh_elements array to the correct values for this new element.

b. In the mesh_app_init function, initialize the new light lightness server.
i. Hint: You can use the same callback for both light lightness servers since it is passed the element index when it is called.

4. In led_control.c:
a. Add a define for another PWM channel and add code to led_control_init to initialize the new PWM.
i. Hint: The configurator was not used in the LightDimmable demo application so for consistency you can set up the PWM the same way.
ii. Hint: Copy the code for PWM0 and update it to use PWM1.
iii. Hint: the BSP has a #define for LED_GREEN that you can use.
iv. Hint: you can use the same pwm_config structure for both PWMs – just call wiced_hal_gpio_select_function and wiced_hal_pwm_start two times each – once for each PWM.
b. Add an additional parameter to the function led_control_set_brighness_level so that it knows which element a message is intended for.
i. Hint: unit8_t element_idx.
ii. Hint: remember to update the function prototype in led_control.h too.
c. Update the led_control_set_brighness_level function so that it looks at the element_idx input and updates the appropriate PWM.
5. Back in light_dimmable.c:
a. Search for calls to led_control_set_brighness_level (yes, the 't' is missing in brightness) and add the element_idx parameter.
i. Hint: The timer callback function (attention_timer_cb) doesn't have access to element_idx, but when you init the timer you can set it to pass a uint32_t to the callback. Therefore, you can:
1. Set up a global uint32_t to hold the index value.
2. Pass that variable as an argument when you init the timer
3. Update the value of the variable with the element_idx value just before starting the timer.
ii. Hint: If you don't want to deal with the above, you can just hard code the element_idx to 0 in mesh_app_attention and attention_timer_callback.
6. Program your kit.
7. Provision your device onto your network.
a. Hint: You should see 2 devices show up for this kit instead of just one.
8. Control each of the LEDs from the app individually using the two separate devices
a. Note that the OnOff Switch and Dimmer control both LEDs simultaneously because they control everything in the group at once.

9.12-4 (ADVANCED) UPDATE LIGHTDIMMABLE TO USE THE HSL MODEL

In this exercise, we will change the light lightness model to the Light HSL model. This model extends the light lightness model by adding the ability to control Hue and Saturation. We will use all 3 LEDs in the RGB LED for this exercise.

1. Use the app to remove one LightDimmable device from your network.
2. Create a new application:
    a. Target Hardware: CYBT-213043-MESH
    b. Starter Application: BLE_Mesh_LightDimmable
    c. Application Name: BLE_Mesh_LightDimmable_ch9c_ex06
3. In the light_dimmable.c file:
    a. Remove the mesh_app_attention callback functionality to simplify the changes required.
    b. Change the model from WICED_BT_MESH_MODEL_LIGHT_LIGHTNESS_SERVER to WICED_BT_MESH_MODEL_LIGHT_HSL_SERVER in the appropriate places.
    c. Add 2 additional elements each with one model. The required models are:
        i. WICED_BT_MESH_MODEL_LIGHT_HSL_HUE_SERVER
        ii. WICED_BT_MESH_MODEL_LIGHT_HSL_SATURATION_SERVER
    d. Change the wiced_bt_mesh_model_lightness_server_init function call to wiced_bt_mesh_model_light_hsl_server_init.
    e. In the message handler, change the WICED_BT_MESH_LIGHT_LIGHTNESS_SET event to WICED_BT_MESH_LIGHT_HSL_SET.
    f. Change the function mesh_app_process_set_level to mesh_app_process_set_hsl and make the changes to get the Hue, Saturation and Lightness values from p_status.
        i. Hint: the data provided in p_status will be of type wiced_bt_mesh_light_hsl_status_data_t.
        ii. Hint: Use "Open Declaration" on that datatype to find out what it contains.
        iii. Hint: Change the calls to led_control_set_brighness_level to pass the Hue, Saturation, and Lightness instead of last_known_brightness.
            1. Hint: Hue, Saturation and Lightness are of type unit16_t.
4. In led_control.c:
    a. Set up two additional PWMs – one for the Green LED and one for the Blue LED.
        i. Hint: Use PWM1 and PWM2.
        ii. Hint: the BSP has #defines for LED_GREEN and LED_BLUE that you can use.
        iii. Use the function shown below to convert the HSL values to RGB values and then use the RGB values to control the three LEDs. (In future releases of the SDK, this will be provided as middleware).
            1. Hint: The HSL inputs can be passed in directly from what the model provides. The outputs are provided as pointers to three uint8_t variables (r, g, and b).

```
/* Convert HSL values to RGB values */
/* Inputs: hue (0-360), sat (0-100), and lightness (0-100)
*/
/* Outputs: r (0-100), g (0-100), b (0-100) */
void HSL_to_RGB(uint16_t hue, uint16_t sat, uint16_t light,
uint8_t* r, uint8_t* g, uint8_t* b)
{
      uint16_t v;
```

```
    /* Formulas expect input in the range of 0-255 so we
need to convert
       the input ranges which are H: 0-360, S: 0-100, L: 0-
100 */
    hue = (hue*255)/360;
    sat = (sat*255)/100;
    light = (light*255)/100;

    v = (light < 128) ? (light * (256 + sat)) >> 8 :
          (((light + sat) << 8) - light * sat) >> 8;
    if (v <= 0) {
        *r = *g = *b = 0;
    } else {
        int m;
        int sextant;
        int fract, vsf, mid1, mid2;

        m = light + light - v;
        hue *= 6;
        sextant = hue >> 8;
        fract = hue - (sextant << 8);
        vsf = v * fract * (v - m) / v >> 8;
        mid1 = m + vsf;
        mid2 = v - vsf;

        // Convert output range of 0-255 to 0-100
        v = (v*100)/255;
        m = (m*100)/255;
        mid1 = (mid1*100)/255;
        mid2 = (mid2*100)/255;

        switch (sextant) {
           case 0: *r = v; *g = mid1; *b = m; break;
           case 1: *r = mid2; *g = v; *b = m; break;
           case 2: *r = m; *g = v; *b = mid1; break;
           case 3: *r = m; *g = mid2; *b = v; break;
           case 4: *r = mid1; *g = m; *b = v; break;
           case 5: *r = v; *g = m; *b = mid2; break;
        }
    }
}
```

Note: This function currently expects ranges for Hue of 0-360, Saturation of 0-100 and lightness of 0-100. This is what the app currently supplies. However, according to the BT Mesh spec, these should all be in the range of 0-65535. At some point this will be fixed in the apps and at that time this function will need to change accordingly.

5. Program your kit.
6. Provision your device onto your network.

7. Use the app to adjust the light color and intensity. Also note that if you turn the light on/off using an on/off switch (either in a kit or in the app), the device remembers the last value so that when you turn it back on the color and brightness are the same.

CHAPTER QUESTIONS ANSWER KEY

CHAPTER 1

1.9-1

1. Where is the documentation for the PWM API located?
   It is in the WICED API Reference. The path in that document is:
   Cypress WICED API Reference Guide
   Components
   Hardware Drivers
   PWM

1.9-2A

1. What is the name of the first user application function that is executed? What does it do?
   The first user function is application_start().
   It just initializes the Bluetooth stack and registers the callback.
2. What is the purpose of the function app_bt_management_callback? When does the BTM_ENABLED_EVT case occur?
   It is the Bluetooth stack management callback function. It is called whenever there is a management event from the stack.
   The BTM_ENABLED_EVT case occurs once the stack has completed initialization.
3. What controls the rate of the LED blinking?

The first parameter to the RTOS delay function wiced_rtos_delay_milliseconds specifies the delay which controls the rate of the LED blinking.

9.2-9A

1. How many bytes does the NVRAM read function get before you press the button the first time?

   0

2. What is the return status value before you press the button the first time?

   The return value is 40 (0x28).

   1. What does the return value mean?

      The return value of 40 (0x28) means ERROR. This is defined in the file wiced_result.h.

9.3-1A

1. Do you need *wiced_rtos_delay_millisecon*ds() in the LED thread? Why or why not?
   No, because the wiced_rtos_get_semaphore will cause the thread to suspend each time through the infinite loop while it waits for another button press.

9.3-2A

1. Before you added the mutex, how did the LED behave when you pressed the button?

   The LED flashes in an irregular pattern.

2. What changed when you added the mutex?

   The LED flashes slowly (2Hz) when the button is not pressed and then flashes quickly (5Hz) when the button is pressed.

3. What happens if you forget to unlock the mutex in one of the threads? Why?

   The other thread will never execute. That is:

   If you don't unlock the mutex in the fast LED thread, the slow LED thread will not execute. The LED will blink fast when the button is pressed but will not blink when the button is not pressed.

   If you don't unlock the mutex in the slow LED thread, the fast LED thread will never execute so the LED will blink slowly no matter what happens with the button.

9.4-3D

1. How many bytes is the advertisement packet?

   The advertisement packet is 17 bytes (assuming your initials are 3 characters). They are:

   - Flags (3)
     - Length (1)
     - Type (1)
     - Data (1)
   - Local Name (9)
     - Length (1)
     - Type (1)
     - Data (7)
   - Manufacturer Specific Data (5)
     - Length (1)
     - Type (1)
     - Data (3)

9.4-4D

1. What function is called when there is a Stack event? Where is it registered?
   The function is *app_bt_ management_callback*. It is registered using *wiced_bt_stack_init* in *application_start*.

2. What function is called when there is a GATT database event? Where is it registered?
   The function is *app_gatt_callback*. It is registered using *wiced_bt_gatt_register* in *app_bt_management_callback* in the BTM_ENABLED_EVT event.

3. Which GATT events are implemented? What other GATT events exist? (Hint: right click and select Open Declaration on one of the implemented events)

   Implemented:
   - GATT_CONNECTION_STATUS_EVT
   - GATT_ATTRIBUTE_REQUEST_EVT

   Others:
   - GATT_OPERATION_CPLT_EVT
   - GATT_DISCOVERY_RESULT_EVT
   - GATT_DISCOVERY_CPLT_EVT
   - GATT_CONGESTION_EVT

4. In the GATT "GATT_ATTRIBUTE_REQUEST_EVT", what request types are implemented? What other request types exist?

   Implemented:
   - GATTS_REQ_TYPE_READ
   - GATTS_REQ_TYPE_WRITE

   Others:
   - GATTS_REQ_TYPE_PREP_WRITE
   - GATTS_REQ_TYPE_WRITE_EXEC

GATTS_REQ_TYPE_MTU
GATTS_REQ_TYPE_CONF

9.5-2D

1. How long does the device stay in high duty cycle advertising mode? How long does it stay in low duty cycle advertising mode? Where are these values set?
   High: 30 seconds
   Low: 60 seconds
   These are specified in the wiced_bt_cfg.c file in
   wiced_bt_cfg_settings.ble_advert_cfg.high_duty_duration and
   wiced_bt_cfg_settings.ble_advert_cfg.low_duty_duration

9.5-3E

1. What items are stored in NVRAM?

   Hostinfo (Remote BDADDR and Button CCCD state)
   Local Keys (Privacy Information)
   Paired Device Keys (Encryption Information)

2. Which event stores each piece of information?
   Hostinfo is stored during BTM_PAIRING_COMPLETE_EVT and in ex03_ble_bond_set_value if the Button CCCD value was written
   Local Keys are stored during BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT
   Paired Keys are stored during BTM_PAIRED_DEVICE_LINK_KEYS_UPDATE_EVT
   All three are cleared out (i.e. reset) in the button_cback function to allow re-pairing.
3. Which event retrieves each piece of information?
   Hostinfo is retrieved by BTM_ENCRYPTION_STATUS_EVT (if the device was previously bonded
   Local Keys are retrieved by BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT
   Paired Keys are retrieved by ex03_ble_bond_app_init (at startup) and by BTM_PAIRED_DEVICE_LINK_KEYS_REQUEST_EVT
4. In what event is the privacy info read from NVRAM?
   BTM_LOCAL_IDENTITY_KEYS_REQUEST_EVT
5. Which event is called if privacy information is not retrieved after new keys have been generated by the stack?
   BTM_LOCAL_IDENTITY_KEYS_UPDATE_EVT

9.5-4D

1. Other than BTM_IO_CAPABILITIES_NONE and BTM_IO_CAPABILITIES_DISPLAY_ONLY, what other choices are available? What do they mean?
   BTM_IO_CAPABILITIES_DISPLAY_AND_YES_NO_INPUT
   Device can display values (e.g. 6-digit numbers) and can accept a Yes/No input from the user.
   BTM_IO_CAPABILITIES_KEYBOARD_ONLY
   Device can accept input (e.g. numbers) but cannot display any values.
   BTM_IO_CAPABILITIES_BLE_DISPLAY_AND_KEYBOARD_INPUT
   Device can display values (e.g. 6-digit numbers) and can accept input (e.g. numbers).

2. What additional stack callback event occurs compared to the previous exercise? At what point does it get called?

   BTM_PASSKEY_NOTIFICATION_EVT

   This event is called between BTM_PAIRING_IO_CAPABILITIES_BLE_REQUEST_EVT and BTM_ENCRYPTION_STATUS_EVT.

9.6-1D

1. Which lines in the code are used to configure and initialize sleep?
   Lines 117 – 128:

```
/* Configure and initialize sleep */
sleep_cfg.sleep_mode                = WICED_SLEEP_MODE_NO_TRANSPORT;
sleep_cfg.device_wake_mode                  = WICED_SLEEP_WAKE_ACTIVE_LOW;
sleep_cfg.device_wake_source        = WICED_SLEEP_WAKE_SOURCE_GPIO;
sleep_cfg.device_wake_gpio_num              = WICED_GPIO_PIN_BUTTON_1;
sleep_cfg.host_wake_mode                    = WICED_SLEEP_WAKE_ACTIVE_LOW;
sleep_cfg.sleep_permit_handler              = low_power_sleep_callback;

if(WICED_BT_SUCCESS != wiced_sleep_configure(&sleep_cfg))
{
     WICED_BT_TRACE("Sleep Configure Failed!\n\r");
}
```

2. When in the code is sleep configured (i.e. after which event)?
   Sleep configuration is done in the BTM_ENABLED_EVT callback. This executes once the stack has been enabled.

3. What is used as a wakeup source?
   WICED_GPIO_PIN_BUTTON_1 is used as a wakeup source.

4. What is the name of the sleep permit handler function?

   low_power_sleep_callback

5. When are the connection interval min, max, latency, and timeout values updated and what values are used?

   The values are updated in the GATT connect callback function when a connection is established. The code is:

```
    wiced_bt_l2cap_update_ble_conn_params(p_conn_status->bd_addr,
200, 200, 3, 512);
```

   The values are set to:

Min Interval: 250ms	(200 * 1.25ms)
Max Interval: 250ms	(200 * 1.25ms)
Latency: 3	
Timeout: 5120ms	(512 * 10ms)