Chapter 2: Application Architecture

This chapter will cover details of the application architecture used for AIROC™ Bluetooth® SDK (BTSDK) MCUs. There are a few notable differences compared to applications used for other device families such as PSoC™, so it is important to understand them before digging into the following chapters.

Table of contents

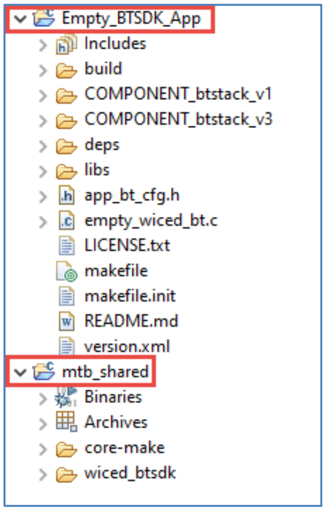
2.1 Application Directory Organization 2
2.1.1 Dependencies 2
2.1.2 Application Source Files 4
2.2 Application makefile 5
2.2.1 SUPPORTED_TARGETS 5
2.2.2 Application Features 5
2.2.3 Application Defines 5
2.3 BSP and Device Driver Library 6
2.3.1 WICED HAL 6
2.3.2 Platform 7
2.3.3 BSP Configuration 8
2.3.4 Custom BSP Configurations 8
2.4 Firmware Structure 9
2.5 Application README file 10
2.6 Exercises 11
Exercise 1: Create and Explore the Empty_BTSDK_App 11

Document conventions

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

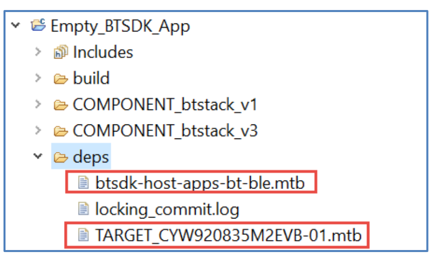
2.1 Application Directory Organization

The process to create a new application for a BTSDK device is exactly the same as with any other device in ModusToolbox™. Typically, that means you use the Project Creator tool to select a board support package (BSP) and a template application. Once the application has been created, the directory structure looks similar to any other application – there is a top-level project and a separate directory called *mtb_shared* containing libraries that can be shared between multiple applications in the workspace.

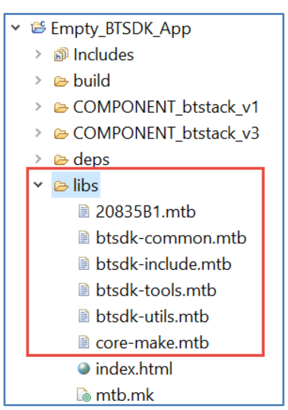


2.1.1 Dependencies

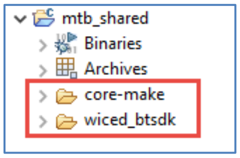
If we open up the *deps* directory in the application, you will see that the application includes the BSP and a library containing a set of Bluetooth® LE host applications (these are applications that you can run on your computer to test Bluetooth® functionality).



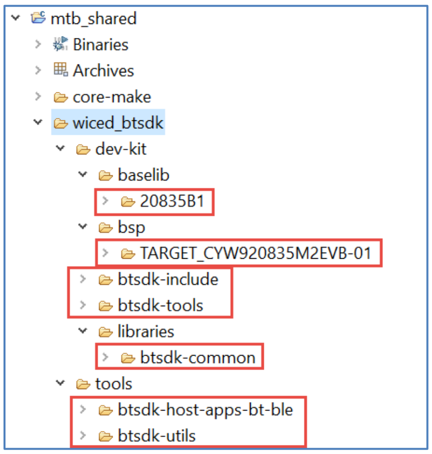
In the *libs* directory, you will see a device driver library (*20820A1*), the *core-make* library, and several other *btsdk-** libraries included as dependencies by the BSP. Notice that the BSP library is <u>not</u> in *libs*. That's the first difference. For BTSDK applications, the BSP is shared by default instead of being local to the application.



The next thing you may notice is that the *mtb_shared* directory has only two sub-directories: one for the *core-make* library and one called *wiced_btsdk*. So, where are all of the other libraries?



The answer to that question is the second difference. Namely, all of the BTSDK related libraries are stored under the *wiced_btsdk* directory. If we open a few more levels of hierarchy, you can see where each of the libraries is placed.

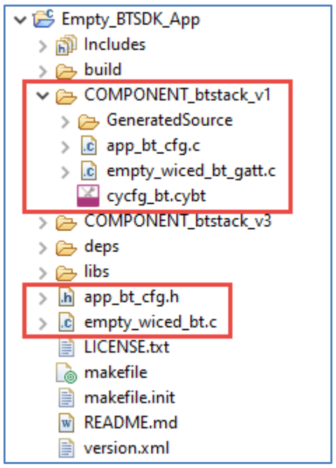


The location of each library relative to *mtb_shared* is specified in each library's **.mtb* file. For example, the path specified in the *btsdk-common.mtb* file looks like this:

```
$$ASSET_REPO$$/wiced_btsdk/dev-kit/libraries/btsdk-common/release-v3.3.0
```

2.1.2 Application Source Files

BTSDK MCUs are meant to be used for Bluetooth® applications – after all, if you are buying a device with Bluetooth® capabilities, you probably want to use it. Therefore, all applications are structured with that in mind. As an example, the relevant source files for the Empty_BTSDK_App code example are highlighted below.



The first thing you may notice is there is no file called *main.c*. Rather, the source code files are named based on the application. For example, the *Empty_BTSDK_App* code example has a file called *empty_wiced_bt.c*. That file contains the application's entry point. There is no reason it couldn't be called *main.c*, but that's how the code examples were created.

The other relevant files you will see are used for configuring Bluetooth® settings. This involves the following files:

app_bt_cfg.c / app_bt_cfg.h	Stack configuration settings
empty_wiced_bt_gatt.c	GATT event handler
cycfg_bt.cybt	Bluetooth® configurator file

Note: *These files may have different names in different code examples or template applications but the purpose is the same.*

Note: *The contents of these files will be discussed in detail in the level 3 Bluetooth® class.*

Note: *Some of these files are included via a* `COMPONENT` *directory. That's to make the code examples usable across different devices. The value for the* `COMPONENT` *is set inside the BSP and should not be changed.*

For the BSP used in this class, the `COMPONENT_btstack_v1` *files will be used from the code examples. The template applications provided for some of the exercises in this class are only intended for btstack_v1 BSPs, so the* `COMPONENT` *mechanism will not be used in those applications.*

The differences between the btstack_v1 and btstack_v3 versions of these files will be discussed in the level 3 Bluetooth® class.

2.2 Application makefile

The application *makefile* for BTSDK devices contains most of the same variables as any other ModusToolbox™ application. However, there are a few differences you should be aware of.

2.2.1 SUPPORTED_TARGETS

The first difference is that BTSDK applications will only build if the target you are using (i.e. BSP) is listed as a supported target for that application. This is done using the `SUPPORTED_TARGETS` variable. The variable contains the list of valid targets for the application. For example, a partial list may look like this:

```
SUPPORTED_TARGETS = \
  CYW920835M2EVB-01 \
  CYW920820M2EVB-01 \
  …
  CYW955572BTEVK-01 \
  CYW920721M2EVB-03
```

Note: *Back-slash characters are used as continuations. That is, the back-slash characters are used so that the BSP list can be more readable than if they were all entered on a single line.*

Note: *If you change the TARGET for an application, make sure it is in the SUPPORTED_TARGETS list; otherwise, the application will not build.*

2.2.2 Application Features

The next difference in the *makefile* for BTSDK applications is the set of application features. These are variables used to control various aspects of the application. An example list of features is shown here. We will talk about some of these in detail later in this class or in the level 3 Bluetooth® class, but for now, just leave them as-is.

```
OTA_FW_UPGRADE?=0
BT_DEVICE_ADDRESS?=default
UART?=AUTO
XIP?=xip
TRANSPORT?=UART
ENABLE_DEBUG?=0
```

2.2.3 Application Defines

The final notable difference in the *makefile* is a define for `WICED_BT_TRACE_ENABLE`. This is used to enable debug printing from the application as you will see in a later chapter. You can remove (or comment out) this variable if you want to eliminate debug UART messages. This is typically done once a device goes into production. We will leave it enabled for all of our exercises.

```
CY_APP_DEFINES+=\
    -DWICED_BT_TRACE_ENABLE
```

2.3 BSP and Device Driver Library

As mentioned above, BSPs are located in the *mtb_shared/wiced_btsdk/dev-kit/bsp* directory, so the BSP is shared by all applications in a workspace by default. Likewise, the drivers for the MCU are in *mtb_shared/wiced_btsdk/dev-kit/baselib.*

The BSP and device driver libraries contain macros for pins and other useful items such as LED polarities. These are split across a few different files that give you a few ways to access things.

2.3.1 WICED HAL

The first set of files are in the device driver library under *baselib/<device>/<version>/COMPONENT_<device>/include/hal*. This directory contains header files with definitions for on-chip peripherals to use with the HAL functions. For example, the file *wiced_hal_gpio.h* has the following enumeration for the GPIO pins on the chip:

```
/** GPIO Numbers: last 8 are ARM GPIOs and rest are LHL GPIOs */
typedef enum
{
    /* GPIO_P00 to GPIO_P39 are LHL GPIOs */
    WICED_P00 = 0,   /**< LHL GPIO 0 */
    WICED_P01,       /**< LHL GPIO 1 */

    WICED_P02,       /**< LHL GPIO 2 */
    …
    WICED_P38,       /**< LHL GPIO 38 */
    WICED_P39,       /**< LHL GPIO 39 */
    /* GPIO_00 to GPIO_07 are ARM GPIOs */
    WICED_GPIO_00,   /**< ARM GPIO 0 - 40 */
    WICED_GPIO_01,   /**< ARM GPIO 1 - 41 */
    …
    WICED_GPIO_06,   /**< ARM GPIO 6 - 46 */
    WICED_GPIO_07,   /**< ARM GPIO 7 - 47 */
    MAX_NUM_OF_GPIO
}wiced_bt_gpio_numbers_t;
```

As a second example, the *wiced_hal_pwm.h* file has:

```
/**
* PWM HW block has 6 PWM channels each with its own 16 bit counter.
* The first PWM channel is PWM0.
*/
typedef enum
{
    PWM0  = 0,
    PWM1  = 1,
    PWM2  = 2,
    PWM3  = 3,
    PWM4  = 4,
    PWM5  = 5,
    MAX_PWMS = 6
} PwmChannels;
```

2.3.2 Platform

Next is the file *wiced_platform.h,* which is at the top-level of the BSP. It will have definitions such as LED and button pins, LED on and off state, button pressed or not pressed, etc. For example:

```
/*! pin for button 1 */
#define WICED_GPIO_PIN_BUTTON_1       WICED_P00
#define WICED_GPIO_PIN_BUTTON         WICED_GPIO_PIN_BUTTON_1

/*! configuration settings for button, x can be GPIO_EN_INT_RISING_EDGE or
GPIO_EN_INT_FALLING_EDGE or GPIO_EN_INT_BOTH_EDGE */
#define WICED_GPIO_BUTTON_SETTINGS(x)                        (
GPIO_INPUT_ENABLE | GPIO_PULL_UP | x )

/*! pin for LED 1 */
#define WICED_GPIO_PIN_LED_1     WICED_P27
/*! pin for LED 2 */
#define WICED_GPIO_PIN_LED_2     WICED_P26

#define WICED_PUART_TXD          WICED_P32
#define WICED_PUART_RXD          WICED_P29

/** Pin state for the LED on. */
#ifndef LED_STATE_ON
#define LED_STATE_ON           (GPIO_PIN_OUTPUT_LOW)
#endif
/** Pin state for the LED off. */
#ifndef LED_STATE_OFF
#define LED_STATE_OFF          (GPIO_PIN_OUTPUT_HIGH)
#endif

/** Pin state for when a button is pressed. */
#ifndef BTN_PRESSED
#define BTN_PRESSED            (GPIO_PIN_OUTPUT_LOW)
#endif
/** Pin state for when a button is released. */
#ifndef BTN_OFF
#define BTN_OFF                (GPIO_PIN_OUTPUT_HIGH)
#endif
```

2.3.3 BSP Configuration

The final set of files of interest are the device configuration. As with other ModusToolbox™ applications, these are generated by the configurator, so you can either view the files inside the BSP under *COMPONENT_bsp_design_modus/GeneratedSource*, or you can just open the Device Configurator from the application to see the values. For example, you can see aliases available for the pins from the Device Configurator:

Resource	Name(s)	Personality
☑ Pins	ioss_0	Pins-1.0
☑ P0	CYBSP_D2,SW3,USER_BUTTON1	Pin-1.0
☐ P1	ioss_0_pin_1	
☑ P2	CYBSP_D4	Pin-1.0
☑ P3	CYBSP_D5	Pin-1.0
☑ P4	CYBSP_D6	Pin-1.0
☑ P5	CYBSP_D7	Pin-1.0
☑ P6	CYBSP_D11	Pin-1.0
☑ P7	CYBSP_D13	Pin-1.0
☑ P8	CYBSP_A0,CYBSP_THERM_TEMP_SENSE	Pin-1.0
☑ P9	CYBSP_D8	Pin-1.0
☑ P10	CYBSP_A2,PUART_CTS	Pin-1.0
☑ P11	CYBSP_A3,PUART_RTS	Pin-1.0
☑ P12	CYBSP_A4	Pin-1.0
☑ P13	CYBSP_A5	Pin-1.0
☑ P14	CYBSP_D10	Pin-1.0
☐ P15/XTALI_32K	ioss_0_pin_15	

2.3.4 Custom BSP Configurations

Typically, a BSP is not modified for BTSDK applications so sharing it is not an issue. However, the `COMPONENT_CUSTOM_DESIGN_MODUS` mechanism is available if the configuration must be changed for an application. There are two differences from typical ModusToolbox™ applications. First, the BSPs configuration is in a directory called *COMPONENT_bsp_design_modus*, so be aware of the case difference. Second, the BSP configuration is not included by the BSP itself, but instead is included in the application's *makefile*:

```
COMPONENTS +=bsp_design_modus
```

Therefore, in order to override the BSP configuration, you only need to change the `COMPONENTS` variable to remove the default configuration and add the custom configuration. Assuming an application has a custom configuration in *COMPONENT_CUSTOM_DESIGN_MODUS*, you would change the `COMPONENTS` variable in the *makefile* to this:

```
COMPONENTS +=CUSTOM_DESIGN_MODUS
```

2.4 Firmware Structure

As was described earlier, the application entry point is in a source file that is not typically called *main.c*. In the *Empty_BTSDK_App* code example, that source file is called *empty_wiced_bt.c*. In the templates used for this class, it is called *app.c*.

BTSDK MCUs use a real time operating system or RTOS (usually ThreadX) to manage Bluetooth® operation, so all applications run inside an RTOS. We will talk about the RTOS in more detail in a later chapter, but keep in mind that all of your applications will be running in RTOS threads. Here are the basics of how it works:

The user application source file begins with various `#include` lines depending on the resources used in your application. These header files can be found in the SDK under *wiced_btsdk/dev-kit/baselib/<device>/<version>/COMPONENT_<device>/include*.

After the includes list, you will find the `application_start` function (or `APPLICATION_START`), which is the main entry point into the application. That function typically does a minimal amount of initialization, starts the Bluetooth® stack and registers a stack callback function by calling `wiced_bt_stack_init`. The configuration parameters from *app_bt_cfg.c* are provided to the stack here. Once the `application_start` function has done its initialization and started the stack (which is an RTOS thread) it exits.

The stack callback function registered in the prior step is called by the stack whenever it has an event that the application might need to know about. It controls the rest of the application based on Bluetooth® events.

Most application initialization is done after the Bluetooth® stack has been enabled. You may even create additional threads at this point depending on what functionality your application requires.

The event that the Bluetooth® callback function receives to indicate that the stack has been enabled is `BTM_ENABLED_EVT`. The full list of events from the Bluetooth® stack is enumerated in the `wiced_bt_management_evt_t` typedef, which can be found in the file *wiced_btsdk/dev-kit/baselib/<device>/<version>/COMPONENT_<device>/include/wiced_bt_dev.h*.

A minimal C file for an application will look something like this:

```
#include "app_bt_cfg.h"
#include "sparcommon.h"
#include "wiced_bt_dev.h"
#include "wiced_bt_trace.h"
#include "wiced_bt_stack.h"
#include "wiced_platform.h"
#include "wiced_transport.h"
#include "wiced_rtos.h"
#include "cycfg.h"
#include "cycfg_gatt_db.h"

/* Prototype for Bluetooth stack management event callback */
wiced_bt_dev_status_t  app_bt_management_callback( wiced_bt_management_evt_t event,
wiced_bt_management_evt_data_t *p_event_data );

/* Entry point for user application */
void application_start(void)
{
     /* Start the Bluetooth stack */
     wiced_bt_stack_init( app_bt_management_callback, &wiced_bt_cfg_settings,
                          wiced_bt_cfg_buf_pools );
}

/* Define callback function for Bluetooth stack management events */
wiced_result_t app_bt_management_callback( wiced_bt_management_evt_t event,
                                           wiced_bt_management_evt_data_t *p_event_data )
{
    wiced_result_t status = WICED_BT_SUCCESS;
```

```
        switch( event )
        {
        case BTM_ENABLED_EVT:       /* Bluetooth Controller and Host Stack Enabled */

            if( WICED_BT_SUCCESS == p_event_data->enabled.status )
            {
               /* Initialize and start your application here once the Bluetooth stack is running */
            }
            break;

        default:
            break;
        }

        return status;
    }
```

2.5 Application README file

The *README.md* file in the *Empty_BTSDK_App* application contains a lot of useful information about application settings in the *makefile*, application structure, BTSDK features, software tools available for testing your Bluetooth® applications and more. It is worthwhile reviewing that file and using it as a reference as you progress in your learning.

2.6 Exercises

Exercise 1: Create and Explore the Empty_BTSDK_App

☐ 1. Use the ModusToolbox™ Project Creator to create the *Empty_BTSDK_App* code example.

Note: *You can use either the Project Creator in stand-alone mode or you can launch it from the Eclipse IDE for ModusToolbox™.*

☐ 2. Explore the application directory and the libraries that are downloaded into *mtb_shared/wiced_btsdk*.

☐ 3. Look at *empty_wiced_bt.c* to see what the application does.

Note: *The `wiced_transport_init` and `wiced_bt_stack_init` functions are called to configure the PUART to print debug messages using `WICED_BT_TRACE`. These functions and their configuration settings will be explained later in this class and in the level 3 Bluetooth® class. For now, just leave them as-is.*

☐ 4. Build the application and program it to your kit.

☐ 5. Open a serial terminal emulator window.

Note: *Remember that there are two UART interfaces from the kit. You want to open the PUART interface. If you look at the output during programming, you can determine which UART is used for programming. The PUART will be the other one of the two KitProg3 COM ports.*

Note: *The default BAUD rate for the PUART is 115200.*

☐ 6. Reset the kit and observe the messages printed to the serial terminal.

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

Published by
Infineon Technologies AG
81726 Munich, Germany

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

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

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

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

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

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

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