AIROC™ BTSTACK-v3.x application memory management

ModusToolbox™

About this document

Intended audience

AIROC™ Bluetooth® SDK (BTSDK) embedded developers using any of the following device families (all use BTSTACK versions 3.0 or greater):

AIROC™ CYW55572, AIROC™ CYW20829

Table of contents

1 Introduction

This document describes the heap and buffer memory management used by AIROC™ Bluetooth® SDK (BTSDK) applications and the upper layer stack of the Infineon AIROC™ Bluetooth® stack (BTSTACK). This applies only to devices that contain BTSTACK versions 3.0 or later.

In BTSTACK versions earlier than 3.0, buffer management is handled by the application differently; see the document 002-16403 – *AIROC™ Application Buffer Pools*.

Currently, only CYW55572 and CYW20829 based devices use BTSTACK versions 3.0 or later.

2 IoT resources

Infineon provides a wealth of data **https://www.infineon.com/cms/en/about-infineon/make-iot-work/iot-solutions/** to help you to select the right IoT device for your design and quickly and effectively integrate the device into your design. Infineon provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates. Customers can acquire technical documentation and software from the Infineon **Support Community** website.

3 Memory management

3.1 Memory ownership

A fundamental construct of the AIROC™ BTSTACK architecture is the concept of memory ownership. Application memory and stack memory are kept separate with clear ownership and handover.

On startup, the stack allocates the memory it needs for internal protocol management. The size of the allocated memory is based on the configuration provided by the application using the `wiced_bt_stack_init()` function.

To transmit data to a peer Bluetooth® device, a pointer to the memory containing that data is passed to the stack through a variety of BTSDK functions. This memory must not be altered in any way while being processed by the stack. When the transmission is complete (or fails), the stack calls back to inform the application that the memory is now free to be reused.

Similarly, when application data is received from a peer device, the stack stores it in one of its own internal memory buffers and calls up into the application with a pointer to that data. The data is expected to be processed immediately or copied into application buffers for later processing. The stack assumes it can then immediately reuse its memory buffer for the next incoming packet.

3.2 Application memory

Applications typically need static RAM or dynamically allocated RAM. Even though applications can use native OS functions to allocate and free the RAM, to enable portability across multiple operating systems, BTSDK provides utility functions to create and use memory from stack heaps and buffer pools. Buffer pools may be thought of as heaps that support a fixed-size allocation. This can be much more efficient for certain types of applications that transfer fixed-size blocks of data.

Applications are free to create multiple heaps and/or buffer pools. One of the application heaps should be designated as “default”.

Note: *Most AIROC™ libraries require applications to create a default heap.*

The following figure illustrates the separation of application and stack memory.

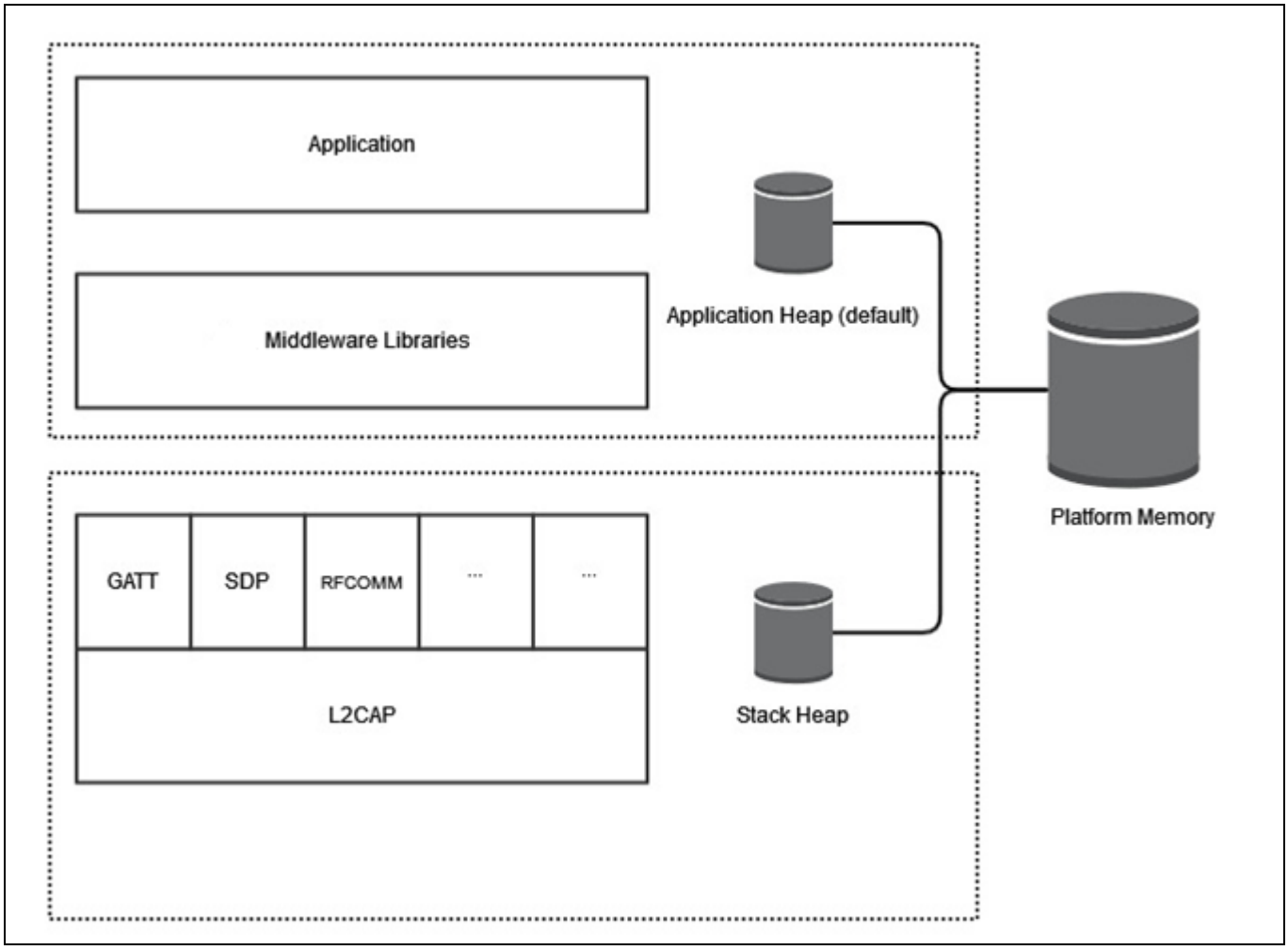

Figure 1 Memory management using separate heaps

Keep in mind the following memory-related concepts:

1. The stack allocates resources for various items such as control blocks for connections and data receive buffers (DRBs) for receiving the incoming asynchronous connection-less (ACL) data based on the configuration provided in the `wiced_bt_cfg_settings_t` structure that is passed to the stack in the `wiced_bt_stack_init()` function at application start.
2. Note the following on the memory for application data transfer:
   a) The application allocates the memory.
   b) The memory is returned back to the application typically through a transmit-complete callback after sending the data over the Host Controller Interface (HCI).
   c) The application can free and reallocate the data memory buffer returned.
3. The stack creates a 2000-byte data heap for its dynamic memory needs.
4. Stack data heap memory is used for the following:
   a) Queueing commands to the controller
   b) Data transfer during responses to be sent by the stack
   c) For data transfer related bookkeeping
5. Application memory for its data structures as defined by the application design and buffers for data transfer as defined by the BTStack functions are kept separate from the stack memory area.

6. Applications are expected to create a default heap of appropriate size, which can be used by the application code and helper libraries provided as part of BTSDK.
7. Applications can create private heaps/pools for their use.

4 Creating memory resources

Two types of memory resources can be created and used by applications based on need.

1. **wiced_bt_heap_t**: Variable sized allocator to obtain heaps. The application can allocate/free variable lengths of memory from the heap. The stack manages the allocations and frees of the memory to minimize fragmentation issues.
2. **wiced_bt_pool_t**: Fixed sized allocator to obtain buffer pools of fixed length chunks of memory. Buffer pools are more efficient than variable allocation heaps when an application knows it needs a certain number of buffers of fixed size.

BTSDK also provides utility functions for storing the memory allocated from stack heaps in buffer queues, and provides debug functions to show the memory usage.

Applications are free to create multiple heaps. Allocations from specific heaps and pools are done using the `wiced_bt_heap_t` or `wiced_bt_pool_t` pointers returned from calls to `wiced_bt_create_heap()` and `wiced_bt_create_pool()` functions respectively. Because allocations from each heap or pool take the created pointer as an argument, the allocation mechanism allows for easy accounting of the allocated memory.

Requirement	Function to call
Allocate memory from a heap	`wiced_bt_get_buffer_from_heap(wiced_bt_heap_t *, size)`
Allocate memory from a pool	`wiced_bt_get_buffer_from_pool(wiced_bt_pool_t *)`

4.1 General purpose heap allocation

BTSDK provides the `wiced_bt_get_buffer()` function that allows applications to get memory from the default heap created with the `wiced_bt_create_heap() function`. The `wiced_bt_get_buffer()` function is typically used like malloc, i.e., for general purpose allocations in an application.

Note: *AIROC™ libraries invoke calls to `wiced_bt_get_buffer()`. Therefore, the application must create one heap marked as default, even if the application does not require the default heap itself.*

5 Usage statistics

AIROC™ middleware libraries use memory from the default heap, typically for data transfer. As a guideline, the default heap should not be used for long-term memory requirements.

To set appropriate heap size, set the size of the default heap to more than the MTU (Maximum Transmission Unit) × number of outstanding packets with the stack.

Requirement	Function to call
See all heaps in use in the application	Call the `wiced_bt_get_heap_statistics_with_index()` function in a loop, and increment the index parameter for each iteration.
Get specific heap usage statisticsa	Call the `wiced_bt_get_heap_statistics()` function with the specific heap pointer as a parameter.
Get usage statistics of application pools	Call the `wiced_bt_get_pool_statistics()` function.
Get an estimate of the size of the stack required for a given configuration	Call the `wiced_bt_stack_get_dynamic_memory_size_for_config()` function with the `wiced_bt_cfg_settings_t` structure it intends to use in the `wiced_bt_stack_init()` call. This estimation function can be invoked before or after calling `wiced_bt_stack_init()`. The function call is reentrant and returns the memory required to support the config settings passed to the function. The current configuration is not altered by a call to this function.

Code Listing 1 Heap usage statistics

```
wiced_bt_heap_statistics_t stats;
int i = 0;

while ( wiced_bt_get_heap_statistics_with_index(i, &stats) == WICED_TRUE )
{
    WICED_BT_TRACE ("%d Heap Stats: size:%d max_single alloc:%d max_used:%d
fail_cnt:%d\n",
                    i,
                    stats.heap_size,
                    stats.max_single_allocation,
                    stats.max_heap_size_used);

    WICED_BT_TRACE ("current num_alloc:%d size_alloced:%d",
                    stats.current_num_allocations,
                    stats.current_size_allocated);

    WICED_BT_TRACE ("largest_free:%d num_free:%d free_size:%d",
                    stats.current_largest_free_size,
                    stats.current_num_free_fragments,
                    stats.current_num_free_fragments);
}
```

Code Listing 2 Declaration of wiced_bt_stack_get_dynamic_memory_size_for_config()

```
/**
* Returns the expected dynamic memory size required for the stack based on the
* p_bt_cfg_settings
*
* @param[in] p_bt_cfg_settings         : Bluetooth stack configuration
*
* @return    dynamic memory size requirements of the stack
*/
int32_t wiced_bt_stack_get_dynamic_memory_size_for_config(const
wiced_bt_cfg_settings_t* p_bt_cfg_settings);
```

Revision history

Document version	Date of release	Description of changes
**	2022-03-02	Initial release.

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

Edition 2022-03-02

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Go to www.infineon.com/support

Document reference

002-34884 Rev. **

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.