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

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

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





ModusToolbox™

WICED AMS Library

Document Number: 002-16782 Rev. *C

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

Contents

1 Introduction 3

2 IoT Resources 3

3 Design and Architecture 3

4 AMS Application and AMS Library 4

5 Library Reference 5

5.1 AMS Client Initialize 5
5.2 AMS Client Discover 5
5.3 AMS Discovery Complete Callback 5
5.4 AMS Client Discovery Result 6
5.5 AMS Client Discovery Complete 6
5.6 AMS Client Start 6
5.7 AMS Start Complete Callback 6
5.8 AMS Client Stop 7
5.9 AMS Stop Complete Callback 7
5.10 AMS Client Connection Up 7
5.11 AMS Client Connection Down 7
5.12 AMS Client Write Response 8
5.13 AMS Send Remote Command 8
5.14 AMS Client Process Notification 9

References 10

Document Revision History 11

Worldwide Sales and Design Support 12

1 Introduction

This document describes the functionality of the WICED AMS library used in various applications. The library can be used to exercise the functionality of the Apple Media Service [2] running on an iOS device. The document provides information on how the library can be accessed to discover characteristics of the AMS service, send commands to the iOS device to control the player, and register to receive notifications about media object changes. It is assumed that the reader is familiar with the Bluetooth Core Specification [1].

2 IoT Resources

Cypress provides a wealth of data at http://www.cypress.com/internet-things-iot to help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. Cypress 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 Cypress Support Community website (http://community.cypress.com/).

3 Design and Architecture

ModusToolbox™ provides an ‘ams’ sample application that utilizes the AMS library. If your installation of ModusToolbox or included SDKs do not contain the ‘ams’ sample application, it can be downloaded from GitHub – see https://github.com/cypresssemiconductorco/Code-Examples-BT-20819A1-1.0-for-ModusToolbox-1.1 or the appropriate Git repo there for your version of ModusToolbox and SDK. While applications themselves provide generic WICED application functionality, they use the WICED AMS Library to access the AMS service of the iOS device over the BLE GATT profile.

4 AMS Application and AMS Library

The ModusToolbox 'ams' sample application, together with the WICED AMS library, provides an implementation of the AMS client protocol to access the AMS service on an iOS device.

The application typically performs in a GAP peripheral role. At startup, it advertises as a peripheral device and allows an iOS device to connect. When a connection is established or dropped, the application shall call the `wiced_bt_ams_client_connection_up` or the `wiced_bt_ams_client_connection_down` functions to notify the library about the connection state change.

When a connection is established, the application performs a GATT discovery of the primary services of the connected device. If an AMS service is discovered, the application initializes the AMS library.

Note: The iOS 9.x version does not publish the AMS service to unpaired devices. To accommodate that problem, applications that use the AMS library must force iOS devices to perform pairing or set up encryption during every connection. To do so, the application sets the Device Name characteristic in the GATT database to require authentication. During each connection, the iOS device reads the device name. This initiates the secure connection setup.

When the application locates the AMS service on the iOS device, it asks the AMS library to perform the discovery of the characteristics and descriptors of the AMS service by calling the `wiced_bt_ams_client_discover` function. When the discovery is completed, the library executes the `wiced_bt_ams_discovery_complete_callback` function**,** passing the status of the discovery operation. During the discovery process, the application should pass the discovery result and discovery complete events to the library using the `wiced_bt_ams_client_discovery_result` and the `wiced_bt_ams_client_discovery_complete` function calls.

The application can instruct the library to start using the AMS service by calling the `wiced_bt_ams_client_start` function. At *client start*, the library registers with the iOS device to monitor the states of the different media objects. Similarly, the application can deregister from the AMS service on the iOS device by calling the `wiced_bt_ams_client_stop` function. The results of the registration and deregistration are passed back to the application using `wiced_bt_ams_start_complete_callback` and `wiced_bt_ams_stop_complete_callback`.

While the library is performing the discovery or registrations with the iOS device, the application should not send GATT requests to the iOS device on its own. Similarly, the application should not try to initialize two or more libraries at the same time.

The application can call the `wiced_bt_ams_send_remote_command` function to control the media player on the iOS device. Any changes to the state of the media player entities are sent back to the application using `wiced_bt_ams_notification_callback`. Using AMS notifications, the application receives information about changes to three sets of attributes:

- Player: The currently active media app. Attributes for this entity include values such as its name, playback state, and playback volume.
- Queue: The currently loaded playback queue. Attributes for this entity include values such as its size and its shuffle and repeat modes.
- Track: The currently loaded track. Attributes for this entity include values such as its artist, title, and duration.

Detailed information is available in [2].

5 Library Reference

5.1 AMS Client Initialize

The application should call this function to register application callbacks.

Prototype

```
void wiced_bt_ams_client_initialize (wiced_bt_ams_reg_t *p_reg)
```

Parameters

`p_reg` : Registration control block that includes AMS application callbacks.

Returns

None

5.2 AMS Client Discover

The application should call this function when it discovers that the connected central device contains the AMS service. The function starts the GATT discovery of AMS characteristics and descriptors.

After the application has started the discovery, and until it receives the discovery complete callback (see Section 5.3), the application shall pass the discovery results (see Section 5.4) and discovery complete events (see Section 5.5) that it receives from the stack to the library.

Prototype

```
wiced_bool_t wiced_bt_ams_client_discover (uint16_t conn_id, uint16_t s_handle,
uint16_t e_handle)
```

Parameters

`conn_id` : GATT connection ID.

`s_handle` : Start GATT handle of the AMS service.

`e_handle` : End GATT handle of the AMS service.

Returns

`WICED_TRUE` if the library started discovery successfully; `WICED_FALSE` otherwise.

5.3 AMS Discovery Complete Callback

This callback is executed when the AMS library completes the discovery of AMS service characteristics and descriptors.

Prototype

```
typedef void (*wiced_bt_ams_discovery_complete_callback_t) (uint16_t conn_id,
wiced_bool_t success)
```

Parameters

`conn_id` : GATT connection ID.

`status` : `WICED_TRUE` if the discovery has been completed successfully; `WICED_FALSE` otherwise.

Returns

None

5.4 AMS Client Discovery Result

While the library performs the GATT discovery, the application shall pass the discovery results received from the stack to the AMS Library. The library must locate three characteristics that belong to the AMS service including the remote control, the entity update, and the entity attribute. The second characteristic is the client configuration descriptor.

Prototype

```
void wiced_bt_ams_client_discovery_result (wiced_bt_gatt_discovery_result_t *p_data)
```

Parameters

`p_data` : Discovery result data as passed from the stack.

Returns
None

5.5 AMS Client Discovery Complete

While the library performs the GATT discovery, the application shall pass discovery complete callbacks to the AMS library. As the GATT discovery consists of multiple steps, this function initiates the next discovery request or writes a request to configure the AMS service on the iOS device.

Prototype

```
void wiced_bt_ams_client_discovery_complete(wiced_bt_gatt_discovery_complete_t *p_data)
```

Parameters

`p_data` : Discovery complete data as passed from the stack.

Returns

None

5.6 AMS Client Start

The application may call this function to start an AMS client. The discovery should be completed before calling this function. The start function configures the AMS server on the iOS device for notification and configuration information that the client wants to monitor.

Prototype

```
wiced_bool_t wiced_bt_ams_client_start (uint16_t conn_id)
```

Parameters

`conn_id` : GATT connection ID.

Returns

`WICED_TRUE` if the operation has been initiated successfully; `WICED_FALSE` otherwise.

5.7 AMS Start Complete Callback

This callback is executed when the AMS library completes a startup operation of the AMS service.

Prototype

```
typedef void (*wiced_bt_ams_start_complete_callback_t) (uint16_t conn_id, wiced_bool_t
success)
```

Parameters

`conn_id` : GATT connection ID.

`status` : `WICED_TRUE` if the operation has been completed successfully; `WICED_FALSE` otherwise.

Returns

None

5.8 AMS Client Stop

The application calls this function to deregister with the AMS client and stop receiving notifications.

Prototype

```
wiced_bool_t wiced_bt_ams_client_start (uint16_t conn_id)
```

Parameters

`conn_id` : GATT connection ID.

Returns

`WICED_TRUE` if the operation has been initiated successfully; `WICED_FALSE` otherwise.

5.9 AMS Stop Complete Callback

This callback is executed when the AMS library completes the deregistration with the AMS client.

Prototype

```
typedef void (*wiced_bt_ams_stop_complete_callback_t) (uint16_t conn_id, wiced_bool_t
success)
```

Parameters

`conn_id` : GATT connection ID.

`status` : `WICED_TRUE` if the operation has been completed successfully, `WICED_FALSE` otherwise.

Returns

None

5.10 AMS Client Connection Up

The application should call this function when a BLE connection with a peer device has been established.

Prototype

```
void wiced_bt_ams_client_connection_up (wiced_bt_gatt_connection_status_t *p_conn_status)
```

Parameters

`p_conn_status` : Pointer to a GATT connection status structure that includes the address and connection ID.

Returns

None

5.11 AMS Client Connection Down

The application should call this function when a BLE connection with a peer device has been disconnected.

Prototype

```
void wiced_bt_ams_client_connection_down (wiced_bt_gatt_connection_status_t
*p_conn_status)
```

Parameters

`p_conn_status` : Pointer to a GATT connection status structure that includes the address and connection ID.

Returns

None

5.12 AMS Client Write Response

The application should call this function when it receives a GATT Write Response for an attribute handle that belongs to the AMS service.

Prototype

```
void wiced_bt_ams_client_write_rsp (wiced_bt_gatt_operation_complete_t *p_data)
```

Parameters

`p_data` : Pointer to a GATT operation complete data structure.

Returns

None

5.13 AMS Send Remote Command

The application calls this function to send a remote control command to the connected iOS device. The discovery process should be completed prior to this call.

Prototype

```
wiced_bt_gatt_status_t wiced_bt_ams_send_remote_command (uint16_t conn_id, uint16_t
ams_command)
```

Parameters

`conn_id` : GATT connection ID.

`ams_command` : One of the following AMS commands:

```
AMS_REMOTE_COMMAND_ID_PLAY
AMS_REMOTE_COMMAND_ID_PAUSE
AMS_REMOTE_COMMAND_ID_TOGGLE_PLAY_PAUSE
AMS_REMOTE_COMMAND_ID_NEXT_TRACK
AMS_REMOTE_COMMAND_ID_PREVIOUS_TRACK
AMS_REMOTE_COMMAND_ID_VOLUME_UP
AMS_REMOTE_COMMAND_ID_VOLUME_DOWN
AMS_REMOTE_COMMAND_ID_ADVANCED_REPEAT_MODE
AMS_REMOTE_COMMAND_ID_ADVANCED_SHUFFLE_MODE
AMS_REMOTE_COMMAND_ID_SKIP_FORWARD
AMS_REMOTE_COMMAND_ID_SKIP_BACKWARD
```

Returns

Result of the GATT operation.

5.14 AMS Client Process Notification

The application processes GATT Notifications from the iOS device. The application can call this function to parse notifications received from the stack if the attribute handle belongs to the AMS service. This function verifies the data and, if successful, fills the event details in the provided AMS event object.

Prototype

```
wiced_bool_t wiced_bt_ams_client_process_notification (wiced_bt_gatt_operation_complete_t
*p_data, wiced_bt_ams_event_t *p_event)
```

Parameters

`p_data` : GATT notification as received from the iOS device.

`p_event` : Pointer to an AMS event structure to be filled.

Returns

`WICED_TRUE` if event is parsed successfully; `WICED_FALSE` otherwise.

References

[1] Bluetooth Core Specification, Version 4.2 (see Bluetooth Core Specification 4.2)

[2] Apple Media Service (AMS) Specification (see Apple Media Service)

Document Revision History

Document Title: WICED AMS Library

Document Number: 002-16782

Revision	ECN	Issue Date	Description of Change
**	5555181	08/11/2017	Initial release
*A	6333935	10/04/2018	Replaced “WICED Studio” with “ModusToolbox”. Updated Sales page.
*B	6486558	02/15/2019	Updated Design and Architecture Updated template
*C	6554890	04/23/2019	Removed Associated Part Family

Worldwide Sales and Design Support

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

Products

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

PSoC® Solutions

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

Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training| Components

Technical Support

cypress.com/support

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

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

© Cypress Semiconductor Corporation, 2017-2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, "Security Breach"). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. "High-Risk Device" means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. "Critical Component" means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress's published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

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