# User manual

## DA14580 Software Development Guide

### UM-B-003

**Abstract**

*This document provides basic guidelines for developers, in order to get familiar with DA14580 Software Development Kit and create the first BLE application based on it.*

# Contents



 

## Figures



## Tables

# 1 Terms and definitions

| | |
|---|---|
| SDK | Software Development Kit |
| ISR | Interrupt Service Routine |
| GAPM | Generic Access Profile Manager |

# 2 References

1. UM-B-015, DA14580 Software Architecture, Dialog Semiconductor
2. UM-B-006, DA14580 Sleep mode configuration, Dialog Semiconductor
3. UM-B-004, DA14580 Peripheral Drivers, Dialog Semiconductor
4. UM-B-011, DA14580 Memory Map-Scatter File, Dialog Semiconductor
5. Riviera Waves Kernel (RW-BT-KERNEL-SW-FS)
6. GAP Interface Specification (RW-BLE-GAP-IS)
7. ATTDB Interface Specification (RW-BLE-ATTDB-IS)

# 3 Introduction

Basic instructions and guidelines on how a developer can use the DA14580 Software Development Kit (SDK) to create an application based on it and add application specific functionality.

Document reader must have already read in UM-B-015 Software Architecture [1] where the DA14580 SDK organization and architecture are described in details.

# 4 Creating a new project

This section describes the steps needed for creating a new application project starting from the application template provided in DA14580 SDK.

## 4.1 Create an application project folder from template

1. In windows explorer, locate your DA14580 SDK distribution and open the folder: dk_apps\src\modules\app\src\app_project
2. Make a clone of template_fh.
3. Rename the newly created folder to e.g. my_application.
4. Open folder my_application. IMPORTANT NOTE: Do not alter the contents of “system “ folder.
5. Rename the file app_template_proj.c to e.g. app_myproject_proj.c
6. Rename the file app_template_proj.h to e.g. app_myproject_proj.h

## 4.2 Create a Keil project file folder from template

1. Open the folder: **dk_apps\keil_projects**
2. Make a clone of **template** folder. Rename the newly created folder to e.g. **myproject**.
3. Open folder **myproject**.
4. Rename folder **template_fh** to e.g. **myproject**
5. Open folder **myproject**.
6. Rename **fh_project_template.uvproj** (there will be only one of these files in the folder) to e.g. **my_project.uvproj.**

## 4.3 Editing your new project

Double-click **on my_project.uvproj** to open the Keil project.

### 4.3.1 Edit project directory tree

1. In Project Explorer, expand group app and right-click on the file **app_template_proj.c**. On the pop-up menu, click on Options for File ‘**app_template_proj.c**’. In tab Properties, edit the Path from **..\..\..\src\modules\app\src\app_project\template_fh\app_template_proj.c** to **..\..\..\src\modules\app\src\app_project\my_application\app_my_project_proj.c**.
2. Expand group arch and right-click on the file **periph_setup.c**. On the pop-up menu, click on Options for File ‘**periph_setup.c**’. In tab Properties, edit the Path from **..\..\..\src\modules\app\src\app_project\template_fh\common\periph_setup.c** to **..\..\..\src\modules\app\src\app_project\my_application\ common\periph_setup.c**.

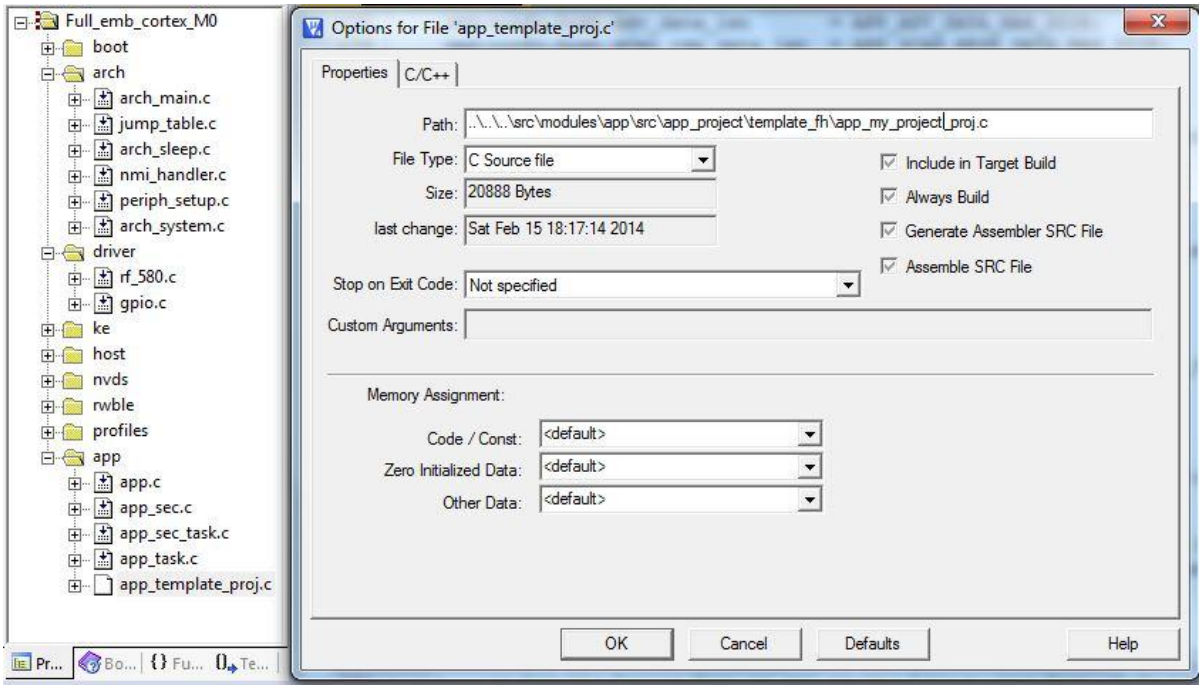


**Figure 1 – Edit project directory tree**

### 4.3.2 Edit project include path

1. In Project Explorer, select the root group “Full_emb_cortex_M0”.
2. On the main menu, click Project → Options for Target ‘Full_emb_cortex_M0’ and select the tab “C/C++”.
3. Double-click on the line: **.\..\..\..\src\modules\app\src\app_project\template_fh**
   and edit it to reflect the path of your application folder, e.g. :
   **.\..\..\..\src\modules\app\src\app_project\my_application**
4. Double-click on the line: **.\..\..\..\src\modules\app\src\app_project\template_fh\system**
   and edit it to reflect the path of the common folder in your application folder, e.g.:
   **.\..\..\..\src\modules\app\src\app_project\ my_application\system**

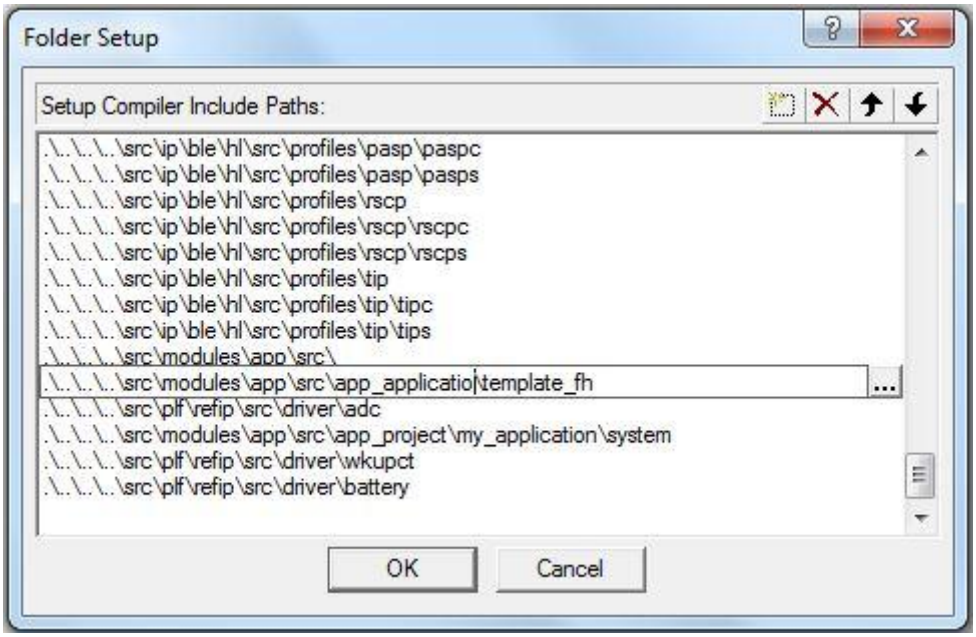


**Figure 2 – Edit include paths**

## 4.4 Edit configuration of the project

1. Open **da14580_config.h** file and replace the following line to declare the specific application:

`#define CFG_APP_TEMPLATE with e.g. #define CFG_APP_MYPROJECT.`

2. Open **rwip_conifg.h** file and add the following lines to convert the configuration directive into a zero/one switch:

```
#if defined(CFG_APP_MYPROJECT)
#define BLE_APP_MYPROJECT   1
#else // defined(CFG_APP_MYPROJECT)
#define BLE_APP_MYPROJECT   0
#endif // defined(CFG_APP_MYPROJECT)
```

3. Open **app_api.h** file and add the following lines to export types and declaration of the specific project to the BLE common application code:

```
#if (BLE_APP_ MYPROJECT)
#include "app_my_project_proj.h"
#endif
```

4. Open **app_my_project_proj.c** file and replace line to replace old header file with the new one:

`#include "app_template_proj.h" with #include "app_my_project_proj.h".`

# 5 Project Oriented Functionality – User Defined

A list of “callback” functions is defined in **file app_my_project_proj.c.** These “callback” functions are referenced by BLE application common code (**files app.c, app_task.c, app_sec.c, app_sec_task.c**) and they cannot be removed. User/developer must enter application specific code in the following “callback” functions defined **in file app_my_project_proj.c.**

**Table 1: Common app api functions.**

| **APP CallBack Functions** | |
|---|---|
| **Initialization** | |
| app_init_func | Project actions in app_init |
| app_sec_init_func | Project actions in app_sec_init during system initialization |

| | |
|---|---|
| app_db_init_func | Project actions for profiles database initialization |
| app_db_init_complete_func | Handles completion of databases creation. i.e. |
| **Device configuration** | |
| app_set_dev_config_complete_func | Called upon device configuration completion |
| app_param_update_func | Sends request to update connection parameters. |
| app_configuration_func | Project configures GAPM. Called upon reset completion |
| app_update_params_complete_func | Called upon connection parameters update completion |
| app_update_params_rejected_func | Called upon connection parameters update rejection |
| **Advertise** | |
| app_adv_func | Setup advertise string |
| app_adv_direct_complete | Handles direct advertising completion |
| app_adv_undirect_complete | Handles undirect advertising completion |
| **Connection** | |
| app_connection_func | Project actions in app_connect (device connection) |
| app_disconnect_func | Project actions in app_disconnect |
| **Security** | |
| app_sec_encrypt_ind_func | Handles encryption indication |
| app_paired_func | Project action when device is paired |
| app_send_pairing_rsp_func | Sends pairing response message Called upon pairing request message reception. |
| app_sec_encrypt_complete_func | Project actions when encryption request is completed successfully |
| app_sec_encrypt_ind_func | Handles encryption indication |
| app_validate_encrypt_req_func | Validates encryption request message |
| app_sec_encrypt_complete_func | Project actions when encryption request is completed successfully |
| app_mitm_passcode_entry_func | Start passcode entry process. Called in gapc_bond_req_ind_handler(tk_type == GAP_TK_KEY_ENTRY) |
| app_send_tk_exch_func | Send GAPC_TK_EXCH. Called in gapc_bond_req_ind_handler(tk_type == GAP_TK_KEY_DISPLAY) |
| app_send_irk_exch_func | Send GAPC_IRK_EXCH. Called in gapc_bond_req_ind_handler/GAPC_IRK_EXCH |
| app_send_csrk_exch_func | Send GAPC_CSRK_EXCH. Called in gapc_bond_req_ind_handler/GAPC_CSRK_EXCH |
| app_send_ltk_exch_func | Send GAPC_LTK_EXCH. Called in gapc_bond_req_ind_handler/GAPC_LTK_EXCH |

# 6 Addition of existing application code for profiles

In this section, application code initializing and controlling Device Information Service Server Role (DISS) profile will be added to the application.

## 6.1 Enable the profile

Open file da14580_config.h and change

```
#define nCFG_PRF_DISS       1
```

into

```
#define CFG_PRF_DISS        1
```

This will define **BLE_DIS_SERVER** identifier as *1 and* includes DISS profile source code files in project.

## 6.2 Add application profile source files to the project

In Project Explorer, right click on group app and select Add Existing Files to Group 'app'. Navigate to the path: **dk_apps\src\modules\app\src\app_profiles\diss,** and add the files: **app_diss.c** and **app_diss_task.c.**

## 6.3 Add the newly referenced header files path to the project include paths:

In Project Explorer, select the root group "Full_emb_cortex_M0".

On the main menu,

1. Click Project → Options for Target 'Full_emb_cortex_M0' … and select the tab "C/C++".
2. Click on the (…) button, next to the Include Paths Edit box. Click on the 'Add' icon and add the following path: **.\..\..\..\src\modules\app\src\app_profiles\diss**

## 6.4 Add the profile header files to the project

In the same file (**app_myproject_proj.c**), go to the top and right-click on the filename at the #include "app_myproject_proj.h" line. Click on 'Open Document "**app_myproject_proj.h**" '. Under the "PROFHEADER" user-edit tag, type the following directives:

```
#if (BLE_DIS_SERVER)
#include "app_dis.h"
#include "app_dis_task.h"
#endif
```

Note: For this example, the header files exist in filesystem.

## 6.5 Create the profile database and enable profile

In group app, open file **app_myproject_proj.c**. In function **app_db_init_func()** and **"switch (app_env.next_prf_init)"** the following code must be added to include the database creation of the profile:

```
#if (BLE_DIS_SERVER)
case (APP_DIS_TASK):
       {
              app_dis_create_db_send();
} break;
#endif //BLE_DIS_SERVER
```

In the same file, in function app_connection_func() the following code must be added to, in order to enable profile when application gets connected:

```
#if (BLE_DIS_SERVER)
    app_dis_enable_prf(app_env.conhdl);
#endif
```

### 6.6 Application verification

At this point you can build your application. Run your application with Keil debugger. With the help of any BLE application perform a scan to discover your device advertising. Connect to your device and discover all services and characteristics. Read values of the characteristics. You must read the following values:

Manufacturer Name: “Dialog Semi”
Model Number String: “DA14580”
System ID: {0x12, 0x34, 0x56, 0xFF, 0xFE, 0x9A, 0xBC, 0xDE} (Hex value)
Software Revision: <SDK Release Version>

## 7 Peripheral drivers utilization

In this section, usage of device peripheral modules drivers is described. The example of an application using SPI interface to access an external SPI flash module will be used.

### 7.1 Configure peripherals

1. Definitions in **src\modules\app\src\app_project\my_application\system\periph_setup.h**: Here various parameters that refer to the operation of the peripherals can be defined (e.g. the desired configuration of the peripherals, the size of the external memory modules). For example, in case of SPI flash, the following definitions must be added:

```
#define          SPI_FLASH_SIZE             32768
#define          SPI_FLASH_PAGE_SIZE        256
const SPI_Pad_t cs_pad = { GPIO_PORT_1, GPIO_PIN_0};
```

2. Peripheral configuration and initialization in **src\modules\app\src\app_project\my_application\system\periph_setup.c**: **GPIO_reservations()** function. Here, the globally reserved GPIOs reservation takes place, allowing the assignment of each pin to exactly one peripheral. **GPIO_Reservations()** function is active only in development mode (**DEVELOPMENT__NO_OTP** is enabled). For SPI interface the following reservations must be done:

```
RESERVE_GPIO( SPI_EN, GPIO_PORT_1, GPIO_PIN_0, PID_SPI_EN);
RESERVE_GPIO( SPI_CLK, GPIO_PORT_0, GPIO_PIN_4, PID_SPI_CLK);
RESERVE_GPIO( SPI_DO, GPIO_PORT_0, GPIO_PIN_6, PID_SPI_DO);
RESERVE_GPIO( SPI_DI, GPIO_PORT_0, GPIO_PIN_7, PID_SPI_DI);
```

3. **set_pad_functions()** function: Device port pins are configured and assigned to peripheral modules. **GPIO_ConfigurePin()** function must be used to set port function. Configuration of previously reserved ports for SPI interface is as follow:

```
GPIO_ConfigurePin( GPIO_PORT_1, GPIO_PIN_0, OUTPUT, PID_SPI_EN, true );
GPIO_ConfigurePin( GPIO_PORT_0, GPIO_PIN_4, OUTPUT, PID_SPI_CLK, false );
GPIO_ConfigurePin( GPIO_PORT_0, GPIO_PIN_6, OUTPUT, PID_SPI_DO, false );
GPIO_ConfigurePin( GPIO_PORT_0, GPIO_PIN_7, INPUT, PID_SPI_DI, false );
```

4. **periph_init()** function: Peripheral drivers initialization functions must be called here. For SPI flash driver example, the following lines must be used to initialize SPI flash and SPI drivers.

```
spi_flash_init(SPI_FLASH_SIZE, SPI_FLASH_PAGE);
spi_init(&spi_FLASH_CS_Pad, SPI_MODE_8BIT,    SPI_ROLE_MASTER,
SPI_CLK_IDLE_POL_LOW,SPI_PHA_MODE_0,SPI_MINT_DISABLE,SPI_XTAL_DIV_8);
```

### 7.2 Add peripheral drivers in project.

DA14580 SDK distribution includes a set of peripheral modules drivers. All drivers source code files reside in folder **dk_apps\src\plf\refip\src\driver**. Driver files must be added in project driver group. If added driver is using a lower layer driver then these files must be added in drivers group as well.

In Order to add external SPI Flash device driver, right click on group app, in Project Explorer, and select Add Existing Files to Group ‘driver’. Navigate to the path: **dk_apps\src\plf\refip\src\driver\spi_flash** and add **spi_flash.c** file. SPI flash driver uses SPI interface driver and SPI drivers files must be added in the same group. Navigate to the path: **dk_apps\src\plf\refip\src\driver\spi** and add **spi.c** file.

Both driver folders must be added in project include path:

On the main menu, click Project → Options for Target ‘Full_emb_cortex_M0’ and select the tab “C/C++”. Add the following paths:

**.\..\..\..\src\ plf\refip\src\driver\spi_flash**

**.\..\..\..\src\ plf\refip\src\driver\spi**

### 7.3 Initialize peripherals

**periph_init()** function initializes device peripheral modules. If application uses extended or deep sleep mode it is called in BLE_WAKEUP_LP ISR. BLE_WAKEUP_LP irq is generated by BLE core when it exits sleep mode. Peripheral module register status is not retained during sleep mode period, hence, each peripheral module used by application must be re-initialized in **periph_setup()** function. For SPI flash example the following line must be added:

```
spi_flash_init(SPI_FLASH_SIZE, SPI_FLASH_PAGE_SIZE, &cs_pad);
```

### 7.4 Further reading

UM-B-004 Peripherals Drivers [3].

## 8 Project Configuration

### 8.1 Configuration Directives

All DA14580 SDK projects pre-include a configuration header file (da14580_config.h) residing in Keil project directory. Directives defined in da14580_config.h modify various settings of the application.

**Table 2: Project configuration**

| Directive | Defined | Undefined |
|---|---|---|
| CFG_APP | Integrated host application | External processor host application |
| CFG_PRF_<profile> | Profile included | Profile not included |
| CFG_APP_<application> | Application identifier. Must be defined for all integrated host applications. | |
| CFG_NVDS | Non Volatile Data Storage (NVDS) structure used (Appendix A) | NVDS structure not used |
| CFG_APP_SEC | Includes BLE security | Excludes BLE security |
| CFG_LUT_PATCH | Performs the calibration of the Voltage Controlled Oscillator of the radio PLL.<br>It must **not** be altered by the customer. | Calibration disabled |
| CFG_WDOG | Watchdog timer enabled | Watchdog timer disabled |
| CFG_EXT_SLEEP<br>CFG_DEEP_SLEEP | Default sleep mode. Only one must be defined | |
| BLE_CONNECTION_MAX_USER | Max connections number (1-6) | |

| Directive | Defined | Undefined |
|---|---|---|
| DEVELOPMENT__NO_OTP | Development mode, OTP copy at system wakeup is disabled. | In the production, if the product loads the code from OTP. |
| CFG_LP_CLK | Low power clock selection (XTAL32 or RCX) | |
| REINIT_DESCRIPT_BUF | Memory Map/Scatter File configuration. | |
| USE_MEMORY_MAP | | |
| DB_HEAP_SZ | | |
| ENV_HEAP_SZ | | |
| MSG_HEAP_SZ | | |
| NON_RET_HEAP_SZ | | |
| CFG_CALIBRATED_AT_FAB | Calibration values written in OTP Header | Un-calibrated device. |

Projects in DA14580 SDK use two additional configuration header files:

**da14580_scatter_config.h**: Scatter file and memory map configuration.

**da14580_stack_config.h**: BLE stack and kernel definitions.

However these files must not be altered by developer.

### 8.2 Further reading

UM-B-006 Sleep Architecture [2], UM-B-011 Memory Map-Scatter File [4]

# 9 Using sleep API

## 9.1 Sleep mode API functions

- void **app_disable_sleep()**: Disables all sleep modes.
- void **app_set_extended_sleep()**: Activates the extended sleep mode.
- void **app_set_deep_sleep()**: Activates the deep sleep mode.
- uint8_t **app_get_sleep_mode()**: Returns the current mode of operation (sleep disabled, extended sleep, deep sleep).
- void **app_force_active_mode()**: Disables sleep. Stores the sleep mode used by the application.
- void **app_restore_sleep_mode()**: Restores the selected mode of operation, if active mode is not requested.
- void **app_ble_ext_wakeup_on()**: Sets system in continuous sleep, waiting a an external event to force wakeup.
- void **app_ble_ext_wakeup_off()**: Restores operation mode to normal.
- Bool **app_ble_ext_wakeup_get()**: Returns current mode. True if system sleeps forever waiting for a forced wakeup

## 9.2 App sleep hooks

In src\modules\app\src\app_project\my_application\system\app_sleep.h sleep software hooks are defined. Developer can add application specific code in these functions if required. Bellow a brief description of these functions usage can be found.

- HOOK1 - **app_asynch_trm():** Used for sending messages to kernel tasks generated from asynchronous events have been processed in HOOK2 - app_asynch_proc().
- HOOK2 - **app_asynch_proc():** Used for processing of asynchronous events at “user” level. The corresponding ISRs should be kept as short as possible and the remaining processing should be done at this point.
- HOOK3 - **app_asynch_sleep_proc():** Used for updating the state of the application just before sleep checking starts.
- HOOK4 - **app_sleep_prepare_proc():** Used to disallow extended or deep sleep based on the current application state. BLE and Radio are still powered off.
- HOOK5 - **app_sleep_entry_proc():** Used for application specific tasks just before entering the low power mode.
- HOOK6 - **app_sleep_exit_proc():** Used for application specific tasks immediately after exiting the low power mode.

### 9.3 Further reading

UM-B-006 Sleep Architecture [2]

## 10 Create a new profile

In this section, the development process of a non-existing profile in DA14580 SDK is described.

A sample profile with 128 bit UUID services and attributes (sample128), will be used as a reference. sample128 profile consists of one 128 bit UUID service (sample128), including one characteristic (sample128_1_char) with read/write properties and one characteristic (sample128_1_char) with read/notify properties and a client configuration attribute. Maximum data size for both characteristic is set to 1 and data size of configuration attribute is 2.

Profile sends an indication with the value of sample128_1_char to application task on every successful write request from a remote device.

A complete implementation of a proprietary profile in SDK can be found in SPOTA Reporter standalone application, where Software Programming Over The Air Reporter is included.

Keil project file of SPOTA Reporter application can be found in **dk_apps\keil_projects\spota\spotar_fh** directory.

### 10.1 Project and Source files

The application developer needs to add source and header files in project “profiles” group.

For sample128 profile the following files added in **dk_apps/src/bleip/src/profiles/sample128** directory:

- sample128.c
- sample128.h
- sample128_task.c
- sample128_task.h

To enable profile, **CFG_PRF_SAMPLE128** directive must be defined in **da14580_config.h**.

### 10.2 API Messages and handlers

In **sample128_task.c** profile state handler table and default state handler is defined, similarly to previously described application task.

In sample128 profile, handlers are defined in **sample128_task.c** file:

```
const struct ke_state_handler sample128_state_handler[SAMPLE128_STATE_MAX] =
{
```

```
[SAMPLE128_DISABLED]=KE_STATE_HANDLER(sample128_disabled),
[SAMPLE128_IDLE] = KE_STATE_HANDLER(sample128_idle),
[SAMPLE128_CONNECTED]=KE_STATE_HANDLER(sample128_connected),
};

const struct ke_state_handler sample128_default_handler =
KE_STATE_HANDLER(sample128_default_state);
```

Profile tasks API messages must be declared and corresponding message handlers for incoming messages must be implemented and declared in task state handlers.

Sample128 API messages are declared in **sample128_task.h** file:

```
enum
{
    /// Start sample128. Device connection
    SAMPLE128_ENABLE_REQ = KE_FIRST_MSG(TASK_SAMPLE128),
    /// Disable confirm.
    SAMPLE128_DISABLE_IND,
    /// Att Value change indication
    SAMPLE128_VAL_IND,
    ///Create DataBase
    SAMPLE128_CREATE_DB_REQ,
    ///Inform APP of database creation status
    SAMPLE128_CREATE_DB_CFM,
    /// Update second characteristics value request
    SAMPLE128_UPD_CHAR2_REQ,
    /// Update second characteristics value confiramtion
    SAMPLE128_UPD_CHAR2_CFM,
    /// Error Indication
    SAMPLE128_ERROR_IND,
};
```

Handler functions are implemented and declared in the corresponding incoming messages in state handlers table in **sample128_task.c**:

```
const struct ke_msg_handler sample128_disabled[] =
{
{SAMPLE128_CREATE_DB_REQ, (ke_msg_func_t) sample128_create_db_req_handler },
};
const struct ke_msg_handler sample128_idle[] =
{
{SAMPLE128_ENABLE_REQ, (ke_msg_func_t) sample128_enable_req_handler},
};
const struct ke_msg_handler sample128_connected[] =
{
{GATTC_WRITE_CMD_IND,       (ke_msg_func_t) gattc_write_cmd_ind_handler},
{SAMPLE128_UPD_CHAR2_REQ,   (ke_msg_func_t) sample128_upd_char2_req_handler},
};
```

## 10.3 Creation of profile task

Profile task type must be added in task types enumeration.

In **rwip_config.h** file:

```
TASK_SAMPLE128    = 64  ,   // Sample128 Task
…
```

A task descriptor must be defined in **sample128.c**:

```
static const struct ke_task_desc TASK_DESC_SAMPLE128 =
{sample128_state_handler,
```

```
&sample128_default_handler, sample128_state, SAMPLE128_STATE_MAX,
SAMPLE128_IDX_MAX};
```

In **sample128_init()** function, task must be created:

```
ke_task_create(TASK_SAMPLE128, &TASK_DESC_SAMPLE128);
```

Finally a call of **sample128_init()** must be added in **prf_init_func()** in **prf_utils.c** file:

```
#if (BLE_SAMPLE128)
sample128_init();
#endif // (BLE_SAMPLE128)
```

## 10.4 Add services and attributes in database

Profile services and attributes must be added in BLE stacks database. Sample128 example provides an API message to application task for the initiation of the procedure. When message is received the corresponding handler runs and creates profile database.

At first the service must be added in database:

```
nb_att_16 = 4;
nb_att_32 = 0;
nb_att_128 = 2;
status = attmdb_add_service(&(sample128_env.sample128_1_shdl),
TASK_SAMPLE128,nb_att_16, nb_att_32, nb_att_128, 58); //16 + (2*19) + 1 + 1 + 2
```

Total number of attributes, including service attribute, number of attributes with 128 bit and sum of attributes maximum data sizes are provided to **attsdb_add_service()** for memory allocation. Total data size of 58 for sample128 service is broken down to the following sizes:

- 16 – Size of service UUID (128 bit)
- 2 * 19 – Size of two characteristics: 128 bit UUIDs + properties size (1 byte) + value attribute size (2 bytes)
- 2 * 1 – Maximum data size of two value attributes.
- 2 – Client Configuration attribute data size.

Handle number of service is returned to **sample128_env.sample128_1_shdl**.

Service attribute is not created by previous function call and must be added in database:

```
status = attmdb_add_attribute(sample128_env.sample128_1_shdl, ATT_UUID_128_LEN,
ATT_UUID_16_LEN, (uint8_t*)&att_decl_svc, PERM(RD, ENABLE),
&(sample128_env.sample128_1_shdl));
```

UUID size is 2 bytes (16 bit) attribute data size is 16 bit.

Handle number in **sample128_env.sample128_1_shdl** will remain unchanged.

Service attribute value must be set to service UUID:

```
const struct att_uuid_128 sample128_1_svc = {{0xf0, 0x28, 0xe3, 0x68, 0x62,
0xd6, 0x34, 0x90, 0x51, 0x43, 0xef, 0xaa, 0xc6, 0x4c, 0x2f, 0xbc}};
status = attmdb_att_set_value(sample128_env.sample128_1_shdl, ATT_UUID_128_LEN,
(uint8_t *)sample128_1_svc.uuid);
```

First Characteristic and value attributes must be added in database.

```
const struct att_uuid_128 sample128_1_val     = {{0x10, 0x11, 0x12, 0x13, 0x14,
0x15, 0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F}};

status = attmdb_add_attribute(sample128_env.sample128_shdl, ATT_UUID_128_LEN +
3, ATT_UUID_16_LEN, (uint8_t*) &att_decl_char, PERM(RD, ENABLE), &(char_hdl));

status = attmdb_add_attribute(sample128_env.sample128_shdl, sizeof(uint8_t),
ATT_UUID_128_LEN, (uint8_t*)&sample128_1_val.uuid, PERM(RD, ENABLE) | PERM(WR,
ENABLE), &(val_hdl));
```

Corresponding handles will be returned in **char_hdl** and **val_hdl** variables.

Value attribute handle is copied to descriptor of first Characteristic and set to Characteristic value.

```
struct att_char128_desc sample128_1_char = {ATT_CHAR_PROP_RD |
ATT_CHAR_PROP_WR, {0,0},{0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18,
0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E,0x1F}};
memcpy(sample128_1_char.attr_hdl, &val_hdl, ATT_UUID_16_LEN);
status = attmdb_att_set_value(sample128_1_char_hdl, sizeof(sample128_1_char),
(uint8_t *)&sample128_1_char);
```

Second Characteristic and value attributes must be added in database.

```
const struct att_uuid_128 sample128_2_val     = {{0x20, 0x21, 0x22, 0x23, 0x24,
0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F}};
status = attmdb_add_attribute(sample128_env.sample128_shdl, ATT_UUID_128_LEN +
3, ATT_UUID_16_LEN, (uint8_t*) &att_decl_char, PERM(RD, ENABLE), &(char_hdl));
status = attmdb_add_attribute(sample128_env.sample128_shdl, sizeof(uint8_t),
ATT_UUID_128_LEN, (uint8_t*)&sample128_2_val.uuid, PERM(RD, ENABLE) | PERM(NTF,
ENABLE), &(val_hdl));
```

Value attribute handle is copied to descriptor of second Characteristic and set to Characteristic value.

```
struct att_char128_desc sample128_2_char = {ATT_CHAR_PROP_RD |
ATT_CHAR_PROP_NTF, {0,0}, {0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29,
0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F}};
memcpy(sample128_2_char.attr_hdl, &val_hdl, sizeof(uint16_t));
status = attmdb_att_set_value(char_hdl, sizeof(sample128_2_char), (uint8_t
*)&sample128_2_char);
```

Finally client configuartion attribute of second characteristic must be added in database:

```
status = attmdb_add_attribute(sample128_env.sample128_shdl, sizeof(uint16_t),
ATT_UUID_16_LEN, (uint8_t*) &att_decl_cfg, PERM(RD, ENABLE) | PERM(WR, ENABLE),
&(val_hdl));
```

## 10.5 Send Notification for second characteristic

**sample128_2_char** characteristic of **sample128** service has notification property. Notification behaviour is determined by client configuration value. If a client has configured notification on the second characteristic and value is changed by host application then client shall be notified. Notification behaviour state is stored in **feature** parameter of **sample_128_env**.

**sample128** profile provides **SAMPLE128_UPD_CHAR2_REQ** message to host application for changing the value of **sample128_2_char**. Message is handled by **sample128_upd_char2_req_handler()** function . Current notification behaviour is checked in handler function and if it is enabled then a notification message for **sample128_2_char** value is sent to client:

```
prf_server_send_event((prf_env_struct *)&sample128_env, false,
sample128_env.sample128_shdl + SAMPLE128_2_IDX_VAL);
```

## 10.6 Further reading

Riviera Waves Kernel (RW-BT-KERNEL-SW-FS) [5], ATTDB Interface Specification (RW-BLE-ATTDB-IS) [7].

# 11 Developing application layer profile code

DA14580 SDK distribution includes a set of application sample code files, interacting with specific profiles (e.g. proximity reporter, DISS etc.). Application source code files related to profiles are organized in different folders under **dk_apps\src\modules\app\src\app_profiles** folder.

In this section the process to develop new application code related to a profile is described. Development and addition of application code to initialize and control sample128 profile will be used as an example.

### 11.1 Create folder and files

In folder dk_apps\src\modules\app\src\app_project, create a subdirectory and name it sample128. Create the following files in this folder:

- app_sample128.c
- app_sample128.h
- app_sample128_task.c
- app_sample128_task.h

Application existing code files, related to other profiles (e.g. proximity reporter, DISS etc.) can be used as reference.

### 11.2 Sending messages to profile

Functions to build and send messages to profile task must be defined in **app_sample128.c** file. Sample128 profile expects to receive from application **SAMPLE128_CREATE_DB_REQ** and **SAMPLE128_ENABLE_REQ** messages.

### 11.3 Message handlers

A message handler function for each message received from sample128 profile must be defined in **app_sample128_task.c** file.

All message handlers with the corresponding message type must be added in application table of default handlers, in file **app_task_handlers.h**. Hence the following lines must be added at the end of **const struct ke_msg_handler app_default_state[]**, array.

```
#if (BLE_SAMPLE128)
//sample128 database creation confirmation message
 {SAMPLE128_CREATE_DB_CFM,  (ke_msg_func_t)sample128_create_db_cfm_handler},
//sample128 disabled indication
 {SAMPLE128_DISABLE_IND,    (ke_msg_func_t)sample128_disable_ind_handler},
//sample128 attribute value change by peer device Indication
 {SAMPLE128_VAL_IND,        (ke_msg_func_t)sample128_val_ind_handler},
#endif
```

In functions **sample128_disable_ind_handler()** and **sample128_val_ind_handler()** user defined code should be added, while **sample128_create_db_cfm_handler()** must build and send an **APP_MODULE_INIT_CMP_EVT** to application task, to allow the continuation of profiles databases creation from application.

### 11.4 Adding code in project

Created code can be added in the project by following the process described for DISS profile application code in paragraph 2.5.

## 12 Application initialization

Application task must follow a specific sequence of actions to initialise and configure BLE stack in order to start operating as BLE device. A brief description of the messages exchanged between application task and stack tasks and actions of application for operating in peripheral and central role, are outlined in this chapter.

## 12.1 Application and Stack initialization

Application initialization runs in common function app_init(). app_init calls api function app_init_func(). No interaction with stack tasks can take place here, since there is no indication that BLE stack is ready to receive messages. Application environment initialization should run here i.e. global variables initialisation, peripheral devices etc.

A GAPM_DEVICE_READY_IND is sent to application task when BLE stack is ready. The message is handled in common code function gapm_device_ready_ind_handler(). Application must send a GAPM_RESET_CMD command. GAPM BLE stack task sends a GAPM_CMP_EVT to confirm GAPM_RESET_CMD. This is handled in common application code function gapm_cmp_evt_handler(). Application must configure BLE stack by sending GAPM_SET_DEV_CONFIG message. One of the parameters configured in this message is device role. GAPM task responds with a new GAPM_CMP_EVT message. Application can now initialize the server role profiles databases if there are any. Most of the Database initialization procedure is implemented in app_db_init() function. In the following diagram the Database initialization sequence of two example profiles (Proximity Reporter and Battery Service Server) is outlined.

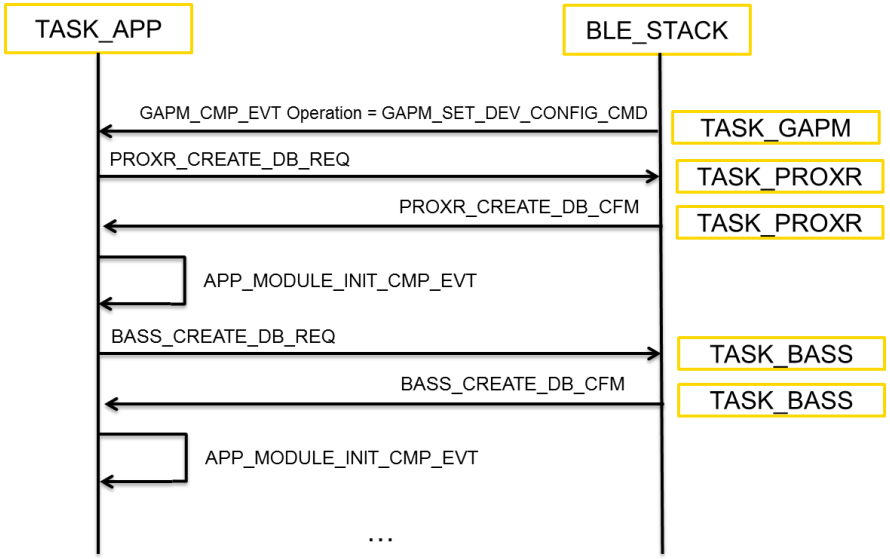


**Figure 3 – Database initialization sequence**

Application sends a database initialization to each supported profile. At reception of confirmation message it sends to itself a APP_MODULE_INIT_CMP_EVT. This is handled in app_module_init_cmp_evt_handler() and a database initialization message is send to next profile until all supported profiles are initialized.

Next action depends on device role. In case of peripheral role device should start advertising. GAPM_START_ADVERTISE_CMD must be sent to GAPM task.

A device operating in central role should initiate a scanning procedure, by sending a GAPM_START_SCAN_CMD to GAPM task. Application must handle GAPM_ADV_REPORT_IND message to know the discovered peripheral devices.

Initialization sequence for peripheral and central role is outlined in the next two diagrams.

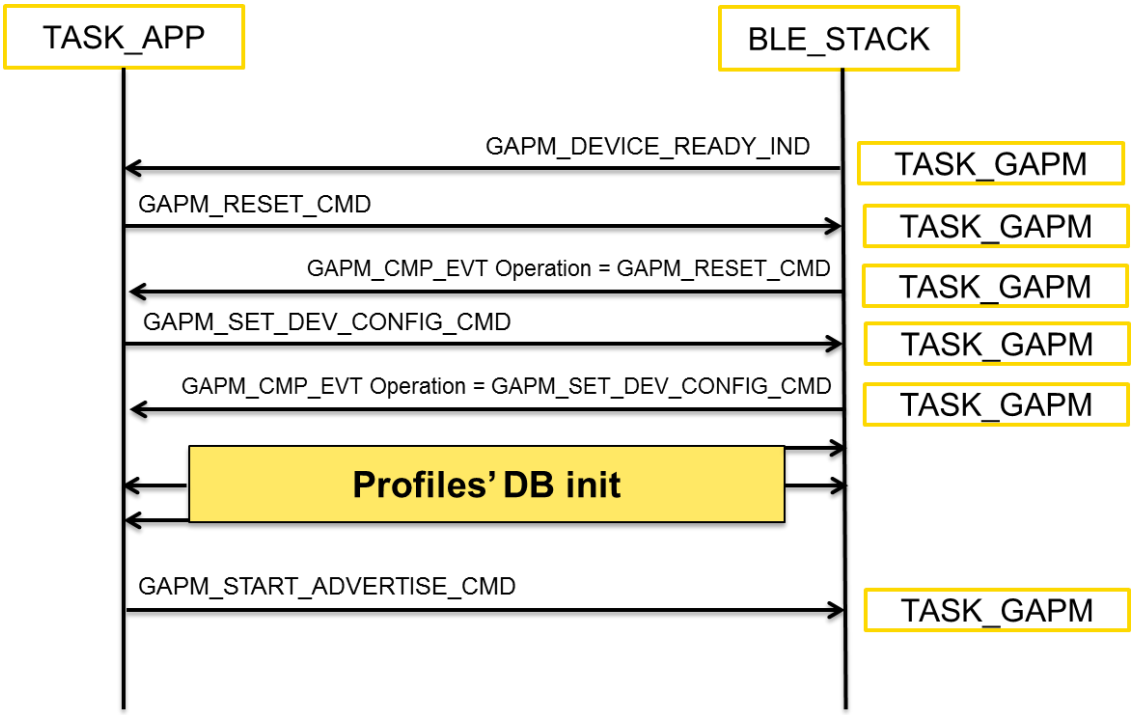


**Figure 4 – Peripheral device initialization sequence**

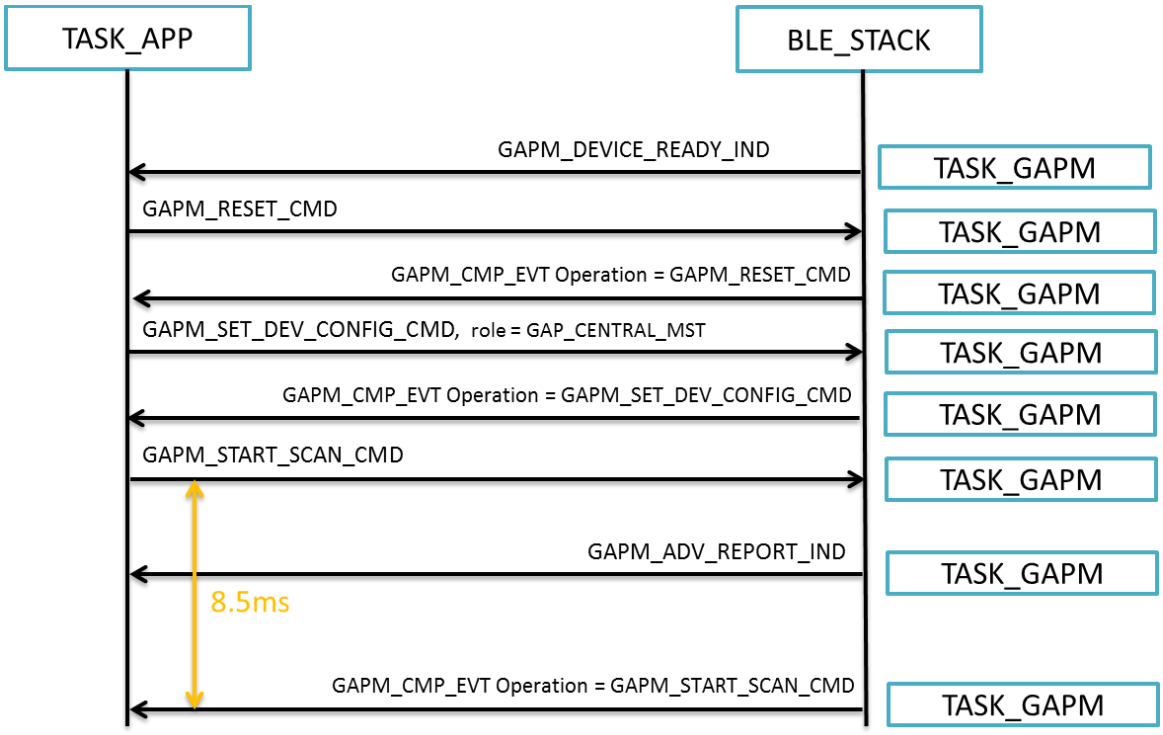


**Figure 5 – Central device initialization sequence**

## 12.2 Further reading

GAP Interface Specification (RW-BLE-GAP-IS) [6]

# 13 Revision history

| Revision | Date | Description |
|---|---|---|
| 1.0 | 28-Mar-2014 | Initial version. |
| | | |
| | | |

### Status definitions

| Status | Definition |
|---|---|
| DRAFT | The content of this document is under review and subject to formal approval, which may result in modifications or additions. |
| APPROVED or unmarked | The content of this document has been approved for publication. |

### Disclaimer

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including without limitation the specification and the design of the related semiconductor products, software and applications.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Customer notes that nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document are subject to Dialog Semiconductor's Standard Terms and Conditions of Sale, unless otherwise stated.



### RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).
Dialog Semiconductor's statement on RoHS can be found on the customer portal https://support.diasemi.com/. RoHS certificates from our suppliers are available on request.

## Contacting Dialog Semiconductor

Germany Headquarters
*Dialog Semiconductor GmbH*
Phone: +49 7021 805-0

United Kingdom
*Dialog Semiconductor (UK) Ltd*
Phone: +44 1793 757700

The Netherlands
*Dialog Semiconductor B.V.*
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
*Dialog Semiconductor Inc.*
Phone: +1 408 845 8500

Japan
*Dialog Semiconductor K. K.*
Phone: +81 3 5425 4567

Taiwan
*Dialog Semiconductor Taiwan*
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Singapore
*Dialog Semiconductor Singapore*
Phone: +65 64 849929

China
*Dialog Semiconductor China*
Phone: +86 21 5178 2561

Korea
*Dialog Semiconductor Korea*
Phone: +82 2 3469 8291