RT58x BLE SDK

Service & Profile Guide

V1.0

Introduction

This document is a reference for creating BLE service and profile for Rafael RT58x BLE SDK.
This version applies to the Rafael RT58x SDK version 1.0 and higher.

Table of Contents

1. ATT and GATT Introduction

The Generic Attribute Profile (GATT) defines a service framework using the Attribute Protocol (ATT). This framework defines procedures and formats of services and their characteristics. The procedures defined include discovering, reading, writing, notifying and indicating characteristics, as well as configuring the broadcast of characteristics.

1.1 Generic Attribute Profile (GATT) Introduction

The GATT Profile defines the structure for data exchange. A profile is composed of some elements such as service, characteristic, etc. All of elements show up in the form of Attribute.

The top level of the hierarchy is a profile. A profile is composed of one or more services necessary to fulfill a use case. A service is composed of characteristics or references to other services. Each characteristic contains a value and may contain optional information about the value. The service and characteristic and the components of the characteristic (i.e. value and descriptors) contain the profile data and are all stored in Attributes on the server. (*BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part G*)

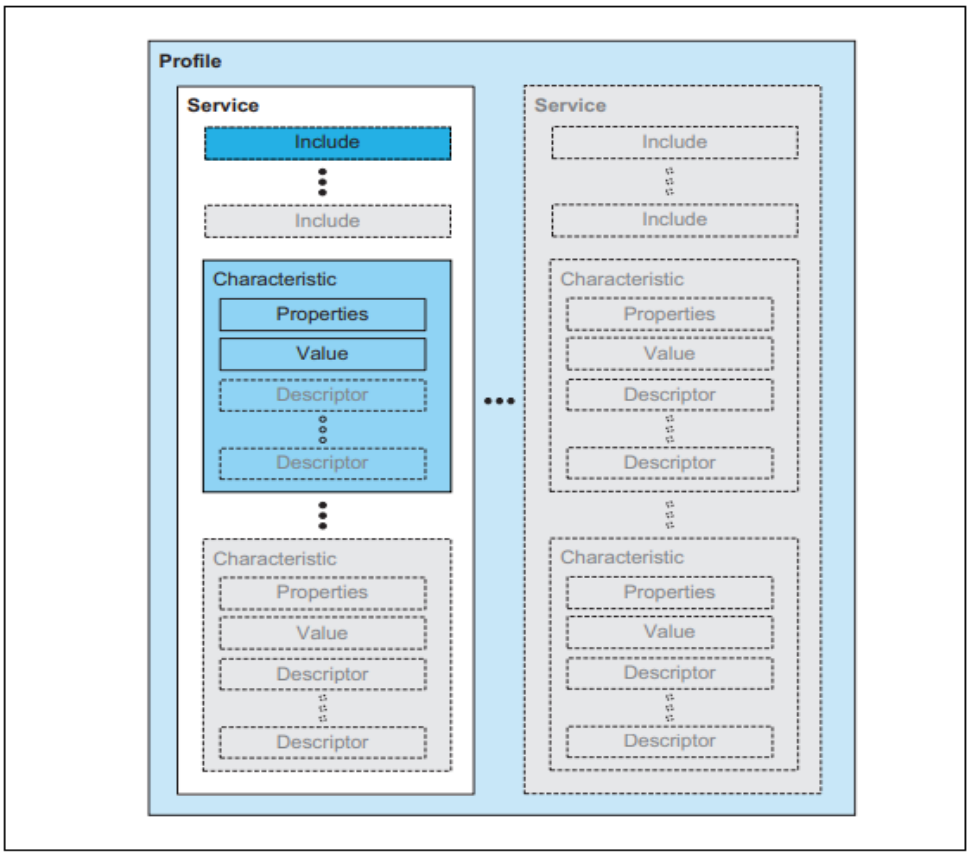


Figure 2.7: GATT Profile hierarchy

1.2 Attribute Protocol (ATT) Introduction

The GATT Profile requires the implementation of the Attribute Protocol (ATT). Attribute is the fundamental form for exchanging data between devices.

An attribute contains Attribute Handle, Attribute Type, Attribute Value, and Attribute Permissions.

- **Attribute Handle**: an index corresponding to a specific Attribute.
- **Attribute Type**: a UUID that describes the Attribute Value.
- **Attribute Value**: the data described by the Attribute Type and indexed by the Attribute Handle.
- **Attribute Permissions**: determine whether read or write access is permitted.

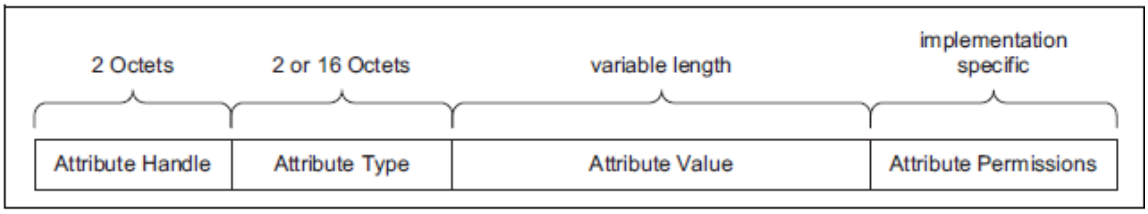


Figure 2.4: Logical Attribute Representation

1.2.1 Service Definition

A service definition shall contain a service declaration and may contain include definitions and characteristic definitions.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

Table 3.1: Service declaration

1.2.2 Characteristic Definition

A characteristic definition shall contain a characteristic declaration, a Characteristic Value declaration and may contain characteristic descriptor declarations.

Attribute Handle	Attribute Types	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Charac-teristic Properties	Character-istic Value Attribute Handle	Character-istic UUID	Read Only, No Authentication, No Authorization

Table 3.3: Characteristic declaration

Worth to mention, the characteristic declaration has contained the characteristic properties.

Properties	Value	Description
Broadcast	0x01	If set, permits broadcasts of the Characteristic Value using Server Characteristic Configuration Descriptor. If set, the Server Characteristic Configuration Descriptor shall exist.
Read	0x02	If set, permits reads of the Characteristic Value using procedures defined in Section 4.8
Write Without Response	0x04	If set, permit writes of the Characteristic Value without response using procedures defined in Section 4.9.1.
Write	0x08	If set, permits writes of the Characteristic Value with response using procedures defined in Section 4.9.3 or Section 4.9.4.
Notify	0x10	If set, permits notifications of a Characteristic Value without acknowledgment using the procedure defined in Section 4.10. If set, the Client Characteristic Configuration Descriptor shall exist.
Indicate	0x20	If set, permits indications of a Characteristic Value with acknowledgment using the procedure defined in Section 4.11. If set, the Client Characteristic Configuration Descriptor shall exist.
Authenticated Signed Writes	0x40	If set, permits signed writes to the Characteristic Value using the procedure defined in Section 4.9.2.
Extended Properties	0x80	If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor defined in Section 3.3.3.1. If set, the Characteristic Extended Properties Descriptor shall exist.

Table 3.5: Characteristic Properties bit field

1.2.3 Characteristic Value Definition

A value of a characteristic is a characteristic value declaration which is the form of attribute.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0xUUUU – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID	Characteristic Value	Higher layer profile or implementation specific

Table 3.6: Characteristic Value declaration

1.2.4 Other Definitions

There are other declarations in specification which doesn't list above. For more detail, please refer the "*BLUETOOTH CORE SPECIFICATION Version 5.2 | Vol 3, Part G*".

2. Create a Customized Service

A profile is composed of one or more services. Therefore, before creating a profile, the services for creating the profile should be built in first.

2.1 Create a Customized Service Table

Follow the following two steps below to create a service table for the next step to create a service.

2.1.1 Step 1: Create a Customized Service Hierarchy

The two required declarations of the characteristic are the characteristic declaration and the Characteristic Value declaration. For each characteristic, need to define the characteristic value and the characteristic properties (here call Properties). Properties represent the Characteristic Properties which determine how the characteristic value can be used. The options of Properties can refer the table below. For properties with notify or indicate, a descriptor called client configuration is necessary in the characteristic.

Properties	Value	Description
Broadcast	0x01	If set, permits broadcasts of the Characteristic Value using Server Characteristic Configuration Descriptor. If set, the Server Characteristic Configuration Descriptor shall exist.
Read	0x02	If set, permits reads of the Characteristic Value using procedures defined in Section 4.8
Write Without Response	0x04	If set, permit writes of the Characteristic Value without response using procedures defined in Section 4.9.1.
Write	0x08	If set, permits writes of the Characteristic Value with response using procedures defined in Section 4.9.3 or Section 4.9.4.
Notify	0x10	If set, permits notifications of a Characteristic Value without acknowledgment using the procedure defined in Section 4.10. If set, the Client Characteristic Configuration Descriptor shall exist.
Indicate	0x20	If set, permits indications of a Characteristic Value with acknowledgment using the procedure defined in Section 4.11. If set, the Client Characteristic Configuration Descriptor shall exist.
Authenticated Signed Writes	0x40	If set, permits signed writes to the Characteristic Value using the procedure defined in Section 4.9.2.
Extended Properties	0x80	If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor defined in Section 3.3.3.1. If set, the Characteristic Extended Properties Descriptor shall exist.

Table 3.5: Characteristic Properties bit field

For example, if the service with three characteristics - read, notify/ indicate. The hierarchy will look like this:

Service	
Characteristic 1	Properties - **read**
	Value
Characteristic 2	Properties – **notify, indicate**
	Value
	Descriptor - client configuration
Characteristic 3	Properties - **read and write**
	Value

2.1.2 Step 2: Create a Customized Service Table

After creating the service hierarchy, we can turn to a table which more like the form in Attribute Protocol. At first, we can turn each level in the hierarchy to a declaration type except for Properties, because Properties will be assign in the value of Characteristic Value Declaration.

For declaration of service or characteristic, we need to give each service and characteristic a UUID as their value, and their permission will be Read only, which means it is readable and no Authentication, No Authorization.

In declaration of characteristic value, we have to define two values, Properties and characteristic value. Properties is the Characteristic Properties which determine above. The characteristic value is optional. It is only can be set a fixed value when its properties contain read and its role is peripheral, so that the value will directly be given to the central when its linked central request of read. Otherwise, it should be set to zero and wait for callback function to do an additional value processing.

For declaration of Client Characteristic Configuration declaration, its value is client configuration bits which decide notify/indicate is active or not. Its Permission is Readable with no authentication or authorization and Writable can be defined by user.

(intentionally blank)

Following the example from Step1 (*section2.1.1*), the following table shows the service hierarchy map to the service table.

Declaration Type	Value	Permission
Service	service UUID	Read only
Characteristic 1	characteristic 1 UUID	Read only
Characteristic 1 Value	Properties – read & characteristic 1 value	(Properties) + no auth
Characteristic 2	characteristic 2 UUID	Read only
Characteristic 2 Value	Properties – notify, indicate	(Properties) + no auth
Characteristic 2 client configuration	characteristic 2 client configuration bits	Read, Write
Characteristic 3	characteristic 3 UUID	Read only
Characteristic 3 Value	Properties - read, write & characteristic 3 value	(Properties)+ no auth

2.2 Map A Service Table to an ble_att_param_t Structure

After creating a service table, we have to turn it to an *ble_att_param_t* structure in code. Rafael RT58x BLE SDK use the *ble_att_param_t* structure to implement corresponding Attribute in specification. This section will divide into three parts to show how a service table turn to *ble_att_param_t*.

- ***ble_att_param_t* Structure Introduction (*2.2.1*)**

  The first part will introduce the *ble_att_param_t* to illustrate the relationship with Attribute in specification.

- **Map Declarations to *ble_att_param_t* (*2.2.2*)**

  The second part will use declaration to illustrate how a declaration turn into the *ble_att_param_t*.

- **Map Service Table to *ble_att_param_t* (*2.3*)**

  The third part will follow the example from *section2.1.2* to show how to map a service table to *ble_att_param_t*.

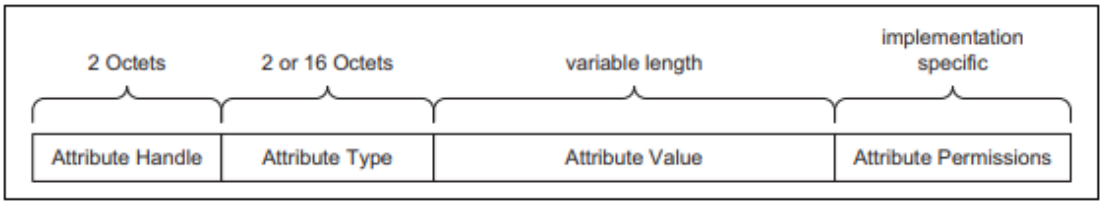
(intentionally blank)

2.2.1 *ble_att_param_t* Structure Introduction

The *ble_att_param_t* is the structure to represent the Attribute of specification.

```
typedef struct ble_att_param_s
{
     /**< Attribute type which defined by a UUID, an UUID is used to identify every
attribute type. */
    void      *p_uuid_type;
     /**< Attribute Value shall be the 16-bit Bluetooth UUID or 128-bit UUID for
          the service/ characteristic, known as the service/ characteristic UUID.*/
    void      *p_uuid_value;
     /**< The length of attribute value.*/
    uint16_t  att_len;
     /**< Characteristic properties.*/
    uint8_t   property_value;
     /**< UUID format and characteristic permission.*/
    uint8_t   db_permission_format;
     /**< Register callback function. */
    void      (*att_handler)(ble_evt_att_param_t *p_param);
} ble_att_param_t;
```

An Attribute in BLE specification is in the form of the following table. For more detail, please refer *Chapter 1.2*.



2.2.1.1 ble_att_param_t – p_uuid_type

The *p_uuid_type* in *ble_att_param_t* is corresponding to Attribute Type in Attribute definition in BLE specification.

```
const ble_att_param_t att_trsps_primary_service =
{
    (void *)attr_uuid_type_primary_service,
    (void *)attr_uuid_trsps_primary_service,
    sizeof(attr_uuid_trsps_primary_service),
     ...
}
```

2.2.1.2 ble_att_param_t – p_uuid_value

The *p_uuid_value* in *ble_att_param_t* is corresponding to Attribute Value in Attribute definition in BLE specification except Characteristic Declaration which is correspond to only Characteristic UUID of Attribute Value.

For Service Declaration/ Characteristic Declaration, the *p_uuid_value* only contains Service UUID / Characteristic UUID of Attribute Value. The definition of service declaration and characteristic declaration please refer to *section1.2.1* and *section1.2.2*.

```
const ble_att_param_t att_trsps_primary_service =
{
    (void *)attr_uuid_type_primary_service,
    (void *)attr_uuid_trsps_primary_service,
    sizeof(attr_uuid_trsps_primary_service),
      ...
}
```

For other declarations, the *p_uuid_value* can directly map to Attribute Value. Worth to mention that the characteristic value declaration has a special feature. For Characteristic Value Declaration, *p_uuid_value* can be set to a fixed value or zero. The *p_uuid_value* can only be set to a fixed value when properties of this characteristic contain read and its role is peripheral, so that the *p_uuid_value* will directly be given to the central when its linked master request of read. If *p_uuid_value* is set to 0, then the mechanism will wait until its linked master request of read or other action and then decide what kind of data to be sent in the callback function.

In Rafael RT58x BLE SDK, all of the Characteristic Value Declaration are set to zero, user shall handle the data into service callback function.

```
const ble_att_param_t att_trsps_udatr01 =
{
    (void *)attr_uuid_trsps_charc_udatr01,
    (void *)0,
    0,
      ...
}
```

2.2.1.3 ble_att_param_t – att_len

The *att_len* is the length of *att_len* in *ble_att_param_t*.

```
const ble_att_param_t att_trsps_primary_service =
{
    (void *)attr_uuid_type_primary_service,
    (void *)attr_uuid_trsps_primary_service,
    sizeof(attr_uuid_trsps_primary_service),
      ...
}
```

2.2.1.4 ble_att_param_t – property_value

The *property_value* in *ble_att_param_t* is corresponding to Characteristic Properties for Characteristic Value Declaration and the readable and writeable function of Attribute Permission for other declarations.

For Characteristic Value Declaration, the *property_value* map to Characteristic Properties of the characteristic. In the same time, it implicates the corresponding Attribute Permission. For example, Characteristic Properties is Read, then in *ble_att_param_t*, the *property_value* should be set like:

property_value = GATT_DECLARATIONS_PROPERTIES_READ

It implicates that the Characteristic Properties is “Read” and the “Attribute Permission is readable”.

For other declarations, the *property_value* directly map to Attribute Permission. It uses GATT_DECLARATIONS_PROPERTIES_READ to represent readable in Attribute Permission and GATT_DECLARATIONS_PROPERTIES_WRITE to represent writable in Attribute Permission. For example, the Permission is readable, then in *ble_att_param_t*, the *property_value* should be set like:

property_value = GATT_DECLARATIONS_PROPERTIES_READ

```
const ble_att_param_t att_trsps_udatr01 =
{
    (void *)attr_uuid_trsps_charc_udatr01,
    (void *)0,
    0,
    (
        //GATT_DECLARATIONS_PROPERTIES_BROADCAST |
        GATT_DECLARATIONS_PROPERTIES_READ |
        //GATT_DECLARATIONS_PROPERTIES_WRITE_WITHOUT_RESPONSE |
        //GATT_DECLARATIONS_PROPERTIES_WRITE |
        //GATT_DECLARATIONS_PROPERTIES_NOTIFY |
        //GATT_DECLARATIONS_PROPERTIES_INDICATE |
        //GATT_DECLARATIONS_PROPERTIES_AUTHENTICATED_SIGNED_WRITES |
        //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
        0x00
    ),
      ...
}
```

(intentionally blank)

2.2.1.5 ble_att_param_t – db_permission_format

The *db_permission_format* in *ble_att_param_t* is corresponding to its Type Format and the encryption, authentication and authorization of Attribute Permission. Type Format is corresponding to the format of *p_uuid_type*. There are two kind of format, 16 bits UUID and 128 bits UUID.

db_permission_format							
Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Type format	Bond enable	Attribute Permission					
		Enc. Read	Enc. Write	authentication read	authentication write	Authorization read	Authorization write

```
const ble_att_param_t att_trsps_udatr01 =
{
    (void *)attr_uuid_trsps_charc_udatr01,
    (void *)0,
    0,
    (
        //GATT_DECLARATIONS_PROPERTIES_BROADCAST |
        GATT_DECLARATIONS_PROPERTIES_READ |
        //GATT_DECLARATIONS_PROPERTIES_WRITE_WITHOUT_RESPONSE |
        //GATT_DECLARATIONS_PROPERTIES_WRITE |
        //GATT_DECLARATIONS_PROPERTIES_NOTIFY |
        //GATT_DECLARATIONS_PROPERTIES_INDICATE |
        //GATT_DECLARATIONS_PROPERTIES_AUTHENTICATED_SIGNED_WRITES |
        //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
        0x00
    ),
    (
        //ATT_TYPE_FORMAT_16UUID |          //otherwise, 128bit UUID
        //ATT_VALUE_BOND_ENABLE |
        //ATT_PERMISSION_ENC_READ |
        //ATT_PERMISSION_ENC_WRITE |
        //ATT_PERMISSION_AUTHE_READ |
        //ATT_PERMISSION_AUTHE_WRITE |
        //ATT_PERMISSION_AUTHO_READ |
        //ATT_PERMISSION_AUTHO_WRITE |
        0x00
    ),
    ble_svcs_trsps_handler,      //registered callback function
};
```

(intentionally blank)

2.2.1.6 ble_att_param_t – att_handler

The *attCallFunc* in *ble_att_param_t* is the callback function for the Characteristic Value declaration. After the linked role requests actions(read/write/notify/indicate), the callback function will be active. For other declarations, *attCallFunc* is set to *attr_null_access*, because it doesn't need a callback function.

```
const ble_att_param_t att_trsps_udatr01 =
{
    (void *)attr_uuid_trsps_charc_udatr01,
    (void *)0,
    0,
    ...
    ble_svcs_trsps_handler,      //registered callback function
};

const ble_att_param_t att_trsps_characteristic_udatni01 =
{
    (void *)attr uuid type characteristic,
    (void *)attr_uuid_trsps_charc_udatni01,
    sizeof(attr_uuid_trsps_charc_udatni01),
    ...
    attr_null_access,
};
```

(intentionally blank)

2.2.2 Map Declarations to ble_att_param_t

This section will use declaration to illustrate how a declaration turn into *ble_att_param_t*. Worth to mention that there is handle in attribute of declaration, but not in *ble_att_param_t*.
The reason is that the mechanism in Rafael RT58x BLE SDK will automatic assign its handle, so we don't need to assign handle here.

2.2.2.1 Service Declaration

This is the attribute structure of service declaration in BLE specification.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for «Primary Service» OR 0x2801 for «Secondary Service»	16-bit Bluetooth UUID or 128-bit UUID for Service	Read Only, No Authentication, No Authorization

Table 3.1: Service declaration

This is the corresponding structure by *ble_att_param_t*.

```
const uint16_t attr_uuid_type_primary_service[] =
{
   GATT_DECL_PRIMARY_SERVICE,
};
const uint16_t attr_uuid_trsps_primary_service[] =
{
   0xEEFF, 0xCCDD,
   0xAABB, 0x8899,
   0x6677, 0x4455,
   0x2233, 0x0011,
};
const ble_att_param_t att_trsps_primary_service =
{
   (void *)attr_uuid_type_primary_service,
   (void *)attr uuid trsps primary service,
   sizeof(attr_uuid_trsps_primary_service),
   (
      //GATT DECLARATIONS PROPERTIES BROADCAST |
      GATT DECLARATIONS PROPERTIES READ |
      //GATT DECLARATIONS PROPERTIES WRITE WITHOUT_RESPONSE |
      //GATT DECLARATIONS PROPERTIES WRITE |
      //GATT DECLARATIONS PROPERTIES NOTIFY |
      //GATT DECLARATIONS PROPERTIES INDICATE |
      //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
      //ATT VALUE BOND ENABLE |
      //ATT PERMISSION ENC READ |
      //ATT PERMISSION ENC WRITE |
      //ATT PERMISSION AUTHE READ |
      //ATT PERMISSION AUTHE WRITE |
      //ATT PERMISSION AUTHO READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   attr_null_access,
};
```

2.2.2.2 Characteristic Declaration

This is the attribute structure of characteristic declaration in BLE specification.

Attribute Handle	Attribute Types	Attribute Value			Attribute Permissions
0xNNNN	0x2803–UUID for «Characteristic»	Charac-teristic Properties	Character-istic Value Attribute Handle	Character-istic UUID	Read Only, No Authentication, No Authorization

Table 3.3: Characteristic declaration

Attribute Value in Characteristic declaration contain three parts in the list below.

- **Characteristic Properties:** assign the value to *db_permission_format* in Characteristic Value declaration.
- **Characteristic Value Attribute Handle:** BLE stack automatic assign.
- **Characteristic UUID:** the value should be set to *p_uuid_value* in Characteristic declaration.

This is the corresponding structure by *ble_att_param_t*.

```
const uint16_t attr_uuid_type_characteristic[] =
{
   GATT_DECL_CHARACTERISTIC,
};
const uint16_t attr_uuid_trsps_charc_udatr01[] =
{
   0x1E1F, 0x1C1D,
   0x1A1B, 0x1819,
   0x1617, 0x1415,
   0x1213, 0x1011,
};
const ble_att_param_t att_trsps_characteristic_udatr01 =
{
   (void *)attr_uuid_type_characteristic,
   (void *)attr uuid trsps charc udatr01,
   sizeof(attr_uuid_trsps_charc_udatr01),
   (
      //GATT DECLARATIONS PROPERTIES BROADCAST |
      GATT DECLARATIONS PROPERTIES READ |
      //GATT DECLARATIONS PROPERTIES WRITE WITHOUT_RESPONSE |
      //GATT DECLARATIONS PROPERTIES WRITE |
      //GATT DECLARATIONS PROPERTIES NOTIFY |
      //GATT DECLARATIONS PROPERTIES INDICATE |
      //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
      //ATT VALUE BOND ENABLE |
      //ATT PERMISSION ENC READ |
      //ATT PERMISSION ENC WRITE |
      //ATT PERMISSION AUTHE READ |
      //ATT PERMISSION AUTHE WRITE |
      //ATT PERMISSION AUTHO READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   attr_null_access,
};
```

2.2.2.3 Characteristic Value Declaration

This is the attribute structure of characteristic value declaration in BLE specification.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0xUUUU – 16-bit Bluetooth UUID or 128-bit UUID for Characteristic UUID	Characteristic Value	Higher layer profile or implementation specific

Table 3.6: Characteristic Value declaration

This is the corresponding structure by *ble_att_param_t*.

```
const uint16_t attr_uuid_trsps_charc_udatr01[] =
{
    0x1E1F, 0x1C1D,
    0x1A1B, 0x1819,
    0x1617, 0x1415,
    0x1213, 0x1011,
};
void ble_svcs_trsps_handler(ble_evt_att_param_t *p_param)
{
     // callback function to handle service events
}
const ble_att_param_t att_trsps_udatr01 =
{
    (void *)attr_uuid_trsps_charc_udatr01,
    (void *)0,
    0,
    (
        //GATT DECLARATIONS PROPERTIES BROADCAST |
        GATT DECLARATIONS PROPERTIES READ |
        //GATT DECLARATIONS PROPERTIES WRITE WITHOUT_RESPONSE |
        //GATT DECLARATIONS PROPERTIES WRITE |
        //GATT DECLARATIONS PROPERTIES NOTIFY |
        //GATT DECLARATIONS PROPERTIES INDICATE |
        //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
        //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
        0x00
    ),
    (
        //ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
        //ATT VALUE BOND ENABLE |
        //ATT PERMISSION ENC READ |
        //ATT_PERMISSION_ENC_WRITE |
        //ATT PERMISSION AUTHE READ |
        //ATT PERMISSION AUTHE WRITE |
        //ATT_PERMISSION_AUTHO_READ |
        //ATT_PERMISSION_AUTHO_WRITE |
        0x00
    ),
    ble_svcs_trsps_handler,      //registered callback function
};
```

2.2.2.4 Client Characteristic Configuration Declaration

This is the attribute structure of client characteristic configuration declaration in BLE specification.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2902 – UUID for «Client Characteristic Configuration»	Characteristic Configuration Bits	Readable with no authentication or authorization. Writable with authentication and authorization defined by a higher layer specification or is implementation specific.

Table 3.10: Client Characteristic Configuration declaration

This is the corresponding structure by *ble_att_param_t*.

```
const uint16_t attr_uuid_type_client_charc_configuration[] =
{
    GATT_DESC_CLIENT_CHARC_CONFIGURATION,
};
void ble_svcs_trsps_handler(ble_evt_att_param_t *p_param)
{
     // callback function to handle service events
}
const ble_att_param_t att_trsps_udatni01_client_charc_configuration =
{
    (void *)attr_uuid_type_client_charc_configuration,
    (void *)0,
    0,
    (
        //GATT DECLARATIONS PROPERTIES BROADCAST |
        GATT DECLARATIONS PROPERTIES READ |
        //GATT DECLARATIONS PROPERTIES WRITE_WITHOUT_RESPONSE |
        GATT DECLARATIONS PROPERTIES WRITE |
        //GATT_DECLARATIONS_PROPERTIES_NOTIFY |
        //GATT DECLARATIONS PROPERTIES INDICATE |
        //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
        //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
        0x00
    ),
    (
        ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
        //ATT VALUE BOND ENABLE |
        //ATT PERMISSION ENC READ |
        //ATT PERMISSION ENC WRITE |
        //ATT PERMISSION AUTHE READ |
        //ATT PERMISSION AUTHE WRITE |
        //ATT PERMISSION AUTHO READ |
        //ATT_PERMISSION_AUTHO_WRITE |
        0x00
    ),
    ble_svcs_trsps_handler,      //registered callback function
};
```

(intentionally blank)

2.3 Map Service Table to ble_att_param_t

Following the previous example from *section2.1.2*, we will create a service table like this:

Declaration Type	Value	Permission
Service	service UUID	Read only
Characteristic 1	characteristic 1 UUID	Read only
Characteristic 1 Value	Properties – read & characteristic 1 value	(Properties) + no auth
Characteristic 2	characteristic 2 UUID	Read only
Characteristic 2 Value	Properties – notify, indicate	(Properties) + no auth
Characteristic 2 client configuration	characteristic 2 client configuration bits	Read, Write
Characteristic 3	characteristic 3 UUID	Read only
Characteristic 3 Value	Properties - read, write & characteristic 3 value	(Properties)+ no auth

Define the UUID of each service and characteristic and the UUID should be in little endian:

Declaration Type	UUID
Service	00112233445566778899AABBCCDDEEFF
Characteristic 1	101112131415161718191A1B1C1D1E1F
Characteristic 2	303132333435363738393A3B3C3D3E3F
Characteristic 3	505152535455565758595A5B5C5D5E5F

```
const uint16_t attr_uuid_trsps_primary_service[] =
{
    0xEEFF, 0xCCDD,
    0xAABB, 0x8899,
    0x6677, 0x4455,
    0x2233, 0x0011,
};
const uint16_t attr_uuid_trsps_charc_udatr01[] =
{
    0x1E1F, 0x1C1D,
    0x1A1B, 0x1819,
    0x1617, 0x1415,
    0x1213, 0x1011,
};
const uint16_t attr_uuid_trsps_charc_udatni01[] =
{
    0x3E3F, 0x3C3D,
    0x3A3B, 0x3839,
    0x3637, 0x3435,
    0x3233, 0x3031,
};
const uint16_t attr_uuid_trsps_charc_udatrw01[] =
{
    0x5E5F, 0x5C5D,
    0x5A5B, 0x5859,
    0x5657, 0x5455,
    0x5253, 0x5051,
};
```

2.3.1 Service Declaration

Refer to the Service Declaration in *section2.2.2.1*.

Declaration Type	Value	Permission
Service (att_trsps_primary_service)	service UUID	Read only

```
const uint16_t attr_uuid_type_primary_service[] =
{
   GATT_DECL_PRIMARY_SERVICE,
};
const uint16_t attr_uuid_trsps_primary_service[] =
{
   0xEEFF, 0xCCDD,
   0xAABB, 0x8899,
   0x6677, 0x4455,
   0x2233, 0x0011,
};
const ble_att_param_t att_trsps_primary_service =
{
   (void *)attr_uuid_type_primary_service,
   (void *)attr_uuid_trsps_primary_service,
   sizeof(attr_uuid_trsps_primary_service),
   (
      //GATT_DECLARATIONS_PROPERTIES_BROADCAST |
      GATT_DECLARATIONS_PROPERTIES_READ |
      //GATT_DECLARATIONS_PROPERTIES_WRITE_WITHOUT_RESPONSE |
      //GATT_DECLARATIONS_PROPERTIES_WRITE |
      //GATT_DECLARATIONS_PROPERTIES_NOTIFY |
      //GATT_DECLARATIONS_PROPERTIES_INDICATE |
      //GATT_DECLARATIONS_PROPERTIES_AUTHENTICATED_SIGNED_WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      ATT_TYPE_FORMAT_16UUID |          //otherwise, 128bit UUID
      //ATT_VALUE_BOND_ENABLE |
      //ATT_PERMISSION_ENC_READ |
      //ATT_PERMISSION_ENC_WRITE |
      //ATT_PERMISSION_AUTHE_READ |
      //ATT_PERMISSION_AUTHE_WRITE |
      //ATT_PERMISSION_AUTHO_READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   attr_null_access,
};
```

(intentionally blank)

2.3.2 Characteristic 1 Declaration

Refer to the Characteristic Declaration in *section2.2.2.2*.

Declaration Type	Value	Permission
...		
Characteristic 1 (att_trsps_characteristic_udatr01)	characteristic 1 UUID	Read only

```
const uint16_t attr_uuid_type_characteristic[] =
{
   GATT_DECL_CHARACTERISTIC,
};
const uint16_t attr_uuid_trsps_charc_udatr01[] =
{
   0x1E1F, 0x1C1D,
   0x1A1B, 0x1819,
   0x1617, 0x1415,
   0x1213, 0x1011,
};
const ble_att_param_t att_trsps_characteristic_udatr01 =
{
   (void *)attr uuid type characteristic,
   (void *)attr uuid trsps charc udatr01,
   sizeof(attr_uuid_trsps_charc_udatr01),
   (
      //GATT DECLARATIONS PROPERTIES BROADCAST |
      GATT DECLARATIONS PROPERTIES READ |
      //GATT DECLARATIONS PROPERTIES WRITE WITHOUT_RESPONSE |
      //GATT DECLARATIONS PROPERTIES WRITE |
      //GATT DECLARATIONS PROPERTIES NOTIFY |
      //GATT DECLARATIONS PROPERTIES INDICATE |
      //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
      //ATT VALUE BOND ENABLE |
      //ATT PERMISSION ENC READ |
      //ATT_PERMISSION_ENC_WRITE |
      //ATT PERMISSION AUTHE READ |
      //ATT PERMISSION AUTHE WRITE |
      //ATT_PERMISSION_AUTHO_READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   attr_null_access,
};
```

(intentionally blank)

2.3.3 Characteristic 1 Value Declaration

Refer to the Characteristic Value Declaration in *section2.2.2.3*.

Declaration Type	Value	Permission
...		
Characteristic 1 Value (att_trsps_udatr01)	Properties – read & characteristic 1 value	(Properties) + no auth

```
const uint16_t attr_uuid_trsps_charc_udatr01[] =
{
   0x1E1F, 0x1C1D,
   0x1A1B, 0x1819,
   0x1617, 0x1415,
   0x1213, 0x1011,
};
void ble_svcs_trsps_handler(ble_evt_att_param_t *p_param)
{
     // callback function to handle service events
}
const ble_att_param_t att_trsps_udatr01 =
{
   (void *)attr_uuid_trsps_charc_udatr01,
   (void *)0,
   0,
   (
      //GATT DECLARATIONS PROPERTIES BROADCAST |
      GATT DECLARATIONS PROPERTIES READ |
      //GATT DECLARATIONS PROPERTIES WRITE WITHOUT_RESPONSE |
      //GATT DECLARATIONS PROPERTIES WRITE |
      //GATT DECLARATIONS PROPERTIES NOTIFY |
      //GATT DECLARATIONS PROPERTIES INDICATE |
      //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      //ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
      //ATT VALUE BOND ENABLE |
      //ATT PERMISSION ENC READ |
      //ATT_PERMISSION_ENC_WRITE |
      //ATT PERMISSION AUTHE READ |
      //ATT PERMISSION AUTHE WRITE |
      //ATT_PERMISSION_AUTHO_READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   ble_svcs_trsps_handler,      //registered callback function
};
```

(intentionally blank)

2.3.4 Characteristic 2 Declaration

Refer to the Characteristic Declaration in *section2.2.2.2*.

Declaration Type	Value	Permission
...		
Characteristic 2 (att_trsps_characteristic_udatni01)	characteristic 2 UUID	Read only

```
const uint16_t attr_uuid_type_characteristic[] =
{
   GATT_DECL_CHARACTERISTIC,
};
const uint16_t attr_uuid_trsps_charc_udatni01[] =
{
   0x3E3F, 0x3C3D,
   0x3A3B, 0x3839,
   0x3637, 0x3435,
   0x3233, 0x3031,
};
const ble_att_param_t att_trsps_characteristic_udatni01 =
{
   (void *)attr_uuid_type_characteristic,
   (void *)attr uuid trsps charc udatni01,
   sizeof(attr_uuid_trsps_charc_udatni01),
   (
      //GATT DECLARATIONS PROPERTIES BROADCAST |
      GATT DECLARATIONS PROPERTIES READ |
      //GATT DECLARATIONS PROPERTIES WRITE WITHOUT_RESPONSE |
      //GATT DECLARATIONS PROPERTIES WRITE |
      //GATT DECLARATIONS PROPERTIES NOTIFY |
      //GATT DECLARATIONS PROPERTIES INDICATE |
      //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
      //ATT VALUE BOND ENABLE |
      //ATT PERMISSION ENC READ |
      //ATT PERMISSION ENC WRITE |
      //ATT PERMISSION AUTHE READ |
      //ATT PERMISSION AUTHE WRITE |
      //ATT PERMISSION AUTHO READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   attr_null_access,
};
```

(intentionally blank)

2.3.5 Characteristic 2 Value Declaration

Refer to the Characteristic Value Declaration in *section2.2.2.3*.

Declaration Type	Value	Permission
…		
Characteristic 2 Value (att_trsps_udatni01)	Properties – notify, indicate	(Properties) + no auth

```
const uint16_t attr_uuid_trsps_charc_udatni01[] =
{
    0x3E3F, 0x3C3D,
    0x3A3B, 0x3839,
    0x3637, 0x3435,
    0x3233, 0x3031,
};
void ble_svcs_trsps_handler(ble_evt_att_param_t *p_param)
{
     // callback function to handle service events
}
const ble_att_param_t att_trsps_udatni01 =
{
    (void *)attr_uuid_trsps_charc_udatni01,
    (void *)0,
    0,
    (
        //GATT DECLARATIONS PROPERTIES BROADCAST |
        //GATT DECLARATIONS PROPERTIES READ |
        //GATT DECLARATIONS PROPERTIES WRITE WITHOUT_RESPONSE |
        //GATT DECLARATIONS PROPERTIES WRITE |
        GATT DECLARATIONS PROPERTIES NOTIFY |
        GATT DECLARATIONS PROPERTIES INDICATE |
        //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
        //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
        0x00
    ),
    (
        //ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
        //ATT VALUE BOND ENABLE |
        //ATT PERMISSION ENC READ |
        //ATT_PERMISSION_ENC_WRITE |
        //ATT PERMISSION AUTHE READ |
        //ATT PERMISSION AUTHE WRITE |
        //ATT_PERMISSION_AUTHO_READ |
        //ATT_PERMISSION_AUTHO_WRITE |
        0x00
    ),
    ble_svcs_trsps_handler,      //registered callback function
};
```

(intentionally blank)

2.3.6 Characteristic 2 Client Characteristic Configuration Declaration

Refer to the Client Characteristic Configuration Declaration from *section2.2.2.4*.

Declaration Type	Value	Permission
...		
Characteristic 2 client configuration	characteristic 2 client configuration bits	Read, Write

```
const uint16_t attr_uuid_type_client_charc_configuration[] =
{
   GATT_DESC_CLIENT_CHARC_CONFIGURATION,
};
void ble_svcs_trsps_handler(ble_evt_att_param_t *p_param)
{
     // callback function to handle service events
}
const ble_att_param_t att_trsps_udatni01_client_charc_configuration =
{
   (void *)attr_uuid_type_client_charc_configuration,
   (void *)0,
   0,
   (
      //GATT_DECLARATIONS_PROPERTIES_BROADCAST |
      GATT_DECLARATIONS_PROPERTIES_READ |
      //GATT_DECLARATIONS_PROPERTIES_WRITE_WITHOUT_RESPONSE |
      GATT_DECLARATIONS_PROPERTIES_WRITE |
      //GATT_DECLARATIONS_PROPERTIES_NOTIFY |
      //GATT_DECLARATIONS_PROPERTIES_INDICATE |
      //GATT_DECLARATIONS_PROPERTIES_AUTHENTICATED_SIGNED_WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      ATT_TYPE_FORMAT_16UUID |          //otherwise, 128bit UUID
      //ATT_VALUE_BOND_ENABLE |
      //ATT_PERMISSION_ENC_READ |
      //ATT_PERMISSION_ENC_WRITE |
      //ATT_PERMISSION_AUTHE_READ |
      //ATT_PERMISSION_AUTHE_WRITE |
      //ATT_PERMISSION_AUTHO_READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   ble_svcs_trsps_handler,      //registered callback function
};
```

(intentionally blank)

2.3.7 Characteristic 3 Declaration

Refer to the Characteristic Declaration in *section2.2.2.2*.

Declaration Type	Value	Permission
...		
Characteristic 3 (att_trsps_characteristic_udatrw01)	characteristic 3 UUID	Read only

```
const uint16_t attr_uuid_type_characteristic[] =
{
   GATT_DECL_CHARACTERISTIC,
};
const uint16_t attr_uuid_trsps_charc_udatrw01[] =
{
   0x5E5F, 0x5C5D,
   0x5A5B, 0x5859,
   0x5657, 0x5455,
   0x5253, 0x5051,
};
const ble_att_param_t att_trsps_characteristic_udatrw01 =
{
   (void *)attr uuid type characteristic,
   (void *)attr uuid trsps charc udatrw01,
   sizeof(attr_uuid_trsps_charc_udatrw01),
   (
      //GATT DECLARATIONS PROPERTIES BROADCAST |
      GATT DECLARATIONS PROPERTIES READ |
      //GATT DECLARATIONS PROPERTIES WRITE WITHOUT_RESPONSE |
      //GATT DECLARATIONS PROPERTIES WRITE |
      //GATT DECLARATIONS PROPERTIES NOTIFY |
      //GATT DECLARATIONS PROPERTIES INDICATE |
      //GATT DECLARATIONS PROPERTIES AUTHENTICATED SIGNED WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      ATT TYPE FORMAT 16UUID |          //otherwise, 128bit UUID
      //ATT VALUE BOND ENABLE |
      //ATT PERMISSION ENC READ |
      //ATT PERMISSION ENC WRITE |
      //ATT PERMISSION AUTHE READ |
      //ATT PERMISSION AUTHE WRITE |
      //ATT PERMISSION AUTHO READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   attr_null_access,
};
```

(intentionally blank)

2.3.8 Characteristic 3 Value Declaration

Refer to the Characteristic Value Declaration in *section2.2.2.3*.

Declaration Type	Value	Permission
...		
Characteristic 3 Value (att_trsps_udatrw01)	Properties - read, write & characteristic 3 value	(Properties)+ no auth

```
const uint16_t attr_uuid_trsps_charc_udatrw01[] =
{
   0x5E5F, 0x5C5D,
   0x5A5B, 0x5859,
   0x5657, 0x5455,
   0x5253, 0x5051,
};
void ble_svcs_trsps_handler(ble_evt_att_param_t *p_param)
{
     // callback function to handle service events
}
const ble_att_param_t att_trsps_udatrw01 =
{
   (void *)attr_uuid_trsps_charc_udatrw01,
   (void *)0,
   0,
   (
      //GATT_DECLARATIONS_PROPERTIES_BROADCAST |
      GATT_DECLARATIONS_PROPERTIES_READ |
      GATT_DECLARATIONS_PROPERTIES_WRITE_WITHOUT_RESPONSE |
      GATT_DECLARATIONS_PROPERTIES_WRITE |
      //GATT_DECLARATIONS_PROPERTIES_NOTIFY |
      //GATT_DECLARATIONS_PROPERTIES_INDICATE |
      //GATT_DECLARATIONS_PROPERTIES_AUTHENTICATED_SIGNED_WRITES |
      //GATT_DECLARATIONS_PROPERTIES_EXTENDED_PROPERTIES |
      0x00
   ),
   (
      //ATT_TYPE_FORMAT_16UUID |          //otherwise, 128bit UUID
      //ATT_VALUE_BOND_ENABLE |
      //ATT_PERMISSION_ENC_READ |
      //ATT_PERMISSION_ENC_WRITE |
      //ATT_PERMISSION_AUTHE_READ |
      //ATT_PERMISSION_AUTHE_WRITE |
      //ATT_PERMISSION_AUTHO_READ |
      //ATT_PERMISSION_AUTHO_WRITE |
      0x00
   ),
   ble_svcs_trsps_handler,      //registered callback function
};
```

(intentionally blank)

2.4 Combine *ble_att_param_t* to a Service

A set of *ble_att_param_t* structures will be created if following the steps in *section2.3*. Then, define a service which composed with the *ble_att_param_t* structures. This service define will be used in creating a profile in *section3*.

```
/** TRSPS Definition
 * @ingroup service_trsps_ServiceChardef
*/
#define ATT_TRSPS_SERVICE                                   \
   &att_trsps_primary_service,                              \
   &att_trsps_characteristic_udatr01,                       \
   &att_trsps_udatr01,                                      \
   &att_trsps_characteristic_udatni01,                      \
   &att_trsps_udatni01,                                     \
   &att_trsps_udatni01_client_charc_configuration,          \
   &att_trsps_characteristic_udatrw01,                      \
   &att_trsps_udatrw01                                      \
```

2.5 Implement Service Related Definition and Function

Implement the service definitions and functions for application in service related files.

- **Customized Definition**
  - Service event (Optional)
  - Service attribute handle structure
  - Service data structure (Optional)
  - Service application data structure
  - Service event callback function (Optional)
  - Other customized service definitions (Optional)
- **Customized Function**
  - Initial service
  - Get service attribute handle
  - Handle callback function (Optional)
  - Other customized functions (Optional)

(intentionally blank)

2.5.1 Customized Definition

All the definitions are set in “ble_service_xxx.h” file.

2.5.1.1 Service event definition (Optional)

Define service event indicates different situations for application.

```
/**< TRSPS characteristic UDATR01 read event.*/
#define BLESERVICE_TRSPS_UDATR01_READ_EVENT                    0x01
/**< TRSPS characteristic UDATR01 read response event.*/
#define BLESERVICE_TRSPS_UDATR01_READ_RSP_EVENT                0x02
/**< TRSPS characteristic UDATNI01 notify event.*/
#define BLESERVICE_TRSPS_UDATNI01_NOTIFY_EVENT                 0x03
/**< TRSPS characteristic UDATNI01 indicate confirm event.*/
#define BLESERVICE_TRSPS_UDATNI01_INDICATE_CONFIRM_EVENT     0x04
/**< TRSPS characteristic UDATNI01 indicate event.*/
#define BLESERVICE_TRSPS_UDATNI01_INDICATE_EVENT               0x05
/**< TRSPS characteristic UDATNI01 cccd read event.*/
#define BLESERVICE_TRSPS_UDATNI01_CCCD_READ_EVENT              0x06
/**< TRSPS characteristic UDATNI01 cccd read response event.*/
#define BLESERVICE_TRSPS_UDATNI01_CCCD_READ_RSP_EVENT          0x07
/**< TRSPS characteristic UDATNI01 cccd write event.*/
#define BLESERVICE_TRSPS_UDATNI01_CCCD_WRITE_EVENT             0x08
/**< TRSPS characteristic UDATNI01 cccd write response event.*/
#define BLESERVICE_TRSPS_UDATNI01_CCCD_WRITE_RSP_EVENT        0x09
/**< TRSPS characteristic UDATRW01 read event.*/
#define BLESERVICE_TRSPS_UDATRW01_READ_EVENT                    0x0a
/**< TRSPS characteristic UDATRW01 read response event.*/
#define BLESERVICE_TRSPS_UDATRW01_READ_RSP_EVENT               0x0b
 /**< TRSPS characteristic UDATRW01 write event.*/
#define BLESERVICE_TRSPS_UDATRW01_WRITE_EVENT                   0x0c
/**< TRSPS characteristic UDATRW01 write response event.*/
#define BLESERVICE_TRSPS_UDATRW01_WRITE_RSP_EVENT               0x0d
/**< TRSPS characteristic UDATRW01 write without response event.*/
#define BLESERVICE_TRSPS_UDATRW01_WRITE_WITHOUT_RSP_EVENT     0x0e
```

2.5.1.2 Service attribute handle structure definition

Service and characteristic declaration shall not be defined in service attribute handle structure.

```
#define ATT_TRSPS_SERVICE                                  \
    &att_trsps_primary_service,                            \
    &att_trsps_characteristic_udatr01,                     \
    &att_trsps_udatr01,                                    \
    &att_trsps_characteristic_udatni01,                    \
    &att_trsps_udatni01,                                   \
    &att_trsps_udatni01_client_charc_configuration,        \
    &att_trsps_characteristic_udatrw01,                    \
    &att_trsps_udatrw01                                    \

typedef struct ble_svcs_trsps_handles_s
{
    uint16_t hdl_udatr01;      /**< Handle of UDATR01. */
    uint16_t hdl_udatni01;     /**< Handle of UDATNI01. */
    uint16_t hdl_udatni01_cccd; /**< Handle of UDATNI01 cccd. */
    uint16_t hdl_udatrw01;     /**< Handle of UDATRW01. */
} ble_svcs_trsps_handles_t;
```

2.5.1.3 Service data structure definition (Optional)

The service data structure shall be implemented if the service includes the cccd (Client Configuration Characteristic Descriptor), otherwise this structure is an option.

```
typedef struct ble_svcs_trsps_data_s
{
    uint16_t udatni01_cccd;    /**< UDATNI01 cccd value */
} ble_svcs_trsps_data_t;
```

2.5.1.4 Service application data structure definition

The service application data structure shall be defined “BLE GATT role”, “service attribute handles” and “service data” shall be defined if the service includes the cccd and other definitions are optional.

```
/** TRSPS Application Data Structure Definition
 * @ingroup app_trsps_structureDef
*/
typedef struct ble_svcs_trsps_subinfo_s
{
    ble_svcs_trsps_handles_t handles;    /**< GAPS attribute handles */
    ble_svcs_trsps_data_t    data;       /**< GAPS attribute data */
} ble_svcs_trsps_subinfo_t;


typedef struct ble_svcs_trsps_info_s
{
    ble_gatt_role_t           role;         /**< BLE GATT role */
    ble_svcs_trsps_subinfo_t  client_info;
    ble_svcs_trsps_subinfo_t  server_info;
} ble_svcs_trsps_info_t;
```

2.5.1.5 Service event callback function definition (Optional)

The service event callback function shall be implemented if user registers the callback to received BLE GATT event likes write request, write response, etc.

```
/** ble_svcs_trsps_handler
 * @note This callback receives the TRSPS events.
 * Each of these events can be associated with parameters.
*/
void ble_svcs_trsps_handler(ble_evt_att_param_t *p_param);
```

2.5.1.6 Other customized service definitions (Optional)

Suggest user implemented here if there are any customized service definitions.

(intentionally blank)

2.5.2 Customized Function

All the functions are implemented in “ble_service_xxx.c” file. There are variable and structure definitions shall be defined for service application flow control.

- **Service Basic Information**

```
// Service basic information
ble_svcs_common_info_t            trsps_basic_info[MAX_NUM_CONN_TRSPS];
```

- **Service Specific Information**

```
// TRSPS information
ble_svcs_trsps_info_t             *trsps_info[MAX_NUM_CONN_TRSPS];
```

- **Service Event Callback Definition**

```
// TRSPS callback function
ble_svcs_evt_trsps_handler_t      trsps_callback[MAX_NUM_CONN_TRSPS];
```

- **Service Registered Count Number**

  The variable indicated the count of registered services.

```
// TRSPS registered total count
uint8_t                           trsps_count = 0;
```

(intentionally blank)

2.5.2.1 Initial service

Implemented service initialization function to register service with setting service basic information, service specific information, service callback function and the count of registered service.

```
/** TRSPS Initialization */
ble_status_t ble_svcs_trsps_init(uint8_t                          host_id,
                                 ble_gatt_role_t                  role,
                                 ble_svcs_trsps_info_t           *p_info,
                                 ble_svcs_evt_trsps_handler_t callback)
{
    ble_err_t status;
    uint8_t config_index;

    if (p_info == NULL)
    {
        return BLE_ERR_INVALID_PARAMETER;
    }

    // init service client basic information and get "config_index" & "trsps_count"
    status = ble_svcs_common_init(host_id, role,
                                  MAX_NUM_CONN_TRSPS,
                                  trsps_basic_info,
                                  &config_index,
                                  &trsps_count);
    if (status != BLE_ERR_OK)
    {
        return status;
    }

    // Set service role
    p_info->role = role;

    // Set TRSPS data
    trsps_info[config_index] = p_info;

    // Register TRSPS callback function
    trsps_callback[config_index] = callback;

   // Get handles at initialization if role is set to BLE_GATT_ROLE_SERVER
    if ((role & BLE_GATT_ROLE_SERVER) != 0)
    {
        status = ble_svcs_trsps_handles_get(host_id, BLE_GATT_ROLE_SERVER,
trsps info[config index]);
        if (status != BLE_ERR_OK)
        {
            return status;
        }
    }

    return BLE_ERR_OK;
}
```

(intentionally blank)

2.5.2.2 Get service attribute handles

Implemented getting service attribute handles function for service application uses.

```
/* Get TRSPS Handle Numbers */
ble_err_t ble_svcs_trsps_handles_get(uint8_t host_id,  ble_gatt_role_t role,
ble_svcs_trsps_info_t *p_info)
{
    ble_err_t status;
    ble_gatt_handle_table_param_t ble_gatt_handle_table_param;

    status = BLE_ERR_OK;
    do
    {
        ble_gatt_handle_table_param.host_id = host_id;
        ble_gatt_handle_table_param.gatt_role = p_info->role;
        ble_gatt_handle_table_param.p_element = (ble_att_param_t *)&att_trsps_pri-
mary_service;

        if (role == BLE_GATT_ROLE_SERVER)
        {
            ble_gatt_handle_table_param.p_handle_num_addr = (void *)&p_info-
>server_info.handles;
        }
        else if (role == BLE_GATT_ROLE_CLIENT)
        {
            ble gatt handle table param.p handle num addr = (void *)&p info->cli-
ent_info.handles;
        }
        else
        {
            info_color(LOG_RED, "Error role setting.\n");
            status = BLE_ERR_INVALID_PARAMETER;
            break;
        }
        status = ble_svcs_handles_mapping_get(&ble_gatt_handle_table_param);
    } while (0);

    return status;
}
```

2.5.2.3 Handle callback function (Optional)

Issue “ble_svcs_common_info_index_query()” function to query the index of the service data which is registered in “ble_svcs_trsps_init ()” function. User could use this index to get the service related information.

Implemented two handle callback functions for BLE server role and BLE client role.

```
// handle TRSPS client GATT event
static void handle_trsps_client(uint8_t index, ble_evt_att_param_t *p_param)
{
    ...
}
// handle TRSPS server GATT event
static void handle_trsps_server(uint8_t index, ble_evt_att_param_t *p_param)
{
    ...
}
// TRSPS registered callback function
void ble_svcs_trsps_handler(ble_evt_att_param_t *p_param)
{
   uint8_t index;

   if (ble_svcs_common_info_index_query(p_param->host_id,
                                        p_param->gatt_role,
                                        MAX_NUM_CONN_TRSPS,
                                        trsps_basic_info,
                                        &index) != BLE_STATUS_SUCCESS)
   {
      // Host id has not registered so there is no callback function -> do nothing
      return;
   }
   if (p_param->gatt_role == BLE_GATT_ROLE_CLIENT)
   {
      // handle TRSPS client GATT event
      handle_trsps_client(index, p_param);
   }
   if (p_param->gatt_role == BLE_GATT_ROLE_SERVER)
   {
      // handle TRSPS server GATT event
      handle_trsps_server(index, p_param);
   }
}
```

(intentionally blank)

Handle BLE event likes write response, read response, etc. for BLE client role. Using attribute handle number to identify the meaning of the data.

```
// handle TRSPS client GATT event
static void handle_trsps_client(uint8_t index, ble_evt_att_param_t *p_param)
{
    switch (p_param->opcode)
    {
    case OPCODE_ATT_READ_RESPONSE:
        if (p_param->handle_num == trsps_info[index]->client_info.handles.hdl_udatni01_cccd)
        {
            // received read response (cccd value) from server
            p_param->event = BLESERVICE_TRSPS_UDATNI01_CCCD_READ_RSP_EVENT;
            trsps_evt_post(p_param, &trsps_callback[index]);
        }
        break;
    case OPCODE_ATT_WRITE_RESPONSE:
        if (p_param->handle_num == trsps_info[index]->client_info.handles.hdl_udatni01_cccd)
        {
            // received write response from server -> cccd configure completed
            p_param->event = BLESERVICE_TRSPS_UDATNI01_CCCD_WRITE_RSP_EVENT;
            trsps_evt_post(p_param, &trsps_callback[index]);
        }
        else if (p_param->handle_num == trsps_info[index]->client_info.handles.hdl_udatrw01)
        {
            // received write response from server
            p_param->event = BLESERVICE_TRSPS_UDATRW01_WRITE_RSP_EVENT;
            trsps_evt_post(p_param, &trsps_callback[index]);
        }
        break;
    case OPCODE_ATT_HANDLE_VALUE_NOTIFICATION:
        if (p_param->handle_num == trsps_info[index]->client_info.handles.hdl_udatni01)
        {
            // received notification from server
            p_param->event = BLESERVICE_TRSPS_UDATNI01_NOTIFY_EVENT;
            trsps_evt_post(p_param, &trsps_callback[index]);
        }
        break;
    case OPCODE_ATT_HANDLE_VALUE_INDICATION:
        if (p_param->handle_num == trsps_info[index]->client_info.handles.hdl_udatni01)
        {
            // received notification from server
            p_param->event = BLESERVICE_TRSPS_UDATNI01_INDICATE_EVENT;
            trsps_evt_post(p_param, &trsps_callback[index]);
        }
        break;

    default:
        break;
    }
}
```

Handle BLE event likes write request, read request, etc. for BLE server role. Using attribute handle number to identify the meaning of the data.

```
// handle TRSPS server GATT event
static void handle_trsps_server(uint8_t index, ble_evt_att_param_t *p_param)
{
    switch (p_param->opcode)
    {
    case OPCODE_ATT_READ_REQUEST:
    case OPCODE_ATT_READ_BY_TYPE_REQUEST:
        if (p_param->handle_num == trsps_info[index]->server_info.handles.hdl_udatni01_cccd)
        {
            // received read or read by type request from client -> send read or read by type re-
sponse
            ble_svcs_auto_handle_cccd_read_req(p_param, trsps_info[index]-
>server_info.data.udatni01_cccd);
        }
        else if (p_param->handle_num == trsps_info[index]->server_info.handles.hdl_udatr01)
        {
            // received read or read by type request from client -> send read or read by type rsp
with data back to client
            ble_svcs_auto_handle_read_req(p_param, (uint8_t *)ATTR_VALUE_TRSPS_UDATR01,
(sizeof(ATTR_VALUE_TRSPS_UDATR01) - 1));
        }
        else if (p_param->handle_num == trsps_info[index]->server_info.handles.hdl_udatrw01)
        {
            // received read or read by type request from client -> post to user to prepare read
data back to client
            p_param->event = BLESERVICE_TRSPS_UDATRW01_READ_EVENT;
            trsps_evt_post(p_param, &trsps_callback[index]);
        }
        break;
    case OPCODE_ATT_WRITE_REQUEST:
        if (p_param->handle_num == trsps_info[index]->server_info.handles.hdl_udatni01_cccd)
        {
            // received write request (cccd value) from client -> update server defined cccd value
            ble_svcs_handle_cccd_write_req(p_param->data, p_param->length, &trsps_info[index]-
>server_info.data.udatni01_cccd);
        }
        else if (p_param->handle_num == trsps_info[index]->server_info.handles.hdl_udatrw01)
        {
            // received write request from client -> post to user
            p_param->event = BLESERVICE_TRSPS_UDATRW01_WRITE_EVENT;
            trsps_evt_post(p_param, &trsps_callback[index]);
        }
        break;
    case OPCODE_ATT_WRITE_COMMAND:
        if (p_param->handle_num == trsps_info[index]->server_info.handles.hdl_udatrw01)
        {
            // received write command from client -> post to user
            p_param->event = BLESERVICE_TRSPS_UDATRW01_WRITE_WITHOUT_RSP_EVENT;
            trsps_evt_post(p_param, &trsps_callback[index]);
        }
        break;
    default:
        break;
    }
}
```

2.5.2.4 List supported GATT requests and responses.

Client	Server
Write Request	

// call "ble_cmd_gatt_write_req()"
ble_err_t
ble_cmd_gatt_write_req(ble_gatt_data_param_t *p_param);
// handle xxx service event
void
ble_svcs_trsps_handler(ble_evt_att_param_t *p_param)
{
  switch (p_param->opcode)
  {
  case OPCODE_ATT_WRITE_RESPONSE:
  break;
  }
}
	// handle xxx service event
	void ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)
	{
	  switch (p_param->opcode)
	  {
	  case OPCODE_ATT_WRITE_REQUEST:
	  // write response will be issued
	  // automatically by BLE stack.
	  // handle data from client via write
	  // request.
	  break;
	  }
	}

Write Command	

// call "ble_cmd_gatt_write_cmd()"
ble_err_t
ble_cmd_gatt_write_cmd(ble_gatt_data_param_t *p_param);
 	
// No service event shall be handled
	// handle xxx service event
	void ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)
	{
	  switch (p_param->opcode)
	  {
	  case OPCODE_ATT_WRITE_COMMAND:
	  // handle data from client via write
	  // command.
	  break;
	  }
	}

Read Request	

// call "ble_cmd_gatt_read_req()"
ble_err_t
ble_cmd_gatt_read_req(ble_gatt_read_req_param_t *p_param);
 	
// handle xxx service event
void
ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)
{
  switch (p_param->opcode)
  {
  case OPCODE_ATT_READ_RESPONSE:
  // handle data from client via read
  // request.
  break;
  }
}
	// send read response
	ble_err_t
	ble_cmd_gatt_read_rsp(ble_gatt_data_param_t *p_param);
	// handle xxx service event
	void ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)
	{
	  switch (p_param->opcode)
	  {
	  case OPCODE_ATT_READ_REQUEST:
	  // send read response by
	  // "ble_gatt_read_rsp"
	  break;
	  }
	}

Read Blob Request	

`// call "ble_cmd_gatt_read_blob_req()"`
`ble_err_t`
`ble_cmd_gatt_read_blob_req(ble_gatt_read_blob_req_param_t *p_param);`
`// handle xxx service event`
`void`
`ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)`
`{`
`  switch (p_param->opcode)`
`  {`
`  case OPCODE_ATT_READ_BLOB_RESPONSE:`
`  // handle data from client via read`
`  // blob request.`
`  break;`
`  }`
`}`
	`// send read blob response`
	`ble_err_t`
	`ble_cmd_gatt_read_blob_rsp(ble_gatt_data_param_t *p_param);`
	`// handle xxx service event`
	`void ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)`
	`{`
	`  switch (p_param->opcode)`
	`  {`
	`  case OPCODE_ATT_READ_BLOB_REQUEST:`
	`  // send read blob response by`
	`  // "ble_gatt_read_blob_rsp"`
	`  break;`
	`  }`
	`}`

Read By Type Request	

Not supported.	`// send read by type response`
	`ble_err_t`
	`ble_cmd_gatt_read_by_type_rsp(ble_gatt_data_param_t *p_param);`
	`// handle xxx service event`
	`void ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)`
	`{`
	`  switch (p_param->opcode)`
	`  {`
	`  case OPCODE_ATT_READ_BY_TYPE_REQUEST:`
	`  // send read by type response by`
	`  // "ble_gatt_read_by_type_rsp"`
	`  break;`
	`  }`
	`}`

	Notification

`// handle xxx service event`
`void`
`ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)`
`{`
`  switch (p_param->opcode)`
`  {`
`  case OPCODE_ATT_HANDLE_VALUE_NOTIFICATION:`
`  // handle data from server via`
`  // notification.`
`  break;`
`  }`
`}`
	`// call "ble_gatt_notification()"`
	`ble_status_t ble_cmd_gatt_notification(ble_gatt_data_param_t *p_param);`
	
	`// No service event shall be handled`

	Indication

// handle xxx service event
void
ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)
{
  switch (p_param->opcode)
  {
  case OPCODE_ATT_HANDLE_VALUE_INDICATION:
  // confirmation will be issued
  // automatically by BLE stack.
  // handle data from server via indication.
  break;
  }
}
	// send indication
	ble_err_t ble_cmd_gatt_indication(ble_gatt_data_param_t *p_param);
	// handle xxx service event
	void
	ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)
	{
	  switch (p_param->opcode)
	  {
	  case OPCODE_ATT_HANDLE_VALUE_CONFIRMATION:
	  // here received the confirmation
	  // from the client.
	  break;
	  }
	}

- Error response handling:

Client	Server

// handle xxx service event
void
ble_xxx_trsps_handler(ble_evt_att_param_t *p_param)
{
  switch (p_param->opcode)
  {
  case OPCODE_ATT_ERROR_RESPONSE:
  // handle error response from server.
  break;
  }
}
	// call "ble_gatt_error_rsp()"
	ble_err_t ble_cmd_gatt_error_rsp(ble_gatt_err_rsp_param_t *p_param);

(intentionally blank)

2.5.2.5 Other customized functions (Optional)

Suggest user implemented here if there are any customized service functions. For example, TRSP service implements “ble_svcs_trsps_client_send()” function for application. User could easily issue this function to send data via write or write without response.

```
/** Get data from server by reading request (Client ONLY)
*/
ble_err_t ble_svcs_trsps_client_read(uint8_t host_id, uint16_t handle_num)
{
    int status;
    ble_tlv_t *p_tlv;
    ble_gatt_read_req_param_t *p_param;

    p_tlv = pvPortMalloc(sizeof(ble_tlv_t) + sizeof(ble_gatt_read_req_param_t));
    if (p_tlv != NULL)
    {
        p_tlv->type = TYPE_BLE_GATT_READ_REQ;
        p_tlv->length = sizeof(ble_gatt_read_req_param_t);
        p_param = (ble_gatt_read_req_param_t *)p_tlv->value;
        p_param->host_id = host_id;
        p_param->handle_num = handle_num;

        status = ble_event_msg_sendto(p_tlv);
        if (status != BLE_ERR_OK) // send to BLE stack
        {
            info_color(LOG_RED, "<TYPE_BLE_GATT_READ_REQ> Send msg to BLE stack
fail\n");
        }
        vPortFree(p_tlv);
    }
    else
    {
        info_color(LOG_RED, "<TYPE_BLE_GATT_READ_REQ> malloc fail\n");
        status = BLE_ERR_ALLOC_MEMORY_FAIL;
    }

    return (ble_err_t)status;
}
```

(intentionally blank)

3. Create a Customized Profile

A profile is composed of one or more services necessary to fulfill a use case. In this step, user has to define the service combination for the profile and define the necessary arrays for BLE stack.

Here show the steps in summary to let user know how to create a profile:

- Include pre-defined service header files and the customized services could be created by following the steps in *section2.*
- Define the service combination arrays to the customized profiles.
- Set the profiles for each BLE link definition.
- Create the BLE connection link mapping related arrays for BLE stack.
- Implement customized function for the profiles.

In Rafael RT58x BLE SDK there are two files to define the setting of the profile, “ble_profile_def.c” and “ble_profile_app.c”, and the header file is “ble_profile.h”.

- **“ble_profile_def.c”**
  defines the definition of the profile, the combination of services, the necessary array for BLE stack, etc.
- “**ble_profile_app.c”**
  defines the profile related definition for application uses, get service handles function, etc.

3.1 Included Pre-Defined Service Files

Include the service header files in “ble_profile.h”.

```
#include "stdint.h"
#include "ble_common.h"
#include "ble_service_common.h"
#include "ble_service_dis.h"
#include "ble_service_gaps.h"
#include "ble_service_gatts.h"
#include "ble_service_trsps.h"
```

3.2 Create A Profile Table

Define the service combination arrays to the customized profiles.

```
const ble_att_param_t *const att_service_comb00[] =
{
    &ATT_NULL_INVALID,       //mandatory, don't remove it.
    ATT_GAPS_SERVICE,
    ATT_GATTS_SERVICE,
    ATT_DIS_SERVICE,
    ATT_TRSPS_SERVICE
};
```

3.3 Create BLE Link Definition Table

Define the supported links with defined profile (the combination of the defined services *section3.2*) for BLE host uses. For instance, the following definition illustrates the definition for two BLE links, one for the client role with profile “att_service_comb00” and the other one for the server role with profile “att_service_comb01”. Please note that if there is any unsupported shall be set to *((const ble_att_param_t **)0)*.

```
const ble_att_role_by_id_t att_db_link[] =
{
    // Link 0
    {
        att_service_comb00,                 // Client Profile
        ((const ble_att_param_t **)0),      // Server Profile
    },
    // Link 1
    {
        ((const ble_att_param_t **)0),      // Client Profile
        att_service_comb01,                 // Server Profile
    },
};
```

3.4 Create BLE Link Mapping Table

3.4.1 Define BLE Link Mapping Parameters

Create a *ble_att_handle_param_t* array with the size of corresponding the profile table which created from section3.2 to each connection link.

```
ble_att_handle_param_t att_hdl_para_links00[SIZE_ARRAY_ROW(att_service_comb00)]; // Link 0 Client
ble_att_handle_param_t att_hdl_para_links01[SIZE_ARRAY_ROW(att_service_comb01)]; // Link 1 Server
```

3.4.2 Define BLE Link Mapping Table

Create a “ble_att_db_mapping_by_id_t” table with the defined BLE connection link mapping parameters which are created from *section3.4.1* to the corresponding link roles. Please note that if there is any unsupported shall be set to *((ble_att_handle_param_t *)0)*.

```
const ble_att_db_mapping_by_id_t att_db_mapping[] =
{
    // Link 0
    {
        att_hdl_para_links00,               // Client Link Parameter
        ((ble_att_handle_param_t *)0),      // Server Link Parameter
            },
    // Link 1
    {
        ((ble_att_handle_param_t *)0),      // Client Link Parameter
        att_hdl_para_links01,               // Server Link Parameter
    },
};
```

3.4.3 Define BLE Link Mapping Size Table

Create a “ble_att_db_mapping_by_id_size_t” table with the defined profile from *section3.2* to the corresponding link roles.

```
const ble_att_db_mapping_by_id_size_t att_db_mapping_size[] =
{
    // Link 0
    {
        SIZE_ARRAY_ROW(att_service_comb00),    // Client Link Mapping Size
        0,                                     // Server Link Mapping Size
    },
    // Link 1
    {
        0,                                     // Client Link Mapping Size
        SIZE_ARRAY_ROW(att_service_comb00),    // Server Link Mapping Size
    },
};
```

3.5 Define the Maximum Number of BLE Connection

The size of “att_db_mapping_size” is the maximum number of supported BLE connections and the definition of “att_db_mapping_size” is defined in *section3.4.3*.

```
/** Maximum Number of Host Connection Link Definition
 * @attention Do NOT modify this definition.
 * @note Defined for host layer.
*/
const uint8_t max_num_conn_host =
(SIZE_ARRAY_ROW(att_db_mapping_size));
```

3.6 Define BLE Host Connection Link Information

Define “param_rsv_host” array for BLE stack to store BLE host connection link related information.

```
/** Host Connection Link Information Definition
 * @attention Do NOT modify this definition.
 * @note Defined for host layer.
*/
uint8_t *param_rsv_host[SIZE_ARRAY_ROW(att_db_mapping_size)][(REF_SIZE_LE_HOST_PARA >> 2)];
```

(intentionally blank)

3.7 Define the number of connection link for each service

In this combination of services, there is a GAP, a DIS, HRS and a TRSP services for 2 connection links.

```
const ble_att_param_t *const att_service_comb00[] =
{
    &ATT_NULL_INVALID,        //mandatory, don't remove it.
    ATT_GAPS_SERVICE,
    ATT_GATTS_SERVICE,
    ATT_DIS_SERVICE,
    ATT_TRSPS_SERVICE
};
const ble_att_param_t *const att_service_comb01[] =
{
    &ATT_NULL_INVALID,        //mandatory, don't remove it.
    ATT_GAPS_SERVICE,
    ATT_GATTS_SERVICE,
    ATT_DIS_SERVICE,
    ATT_HRS_SERVICE
};

const ble_att_role_by_id_t att_db_link[] =
{
    // Link 0
    {
        att_service_comb00,                // Client Profile
        ((const ble_att_param_t **)0),     // Server Profile
    },
    // Link 1
    {
        ((const ble_att_param_t **)0),     // Client Profile
        att_service_comb01,                // Server Profile
    },
};
```

	GAP	GATT	DIS	TRSP	HRS
TRSP Client	V	V	V	V	
HRS Server	V	V	V		V
Min. # of service link	2	2	2	1	1

Defined in “ble_profile.h”.

```
/** Define the maximum number of BLE GAPS link. */
#define MAX_NUM_CONN_GAPS      2
/** Define the maximum number of BLE GATTS link. */
#define MAX_NUM_CONN_GATTS     2
/** Define the maximum number of BLE DIS link. */
#define MAX_NUM_CONN_DIS       2
/** Define the maximum number of BLE TRSPS link. */
#define MAX_NUM_CONN_TRSPS     1
/** Define the maximum number of BLE HRS link. */
#define MAX_NUM_CONN_HRS       1
```

3.8 Implement customized function

User could implement customized definition or function in “ble_profile_app.c” file. If the profile supports the role of client, then recommend to implement the function to get all service attribute handle numbers. Issued this function to get the updated attribute numbers when receive BLE event “BLE_ATT_GATT_EVT_DB_PARSE_COMPLETE”.

Implement get all services handles function.

```
/** Get BLE (Central) Service All Handles
*/
ble_err_t svcs_handles_get(uint8_t host_id)
{
    ble_err_t status;
    ble profile info t *p profile info = (ble profile info t
*)ble_app_link_info[host_id].profile_info;

    status = BLE_ERR_OK;
    do
    {
        // Get GAPS handles
        status = ble_svcs_gaps_handles_get(host_id, BLE_GATT_ROLE_CLIENT, (void
*)&p_profile_info->svcs_info_gaps);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        // Get GATTS handles
        status = ble_svcs_gatts_handles_get(host_id, BLE_GATT_ROLE_CLIENT, (void
*)&p_profile_info->svcs_info_gatts);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        // Get DIS handles
        status = ble_svcs_dis_handles_get(host_id, BLE_GATT_ROLE_CLIENT, (void
*)&p_profile_info->svcs_info_dis);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        // Get TRSPS handles
        status = ble svcs trsps handles get(host id, BLE GATT ROLE CLIENT, (void
*)&p_profile_info->svcs_info_trsps);
        if (status != BLE_ERR_OK)
        {
            break;
        }
    } while (0);

    return status;
}
```

Handle “DB parsing complete” event.

```
static void app_central_handler(app_req_param_t *p_param)
 {
     switch (p_param->app_req)
    {
          case APP_REQUEST_TRSPC_MULTI_CMD:
        if (svcs_trspc_multi_cmd_handler(host_id) == BLE_ERR_OK)
        {
            // Multiple commands set finished
            printf("Ready to TX/RX data to/from the connected server. \n");
        }
        else
        {
            app_request_set(host_id, APP_REQUEST_TRSPC_MULTI_CMD, false);
        }
        break;
          ...
     }
}

static void ble_evt_handler(ble_evt_param_t *p_param)
{
      switch (event)
    {
    case BLE ATT GATT EVT DB PARSE COMPLETE:
    {
        ble evt att db parse complete t *p parsing param = (ble evt att db parse com-
plete_t *)&p_param->event_param.ble_evt_att_gatt.param.ble_evt_att_db_parse_complete;

        // Get all service handles and related information
        svcs_handles_get(p_parsing_param->host_id);

        // Do GATT commands
        app_request_set(p_parsing_param->host_id, APP_REQUEST_TRSPC_MULTI_CMD, false);

        printf("DB Parsing completed, ID:%d status:0x%02x\n", p_parsing_param-
>host_id, p_parsing_param->result);
    }
          ...
     }
}
```

(intentionally blank)

4. Revision History

Revision	Description	Owner	Date
1.0	Initial version.	Yuwei	2022/03