Qualcomm Technologies International, Ltd.

Confidential and Proprietary – Qualcomm Technologies International, Ltd.

(formerly known as Cambridge Silicon Radio Ltd.)

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm Technologies International, Ltd.or its affiliated companies without the express approval of Qualcomm Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies International, Ltd.

Any software provided with this notice is governed by the Qualcomm Technologies International, Ltd. Terms of Supply or the applicable license agreement at https://www.csrsupport.com/CSRTermsandConditions.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. All Qualcomm Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

© 2015 Qualcomm Technologies International, Ltd. All rights reserved.

Qualcomm Technologies International, Ltd.
Churchill House
Cambridge Business Park
Cambridge, CB4 0WZ
United Kingdom

Push every boundary.™

GATT Database Generator

User Guide

Issue 10

© Cambridge Silicon Radio Limited 2011-2014

Document History

Revision	Date	History
1	29 SEP 11	Original publication of this document
2	21 MAR 12	Updated to include Bluetooth Low Energy
3	26 MAR 12	Correction to section 1
4	26 NOV 12	CSR added to µEnergy product name
5	17 DEC 12	Added information on using guards for multiple database files
6	10 MAY 13	Added information on using 32-bit UUIDs
7	11 NOV 13	Updated to new CSR branding. Specifying numbers added to Section 2.1. Example in Section 2.2.2 updated.
8	11 NOV 13	Minor formatting correction
9	01 DEC 13	Updated the use of 128-bit UUID in canonical form
10	24 JAN 14	Updated page 3

Contacts

General information	www.csr.com
Information on this product	sales@csr.com
Customer support for this product	www.csrsupport.com
More detail on compliance and standards	product.compliance@csr.com
Help with this document	comments@csr.com

Trademarks, Patents and Licences

Unless otherwise stated, words and logos marked with ™ or ® are trademarks registered or owned by CSR plc and/or its affiliates.

Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR.

Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc or its affiliates.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

Use of this document is permissible only in accordance with the applicable CSR licence agreement.

Safety-critical Applications

CSR's products are not designed for use in safety critical devices or systems such as those relating to: (i) life support; (ii) nuclear power; and/or (iii) civil aviation applications, or other applications where injury or loss of life could be reasonably foreseeable as a result of the failure of a product. The customer agrees not to use CSR's products (or supply CSR's products for use) in such devices or systems.

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.

Contents

Tables, Figures and Equations

© Cambridge Silicon Radio Limited 2011-2014

1. Introduction

GATT based profiles require the use of a database which a remote client accesses using procedures as specified in the *GATT Specification* (Volume 3, Part G of the *Bluetooth Core Specification*).

In BlueCore and CSR µEnergy Software Development Kit (SDK) projects the database is described using a special object language. This database can be automatically generated by the GATT Database Generator. This allows the application developer to create the database in an easily readable and maintainable manner without the need for complex binary representations, such as those used for SDP records.

GATT Database Generator input files have a `.db` extension. They are passed through a C pre-processor before being processed by the GATT Database Generator to create `.h` and `.c` files named after the input file. If the `.h` or `.c` files already exist and have not been auto-generated, the GATT Database Generator returns an error code of `1` which causes the build process to abort.

Only one input file may be supplied to the GATT Database Generator, but that file may use the `#include` directive to pull in additional database files. All additional database files except the last in the list must include a trailing comma (`,`) as the last character. It is advisable to use `#include` directive guards in each additional database file. For example, if a database file to be included is named `include_db_1.db`, then it should contain:

```
#ifndef _INCLUDE_DB_1
#define _INCLUDE_DB_1

<... database definitions ...>

#endif /* _INCLUDE_DB_1 */
```

Use the Help command line switch for further details. i.e. `gattdbgen.exe --help`.

© Cambridge Silicon Radio Limited 2011-2014

CSR

2. GATT Database Input File Format

2.1. File Format Basics

The file format is based on JavaScript Object Notation (JSON), which is a lightweight text-based human-readable format. The input file can contain

- Comments
- White space
- Separators (comma)
- Attributes (objects)
- Attribute information (members)

Comments and white space are ignored, and there are no specific rules for indentation.

Every attribute is an object, where the object type defines the type of the attribute:

- Primary Service
- Secondary Service
- Include
- Characteristic Declaration
- Characteristic Extended Properties
- Characteristic User Description
- Client Characteristic Configuration
- Server Characteristic Configuration
- Characteristic Presentation Format
- Characteristic Aggregate Format

All the object types are defined in the *GATT Specification*.

Every attribute may contain extra information to specify the characteristics of the attribute. Attribute information is contained within the attribute object as members.

Objects and members are separated from each other using a comma (`,`).

Numbers can be specified in both decimal and hexadecimal representations, with the exception of 128-bit numbers which must be specified using hexadecimal representation. Hexadecimal numbers must be prefixed with `0x`.

In BlueCore projects, the 128-bit UUIDs can also be specified using the canonical form e.g. 112233-4455-6677-8899-aabbccddeeff.

© Cambridge Silicon Radio Limited 2011-2014

2.2. Object Types

2.2.1. Primary Service

2.2.1.1. Reference

GATT Specification section 3.1

2.2.1.2. Synopsis

```
primary_service {...}
```

2.2.1.3. Supported Member Objects

- Include
- Characteristic Declaration

2.2.1.4. Supported Members

Synopsis	Description	Notes
`uuid` : number	16, 32 or 128 bit UUID describing the type of the service.	Mandatory
`sdp` : boolean	If set to `TRUE` an SDP record is generated for the service.	If `TRUE`, the `name` member is mandatory
`name` : string	A human readable name to describe the attribute.	See section 2.3
`flags` : value `flags` : array	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.1: Primary Service Members

2.2.1.5. Example

Primary service which contains a SDP record for BR/EDR service discovery, and one characteristic:

```
primary_service {
      uuid : 0x1801,
      name : "SERVICE_GATT",
      sdp : true,

      characteristic {      uuid : 0x2a05, properties : 0 }
}
```

2.2.2. Secondary Service

2.2.2.1. Reference

GATT Specification section 3.1

2.2.2.2. Synopsis

```
secondary_service {...}
```

2.2.2.3. Supported Member Objects

- Characteristic Declaration

2.2.2.4. Supported Members

Synopsis	Description	Notes
`uuid : number`	16, 32 or 128 bit UUID describing the type of the service.	Mandatory
`name : string`	A human readable name to describe the attribute.	See section 2.3

`flags : value`
`flags : array`
	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.2: Secondary Service Members

2.2.2.5. Example

Secondary service with 128-bit UUID `00112233-4455-6677-8899-aabbccddeeff`:

```
secondary_service {
     uuid : 0x112233445566778899aabbccddeeff,
     name : "MY_OWN_SERVICE"
}
```

In BlueCore projects, the 128-bit UUID in the above service definition can also be specified in the canonical form:

```
secondary_service {
     uuid : 112233-4455-6677-8899-aabbccddeeff,
     name : "MY_OWN_SERVICE"
}
```

CSR

2.2.3. Include

2.2.3.1. Reference

GATT Specification section 3.2

2.2.3.2. Synopsis

```
include {...}
```

2.2.3.3. Supported Member Objects

None

2.2.3.4. Supported Members

Synopsis	Description	Notes
`ref : ` string	Named reference to a Secondary Service to be included.	Mandatory
`name : ` string	A human readable name to describe the attribute.	See section 2.3
`flags : ` value `flags : ` array	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.3: Include Members

2.2.3.5. Example

Include a secondary service named `"MY_OWN_SERVICE"`:

```
include {
     ref : "MY_OWN_SERVICE"
}
```

2.2.4. Characteristic Declaration

2.2.4.1. Reference

GATT Specification section 3.3.1

2.2.4.2. Synopsis

```
characteristic {...}
```

2.2.4.3. Supported Member Objects

- Characteristic Extended Properties
- Characteristic User Description
- Client Characteristic Configuration
- Server Characteristic Configuration
- Characteristic Presentation Format
- Characteristic Aggregate Format

2.2.4.4. Supported Members

Synopsis	Description	Notes
`properties :` value `properties :` array	Characteristic value properties as defined in GATT Specification section 3.3.1.1. All values are set to a single value using bitwise OR.	Mandatory See Table 2.6.
`uuid :` number	16, 32 or 128 bit UUID describing the type of the characteristic.	Mandatory
`value :` value `value :` array	The value of the characteristic. The value can be a single value, or an array of multiple values. In the case of numeric values the size is determined by the value length, i.e. `0x12` is considered as 8-bit and `0x1234` is considered as 16-bit.	-
`size_value :` number	The length of the characteristic value if the `value` member is not present. This is used to verify write lengths when `FLAG_IRQ` is set.	-
`name :` string	A human readable name to describe the attribute.	See section 2.3
`flags :` value `flags :` array	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.4: Characteristic Declaration Members

© Cambridge Silicon Radio Limited 2011-2014

Table 2.5 describes possible values for characteristic properties.

Name	Description
`number`	A numeric value as defined in the GATT specification.
`broadcast`	If set, permits broadcasts of the Characteristic Value using Characteristic Configuration Descriptor.
`read`	If set, permits reads of the Characteristic Value.
`write_cmd`	If set, permit writes of the Characteristic Value without response.
`write`	If set, permits writes of the Characteristic Value with response.
`notify`	If set, permits notifications of a Characteristic Value without acknowledgement.
`indicate`	If set, permits indications of a Characteristic Value with acknowledgement.
`write_sig`	If set, permits signed writes to the Characteristic Value using Signed Write Command.

Table 2.5: Characteristic Value Properties

2.2.4.5. Example

Readable and notifiable `characteristic` value:

```
characteristic {
     uuid : 0x2a00,
     properties : [ read, notify ],
     value : "My device name"
}
```

© Cambridge Silicon Radio Limited 2011-2014

CSR

2.2.5. Characteristic Extended Properties

2.2.5.1. Reference

GATT Specification section 3.3.3.1

2.2.5.2. Synopsis

```
extended_properties {...}
```

2.2.5.3. Supported Member Objects

None

2.2.5.4. Supported Members

Synopsis	Description	Notes
`properties` : value `properties` : array	Characteristic value extended properties as defined in GATT Specification section 3.3.3.1. All values are set to a single value using bitwise OR.	Mandatory See Table 2.8.
`name` : string	A human readable name to describe the attribute.	See section 2.3
`flags` : value `flags` : array	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.6: Characteristic Extended Properties Members

Table 2.7 describes possible values for characteristic properties.

Name	Description
number	A numeric value as defined in the GATT specification.
`write_reliable`	If set, permits reliable writes of the Characteristic Value.
`write_auxiliaries`	If set, permits writes to the characteristic descriptor.

Table 2.7: Characteristic Extended Properties

2.2.5.5. Example

A `characteristic` which allows reliable writes:

```
characteristic {
      uuid : 0x2a00,
      properties : [ read, write_cmd, write ],
      value : "Device name",

      extended_properties { properties: [ write_reliable ] }
}
```

© Cambridge Silicon Radio Limited 2011-2014

2.2.6. Characteristic User Description

2.2.6.1. Reference

GATT Specification section 3.3.3.2

2.2.6.2. Synopsis

```
user_description {...}
```

2.2.6.3. Supported Member Objects

None

2.2.6.4. Supported Members

Synopsis	Description	Notes
`value : ` value `value : ` array	The value of the characteristic. The value can be a single value, or an array of multiple values. In the case of numeric values the size is determined by the value length, i.e. `0x12` is considered as 8-bit and `0x1234` is considered as 16-bit.	-
`size_value : ` number	The length of the value if `value` member is not present. This is used to verify write lengths when `FLAG_IRQ` is set.	-
`name : ` string	A human readable name to describe the attribute.	See section 2.3
`flags : ` value `flags : ` array	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.8: Characteristic User Description Members

2.2.6.5. Example

A write-only (control point) characteristic with read-only user description:

```
characteristic {
     uuid : 0x1234,
     properties : [ write_cmd, write ],
     name : "CONTROL_POINT",
     flags : [ FLAG_IRQ ],
     size_value : 1,

     user_description { value : "Control point" }
}
```

2.2.7. Client Characteristic Configuration

2.2.7.1. Reference

GATT Specification section 3.3.3.3

2.2.7.2. Synopsis

```
client_config {...}
```

2.2.7.3. Supported Member Objects

None

2.2.7.4. Supported Members

Synopsis	Description	Notes
`name : ` string	A human readable name to describe the attribute.	See section 2.3
`flags : ` value `flags : ` array	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.9: Client Characteristic Configuration Members

2.2.7.5. Example

A characteristic with Client Configuration:

```
characteristic {
     uuid : 0x1234,
     properties :  [ notify ],
     name : "CHARACTERISTIC_NOTIFY",

     client_config {
          flags : [ FLAG_IRQ ],
          name : "CHARACTERISTIC_NOTIFY_CONFIG"
     }
}
```

2.2.8. Server Characteristic Configuration

2.2.8.1. Reference

GATT Specification section 3.3.3.4

2.2.8.2. Synopsis

```
server_config {...}
```

2.2.8.3. Supported Member Objects

None

2.2.8.4. Supported Members

Synopsis	Description	Notes

`properties : value`
`properties : array`
	Characteristic value extended properties as defined in GATT Specification section 3.3.3.4. All values are set to a single value using bitwise OR.	Mandatory
		See Table 2.11.

`name : string`	A human readable name to describe the attribute.	See section 2.3

`flags : value`
`flags : array`
	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.10: Server Characteristic Configuration Members

Table 2.11 describes possible values for server characteristic configuration properties.

Name	Description
number	A numeric value as defined in the GATT Specification.
`broadcast`	The Characteristic Value shall be broadcast when the server is in the broadcast procedure if advertising data resources are available.

Table 2.11: Server Characteristic Configuration Properties

2.2.8.5. Example

A `characteristic` that may be broadcast:

```
characteristic {
     uuid : 0x1234,
     properties : [ broadcast ],

     server_config {
          flags : [ FLAG_IRQ ],
          name : "BROADCAST_CONFIG",
          properties : 0
     }
}
```

2.2.9. Characteristic Presentation Format

2.2.9.1. Reference

GATT Specification section 3.3.3.5

2.2.9.2. Synopsis

```
presentation_format {...}
```

2.2.9.3. Supported Member Objects

None

2.2.9.4. Supported Members

Synopsis	Description	Notes
`format` : value	Characteristic value format as defined in GATT Specification section 3.3.3.5.2.	Mandatory See Table 2.13
`exponent` : number	Characteristic value exponent. `exponent` is a signed integer, and the actual value is calculated using the formula: actual value = characteristic value * 10`exponent`	-
`unit` : number	A 16-bit UUID defined in the Bluetooth Assigned Numbers.	-
`name_space` : number	A value defined in the Bluetooth Assigned Numbers that identifies the organisation responsible for defining the description field.	-
`description` : number	A 16-bit enumerated value describing the characteristic.	-
`name` : string	A human readable name to describe the attribute.	See section 2.3
`flags` : value `flags` : array	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.12: Characteristic Presentation Format Members

Table 2.13 describes possible values for characteristic presentation format.

Name	Description	Exponent
`boolean`	Unsigned 1-bit: 0 = false 1 = true	No
`2bit`	Unsigned 2-bit integer	No
`nibble`	Unsigned 4-bit integer	No
`uint8`	Unsigned 8-bit integer	Yes
`uint12`	Unsigned 12-bit integer	Yes
`uint16`	Unsigned 16-bit integer	Yes
`uint24`	Unsigned 24-bit integer	Yes
`uint32`	Unsigned 32-bit integer	Yes
`uint48`	Unsigned 48-bit integer	Yes
`uint64`	Unsigned 64-bit integer	Yes
`uint128`	Unsigned 128-bit integer	Yes
`sint8`	Signed 8-bit integer	Yes
`sint12`	Signed 12-bit integer	Yes
`sint16`	Signed 16-bit integer	Yes
`sint24`	Signed 24-bit integer	Yes
`sint32`	Signed 32-bit integer	Yes
`sint48`	Signed 48-bit integer	Yes
`sint64`	Signed 64-bit integer	Yes
`sint128`	Signed 128-bit integer	Yes
`float32`	IEEE-754 32-bit floating point	No
`float64`	IEEE-754 64-bit floating point	No
`SFLOAT`	IEEE-11073 16-bit SFLOAT	No
`FLOAT`	IEEE-11073 32-bit FLOAT	No
`duint16`	IEEE-20601 format	No
`utf8s`	UTF-8 string	No
`utf16s`	UTF-16 string	No
`struct`	Opaque structure	No

Table 2.13: Characteristic Presentation Formats

© Cambridge Silicon Radio Limited 2011-2014

2.2.9.5. Example

A `characteristic` set to represent 32.0 using the presentation format and a 16-bit unsigned integer value:

```
characteristic {
     uuid : 0x1234,
     value : 0x0140, /* 320 * 10^(-1) = 32.0 */

     presentation_format {
          format : uint16,
          exponent : -1,
          unit : 0x272f, /* Celsius */
          name_space : 0,
          description : 0
     }
}
```

© Cambridge Silicon Radio Limited 2011-2014

2.2.10. Characteristic Aggregate Format

2.2.10.1. Reference

GATT Specification section 3.3.3.6

2.2.10.2. Synopsis

```
aggregate_format {...}
```

2.2.10.3. Supported Member Objects

None

2.2.10.4. Supported Members

Synopsis	Description	Notes
`aggregate :` value `aggregate :` array	List of references (set using a name member) to Characteristic Presentation Format objects to define the format of an aggregated Characteristic Value.	Mandatory
`name :` string	A human readable name to describe the attribute.	See section 2.3
`flags :` value `flags :` array	Attribute level options which control how the attribute can be accessed.	See section 2.3

Table 2.14: Characteristic Aggregate Format Members

2.2.10.5. Example

An aggregate containing two values, referenced as `WEIGHT_KG` and `HEIGHT_CM`:

```
aggregate_format {
     aggregate : [ "WEIGHT_KG", "HEIGHT_CM" ]
}
```

© Cambridge Silicon Radio Limited 2011-2014

CSR

2.3. Common Members

Members described in this section are valid for all objects.

2.3.1. Flags

2.3.1.1. Synopsis

```
flags : value
flags : array
```

2.3.1.2. Description

The flags field is used to specify attribute level options which control how the attribute can be accessed.

2.3.1.3. Value Description

Comma separated list of flag values. Table 2.15 describes possible flag values.

Name	Description	Notes
`FLAG_AUTH_R`	Reading of the attribute is allowed only over an authenticated (MITM protected) link.	-
`FLAG_AUTH_W`	Writing of the attribute is allowed only over an authenticated (MITM protected) link.	-
`FLAG_DYNLEN`	Attribute value length is dynamic, i.e. the length can be changed by writing a different length value into the attribute.	This flag is ignored in objects where the GATT Specification specifies the length of the value.
`FLAG_ENCR_R`	Reading of the attribute is allowed only over an encrypted link.	-
`FLAG_ENCR_W`	Writing of the attribute is allowed only over an encrypted link.	-
`FLAG_IRQ`	Attribute value is handled by the application. Read and write access to the attribute causes `GATT_ACCESS_IND` messages to be sent to the application, and the application responds using `GattAccessResponse()` (BlueCore projects) or `GattAccessRsp()` (CSR µEnergy projects).	This flag should be used when the application needs to take an action upon access to the attribute.

Table 2.15: Flag Values

2.3.1.4. Example

Enable dynamic length and interrupt features in the attribute.

```
flags : [ FLAG_DYNLEN, FLAG_IRQ ]
```

2.3.2. Name

2.3.2.1. Synopsis

```
name : string
```

2.3.2.2. Description

A human readable name to describe the attribute. The name can be used to refer to the object, and every named object handle is `#defined` in the header file (useful for attributes with `FLAG_IRQ` flag set). `name` is mandatory if the SDP record flag in Primary Service is enabled, and is used as the reference for the service's SDP record.

2.3.2.3. Value description

A string defining the attribute name.

2.3.2.4. Example

Set the name of the attribute to `GAP_SERVICE`:

```
name : "GAP_SERVICE"
```

3. Using Generated Databases

3.1. BlueCore Projects

When a database input file is included in a project, the required functions to access and handle the GATT database and SDP records are automatically generated, compiled and linked in with the application. The application is responsible for registering the database with the GATT library, registering SDP records with the SDP server, and handling attribute access interrupts provided to the application in `GATT_ACCESS_IND` messages.

To access functions provided by the GATT database generator an application source file shall `#include` the generated header file containing the data types, constants and function prototypes required for handling the GATT database, SDP records, and GATT library interrupts.

All files generated have the same base name as the database input file, but instead of a `.db` extension the header file has `.h` and source file has `.c` extensions.

3.1.1. Function Prototypes

3.1.1.1. GattGetDatabase

Synopsis

```
uint16 *GattGetDatabase(uint16 *len)
```

Description

Returns the full GATT database in a format suitable to be passed on to `GattInit()`.

After passing the database to `GattInit()` the GATT library owns the data and application shall not try to access it directly. For more information about `GattInit()`, see the `gatt.h` File Reference in the *SDK Reference Documentation*.

Example

A function to register the GATT library with a generated database:

```
void register_gatt(Task theAppTask)
{
      uint16 *db;
      uint16 size_db;

      db = GattGetDatabase(&size_db);
      GattInit(theAppTask, size_db, db);
}
```

3.1.1.2. GattGetServiceRecord

Synopsis

```
uint8 *GattGetServiceRecord(gatt_sdp service, uint16 *len)
```

Description

Returns the service record for the Primary Service defined by the service parameter in a format suitable to be passed on to `ConnectionRegisterServiceRecord()`.

After passing the service record to `ConnectionRegisterServiceRecord()` the Connection library owns the data and application shall not try to access it directly. For more information about `ConnectionRegisterServiceRecord()`, see the `connection_no_ble.h` File Reference in the *SDK Reference Documentation*.

`gatt_sdp` is an enumerated type containing list of SDP records available. For more information about `gatt_sdp` enumerated type see section 3.1.3.1.

Example

A function to register all SDP records in one go:

```
void register_gatt_sdp(Task theAppTask)
{
     uint8 *sdp;
     uint16 size_sdp;
     uint16 i;

     for (i = 0; i < gatt_sdp_last; i++)
     {
          sdp = GattGetServiceRecord(i, &size_sdp);
          if (sdp)
               ConnectionRegisterServiceRecord(theAppTask, size_sdp, sdp);
          else
               Panic();
     }
}
```

Note:

> If the application implements multiple Primary Services to be used over BR/EDR transport it is recommended to register one SDP record at a time, and wait for a `CL_SDP_REGISTER_CFM` message before moving to the next record.

3.1.2. Enumerated Types

3.1.2.1. gatt_sdp

Description

`gatt_sdp` is an enumerated type containing a list of SDP records available. The list entries are named based on the name member of the Primary Service object, and prefixed with `gatt_sdp_`. The last entry on the list is always `gatt_sdp_last`, even if there are no SDP records available.

Example

Using the `gatt_sdp` enumerated type to get a SDP record:

```
primary_service {
     uuid : 0x1234,
     name : "my_service",
     sdp : true
}

void register_gatt_sdp(Task theAppTask)
{
     uint8 *sdp;
     uint16 size_sdp;

     if ((sdp = GattGetServiceRecord(gatt_sdp_my_service, &size_sdp)))
          ConnectionRegisterServiceRecord(theAppTask, size_sdp, sdp);
     else
          Panic();
}
```

3.1.3. Defined Constants

3.1.3.1. Handle Numbers

Handle numbers of all objects with a name member will be defined as constants based on the name and prefixed with `HANDLE_`. These constants are useful when an object defines a `FLAG_IRQ` flag.

For Characteristic Declaration objects the defined number is the handle number of the Characteristic value.

CSR

3.2. CSR µEnergy Projects

When a database input file is included in a project, the required functions to access and handle the GATT database are automatically generated, compiled and linked in with the application. The application is responsible for registering the database with the GATT Server module and handling attribute access events provided to the application in `GATT_ACCESS_IND` messages.

To access functions provided by the GATT database generator an application source file shall `#include` the generated header file containing the data types, constants and function prototypes required for handling the GATT database and GATT events.

All files generated have the same base name as the database input file, but instead of a `.db` extension the header file has `.h` and source file has `.c` extensions.

3.2.1. Function Prototypes

3.2.1.1. GattGetDatabase

Synopsis

```
uint16 *GattGetDatabase(uint16 *len)
```

Description

Returns the full GATT database in a format suitable to be passed on to `GattAddDatabaseReq()`.

After passing the database to `GattAddDatabaseReq()` the GATT Server module owns the data and application shall not try to access it directly. For more information about `GattAddDatabaseReq()`, see the GATT Server module in the *SDK Reference Documentation*.

Example

A function to register a generated database with the GATT Server module:

```
void register_gatt(void)
{
     uint16 *db;
     uint16 size_db;

     db = GattGetDatabase(&size_db);
     GattAddDatabaseReq(size_db, db);
}
```

3.2.2. Defined Constants

3.2.2.1. Handle Numbers

Handle numbers of all objects with a name member will be defined as constants based on the name and prefixed with `HANDLE_`. These constants are useful when an object defines a `FLAG_IRQ` flag.

For Characteristic Declaration objects the defined number is the handle number of the Characteristic value.

3.2.2.2. Attribute Lengths

The length of each attribute in bytes will be defined as a constant based on the attribute name and prefixed with `ATTR_LEN_`.

CSR

Appendix A Quick Reference

Type	Notation
object	type { members }

type	`primary_service`
	`secondary_service`
	`include`
	`characteristic`
	`extended_properties`
	`client_config`
	`server_config`
	`presentation_format`
	`aggregate_format`

members	pair pair , members
pair	member_type : value member_type : array

member_type	`aggregate`
	`config`
	`description`
	`exponent`
	`flags`
	`format`
	`name`
	`name_space`
	`properties`
	`ref`
	`sdp`
	`unit`
	`uuid`
	`value`

array	[elements]
elements	value value , elements
value	string number array
string	" " " chars "
chars	char char chars

Type	Notation
char	any Unicode character except " or \ \" \\ \n \t \r \b \f
number	[0-9]+ -[0-9]+ 0x[0-9a-fA-F]+ 0X[0-9a-fA-F]+

Table A.1: GATT Database Generator Data Types

Document References

Document	Reference
Bluetooth Assigned Numbers	Available at www.bluetooth.org
Bluetooth Core Specification version 4.1	Available at www.bluetooth.org
GATT Specification	Volume 3, Part G of the Bluetooth Core Specification version 4.1
JSON	RFC 4627 www.json.org
SDK Reference Documentation	Supplied with the SDK as: BlueCore SDK: VM and Native Reference Guide CSR µEnergy SDK: Firmware Library Documentation

© Cambridge Silicon Radio Limited 2011-2014

Terms and Definitions

BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
BR/EDR	Basic Rate/Enhanced Data Rate
CSR	Cambridge Silicon Radio
CSR µEnergy®	Group term for CSR's range of Bluetooth Smart wireless technology chips
e.g.	*exempli gratia*, for example
GATT	Generic Attribute Profile
IC	Integrated Circuit
i.e.	*id est*, that is
IEEE	Institute of Electronic and Electrical Engineers
JSON	JavaScript Object Notation
MITM	Man In The Middle
SDK	Software Development Kit
SDP	Service Discovery Protocol; element of Bluetooth
UTF	Unicode Transformation Format
UUID	Universally Unique Identifier

© Cambridge Silicon Radio Limited 2011-2014