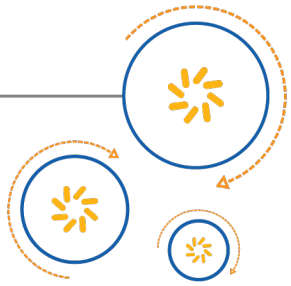
Qualcomm Technologies International, Ltd.



GATT Database Generator

User Guide

80-CT311-1 Rev. AP

October 17, 2017

Qualcomm Technologies International, Ltd. (formerly known as Cambridge Silicon Radio Limited) is a company registered in England and Wales with a registered office at: Churchill House, Cambridge Business Park, Cowley Road, Cambridge, CB4 0WZ, United Kingdom.
Registered Number: 3665875 | VAT number: GB787433096

Revision history

Revision	Date	Description
1	SEP 2011	Initial release. Alternative document number CS-00219225-UG.
2	MAR 2012	Updated to include Bluetooth Low Energy
3	MAR 2012	Correction to Section 1
4	NOV 2012	CSR™ added to product name
5	DEC 2012	Added Information on using guards for multiple database files
6	MAY 2013	Added information on using 32-bit UIDDs
7	NOV 2013	Updated to new CSR branding Specifying numbers added to Section 2.1 Example in Section 2.2 updated
8	NOV 2013	Minor formatting correction
9	DEC 2013	Updated the use of 128-bit UUID in canonical form
10	JAN 2014	Updated page 3
11	APR 2015	Updated for changes that allow GATT database to be stored in VM const memory
12	JUN 2016	Updated to conform to QTI standards; No technical content was changed in this document revision
13	March 2017	Added to the Content Management System. No technical content was changed in this document revision
AP	October 2017	Document Reference Number updated to agile number

Contents

Tables

1 GATT database

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

In Qualcomm® BlueCore™ technology and Bluetooth Low Energy 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` or `.dbi` 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. VM libraries which implement GATT services should define the service as a `.dbi` file so that it is exported to the SDK include folder allowing the application to include the service in the top level `.db` file without duplicating the GATT database definition.

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 `.dbi` files. 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. That is `gattdbgen.exe --help`.

2 GATT database input file format

The GATT database 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, for example 112233-4455-6677-8899-aabbccddeeff.

2.1 GATT database object types

The GATT database uses the following object types which define 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

2.1.1 Object type primary_service

Reference

GATT Specification section 3.1

Synopsis

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

Supported member objects

- Include
- Characteristic Declaration

Supported members

Table 2-1 Primary service 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 GATT database name member
`flags : value` `flags : array`	Attribute level options which control how the attribute can be accessed.	See Section Flags member

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 }
}
```

2.1.2 Object type secondary_service

Reference

GATT Specification section 3.1

Synopsis

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

Supported member objects

- Characteristic Declaration

Supported members

Table 2-2 Secondary service 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 GATT database name member
`flags : value` `flags : array`	Attribute level options which control how the attribute can be accessed.	See Flags member

Example

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

```
secondary_service {
     uuid : 0x11223344556677889aabbccddeeff,
     name : "MY_OWN_SERVICE"
}
```

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

2.1.3 Object type include

Reference

GATT Specification section 3.2

Synopsis

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

Supported member objects

None

Supported members

Table 2-3 Include 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 GATT database name member
`flags : value` `flags : array`	Attribute level options which control how the attribute can be accessed.	See Flags member

Example

Include a secondary service named `"MY_OWN_SERVICE"`.

2.1.4 Object type characteristic

Reference

GATT Specification section 3.3.1

Synopsis

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

Supported member objects

- Characteristic Extended Properties
- Characteristic User Description
- Client Characteristic Configuration
- Server Characteristic Configuration

- Characteristic Presentation Format
- Characteristic Aggregate Format

Supported members

Table 2-4 Characteristic declaration 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 2-5
`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, that is, `0x12` is considered as 8-bit and `0x1234` is considered as 16-bit.	-
`size_value : number`	The length of the characteristic value in octets 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 GATT database name member
`flags : value` `flags : array`	Attribute level options which control how the attribute can be accessed.	See Flags member

The table below describes possible values for characteristic properties.

Table 2-5 Characteristic value 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.

Table 2-5 Characteristic value properties (cont.)

Name	Description
`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.

Example

Readable and notifiable `characteristic` value:

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

2.1.5 Object type extended_properties

Reference

GATT Specification section 3.3.3.1

Synopsis

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

Supported member objects

None

Supported members

Table 2-6 Characteristic extended properties 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 2-7
`name : string`	A human readable name to describe the attribute.	See GATT database name member
`flags : value` `flags : array`	Attribute level options which control how the attribute can be accessed.	See Flags member

The table below describes possible values for characteristic properties.

Table 2-7 Characteristic extended 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.

Example

A `characteristic` which allows reliable writes:

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

2.1.6 Object type user_description

Reference

GATT Specification section 3.3.3.2

Synopsis

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

Supported member objects

None

Supported members

Table 2-8 Characteristic user description 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, that is, `0x12` is considered as 8-bit and `0x1234` is considered as 16-bit.	-
`size_value : number`	The length of the value in octets if `value` member is not present. This is used to verify write lengths when `FLAG_IRQ` is set.	-

Table 2-8 Characteristic user description members (cont.)

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

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" }
}
```

2.1.7 0bject type client_configuration

Reference

GATT Specification section 3.3.3.3

Synopsis

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

Supported member objects

None

Supported members

Table 2-9 Client characteristic configuration members

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

Example

A `characteristic` with Client Configuration:

```
characteristic {
     uuid : 0x1234,
     properties :  [ notify ],
     name : "CHARACTERISTIC_NOTIFY",
     client_config {
          flags : [ FLAG_IRQ ],
          name : "CHARACTERISTIC_NOTIFY_CONFIG"
     }
}
```

2.1.8 Object type server_configuration

Reference

GATT Specification section 3.3.3.4

Synopsis

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

Supported member objects

None

Supported members

Table 2-10 Server characteristic configuration 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 GATT database name member
`flags : value` `flags : array`	Attribute level options which control how the attribute can be accessed.	See Flags member

The table below describes possible values for server characteristic configuration properties.

Table 2-11 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.

Example

A `characteristic` that may be broadcast:

```
characteristic {
     uuid : 0x1234,
     properties : [ broadcast ],
     server_config {
          flags : [ FLAG_IRQ ],
          name : "BROADCAST_CONFIG",
          properties : 0
     }
}
```

2.1.9 Object type presentation_format

Reference

GATT Specification section 3.3.3.5

Synopsis

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

Supported member objects

None

Supported members

Table 2-12 Characteristic presentation format members

Synopsis	Description	Notes
`format : value`	Characteristic value format as defined in *GATT Specification*.	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 * $10^{\texttt{exponent}}$	-
`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 GATT database name member
`flags : value` `flags : array`	Attribute level options which control how the attribute can be accessed.	See Flags member

The table below describes possible values for characteristic presentation format.

Table 2-13 Characteristic presentation formats

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

Table 2-13 Characteristic presentation formats (cont.)

Name	Description	Exponent
`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

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
     }
}
```

2.1.10 Object type aggregate_format

Reference

GATT Specification section 3.3.3.6

Synopsis

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

Supported member objects

None

Supported members

Table 2-14 Characteristic aggregate format 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 GATT database name member
`flags : value` `flags : array`	Attribute level options which control how the attribute can be accessed.	See Flags member

Example

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

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

2.2 GATT database common members

GATT database objects:

- flags
- names

2.2.1 Flags member

Synopsis

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

Description

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

Value description

Comma separated list of flag values. Table 2-15 describes possible flag values.

Table 2-15 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, that is, 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() (QTIL µEnergy projects).	This flag should be used when the application needs to take an action upon access to the attribute. If the GATT database is stored in VM constant memory (ADK 4.0 onwards) then the IRQ flag must be set for all writable attributes.

Example

To enable dynamic length and interrupt features in the attribute.

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

2.2.2 GATT database name member

Synopsis

```
name : string
```

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.

Value description

A string defining the attribute name.

Example

To set the name of the attribute to `GAP_SERVICE`:

```
name : "GAP_SERVICE"
```

3 Use of GATT databases in 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 GATT database functions for use in BlueCore projects

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);
}
```

GattGetDatabaseSize

Synopsis

```
uint16 GattGetDatabaseSize(void)
```

Description

Returns the size of the GATT database in bytes.

Provides the size of the GATT database in bytes. This is required by `GattInitEx/ GattManagerRegisterConstDB` when registering GATT databases stored in VM const memory.

Example

```
extern const uint16 gattDatabase[];
bool init_gatt(void)
{
return GattManagerRegisterConstDB(&gattDatabase[0], GattGetDatabaseSize()))
}
```

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.

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.

3.1.1 GATT database enumerated types for use in BlueCore projects

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

To get a SDP record using the `gatt_sdp` enumerated type:

```
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();
}
```

3.1.2 GATT database defined constants for use in BlueCore projects

Handle numbers of all objects with a name member are 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.

If the GATT service is implemented by the GATT Manager as a VM library then a `.dbi` file should be used to generate local handle definitions in a `.h` file. In this case the local handle numbers start at zero for each service and GATT Manager demuxes the handle numbers before passing on the message to the registered service.

3.2 GattGetDatabase function for use in QTIL Bluetooth Low Energy projects

Introduction

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.

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);
}
```

3.2.1 GATT database defined constants for use in Bluetooth Low Energy projects

Handle numbers

Handle numbers of all objects with a name member are 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.

Attribute lengths

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

A GATT database 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

Type	Notation
value	string number array
string	" " " chars "
chars	char char chars
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]+

Document references

Document	Reference
Bluetooth Assigned Numbers	www.bluetooth.org
Bluetooth Core Specification v4.1	www.bluetooth.org
GATT Specification	Volume 3, Part G of the *Bluetooth Core Specification v4.1*
JSON	RFC 4627 www.json.org
SDK Reference Documentation	Supplied with the SDK as: QTIL BlueCore SDK: *VM and Native Reference Guide* QTIL µEnergy SDK: *Firmware Library Documentation*

Terms and definitions

Term	Definition
BlueCore	Group term for the range of QTIL Bluetooth wireless technologyICs
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
GATT	Generic Attribute Profile
IC	Integrated Circuit
IEEE	Institute of Electronic and Electrical Engineers
JSON	JavaScript Object Notation
MITM	Man In The Middle
QTIL	Qualcomm Technologies International, Ltd.
SDK	Software Development Kit
SDP	Service Discovery Protocol; element of Bluetooth
UTF	Unicode Transformation Format
UUID	Universally Unique Identifier