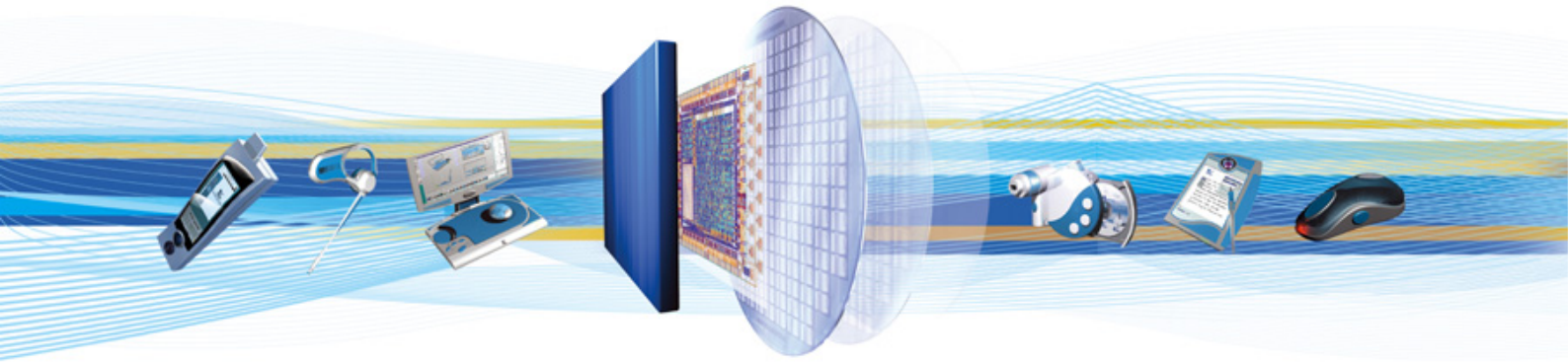
CSR Synergy Bluetooth 18.2.0

GATT – Generic Attribute Profile

API Description

November 2011



Cambridge Silicon Radio Limited

Churchill House
Cambridge Business Park
Cowley Road
Cambridge CB4 0WZ
United Kingdom

Registered in England and Wales 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction and Scope

This document describes the functionality and message interface provided by CSR Synergy Bluetooth for using the Generic Attribute Profile – generally referred to as GATT.

2 Description

2.1 Introduction

GATT, the Generic Attribute Profile, defined in the Bluetooth 4.0 core specification, aims to provide a generic database client and server interface on top of the Attribute Protocol. GATT can use both the traditional Bluetooth BR/EDR radio, and the new Low Energy (LE) radio.

GATT defines two roles: Server and Client. The Client sends commands and requests to the server to obtain information about services, while the server processes database requests from clients. Server may send indications to client. The GATT server stores the data transported over the Attribute Protocol and accepts Attribute Protocol requests, commands and confirmations from the GATT client. The GATT server sends responses to client requests and when configured to do so it may also send indications and notifications asynchronously to the GATT client.

The client and server roles are not in any way tied to the radio roles (peripheral and central or master and slave). Devices can be both client and server at the same time – in fact, devices must always have a database to contain fundamental information about the device, such as the device name.

The Synergy Bluetooth GATT implementation is designed to be used by an application or a higher-level profile and allows the client-server communication described above. Both clients and servers can be implemented using Synergy GATT.

Specifically, the server is implemented by adding entries to the device database. By exposing database entries, known as attributes, clients may retrieve information from – and write data to - servers. Clients can of course also search (discover) what services a server provides. Some services may allow the client to configure special server side behavior such as to notify clients when a value change, and to make servers broadcast data.

GATT specifies the format of data contained on the GATT server. Attributes, as transported by the Attribute Protocol, are formatted as **Services** and **Characteristics**. Services may contain a collection of characteristics. Characteristics contain a single value and any number of descriptors describing the characteristic value.

With the defined structure of services, characteristics and characteristic descriptors a GATT client that is not specific to a profile can still traverse the GATT server and display characteristic values to the user. The characteristic descriptors can be used for displaying descriptions of the characteristic values that may make the value understandable by the user.

Before both clients and servers can use GATT, they must first register with the GATT profile. Once registered, services can be added to the database, and client operations can be performed. See Figure 1 for an overview of the GATT state machine.

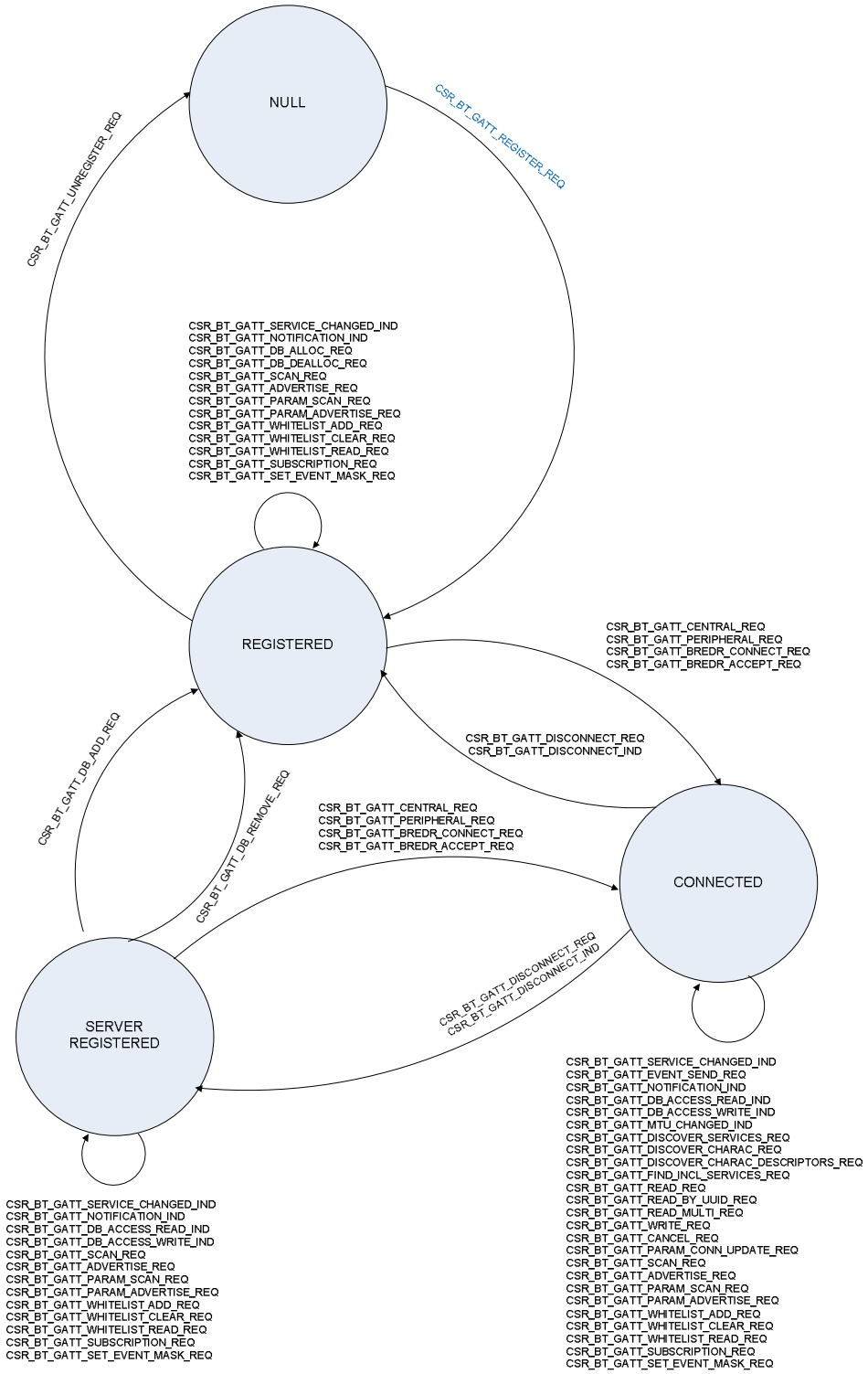


Figure 1: Overall GATT state machine

2.2 Architecture

The GATT architecture is illustrated in Figure 2. Note that the direction of the arrow represents the API provider relationship. E.g. ATT provides an API for GATT.

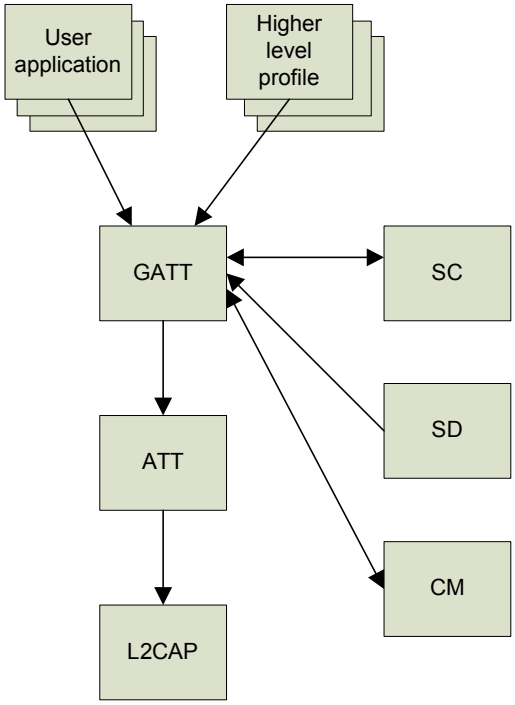


Figure 2: GATT protocol architecture

GATT is the central component, and supplies several other Synergy Bluetooth profiles with basic device services to provide a better user experience, e.g. the SD (service discovery) and SC (security controller) use GATT to obtain the device name of Bluetooth Low Energy devices where the *read remote name* function is not available.

GATT uses the ATT protocol for all signalling towards the peer, and ATT again is based upon L2CAP.

From the application point of view, GATT provides an extensive API that allows multiple clients and multiple servers to register and use GATT at the same time. For example, a user may implement three services in three different applications, and at the same time the user can use multiple client based applications towards multiple peers.

GATT uses a **GATT identifier** (type `CsrBtGattId`, common member name *gattId*) to differentiate applications. Applications register with GATT to obtain a GATT id. All future API dealings with GATT must include the unique GATT id.

On top of GATT ids, GATT uses **connection identifiers** (type `CsrBtConnId`, common name btConnId) to differentiate between different connections towards peers.

A single application has one GATT id, but may have multiple connection ids.

Note that although multiple servers may register with GATT and each server can provide database content, a device in reality has only *one* shared database. I.e. the database that GATT exposes to peer clients is the combined database from all server applications. This design allows for great flexibility and multiple user-supplied applications, however it also makes server application design relatively more complex. For example, in a situation where multiple server applications exist, a server may not be interested in client connection while another server application may be. Since there is only one system wide database, both server applications must be prepared to serve clients whether or not they have explicitly requested a connection.

Multi-server designs are also complicated due to the fact that to ensure database integrity, services must exist on the same database location (known as handle numbers) across system reboots. When multiple servers exist,

GATT cannot guarantee the ordering as GATT does not have any sort of persistent storage. It is the responsibility of the upper layer profiles and applications to ensure persistence across reboots.

The multi-server design challenge can be completely overcome by designing a single server application that has ownership of the entire database such that connections can be coordinated, and that application/handle usage can be recorded and restored from a persistent storage.

3 Interface Description

In this section a series of MSCs will be shown to explain the usage of the GATT. The primitives and the functions available to the application are also described in the subsections of this chapter.

3.1 Registration Procedures

An application is able to register and unregister itself by using the procedures described in this section.

3.1.1 Register

Before an application can be used with GATT, it shall register itself by calling `CsrBtGattRegisterReqSend`. This function will send a `CSR_BT_GATT_REGISTER_REQ` primitive to GATT. The arguments for the function are described in Table 1.

Type	Argument	Description
CsrSchedQid	Qid	Protocol handle of the higher layer entity registering with GATT
CsrUint16	Context	Registration context

Table 1: Arguments for `CsrBtGattRegisterReqSend` function

When the GATT has processed the registration request, a `CSR_BT_GATT_REGISTER_CFM` primitive will be sent back to the application. The primitive members are described in Table 2.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_REGISTER_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrUint16	context	Registration context as specified in request

Table 2: Members in a CSR_BT_GATT_REGISTER_CFM primitive

3.1.2 Un-Register

An application can be un-registered by calling `CsrBtGattUnregisterReqSend`. This function will send a `CSR_BT_GATT_UNREGISTER_REQ` primitive to GATT and takes the arguments described in Table 3.

Type	Argument	Description
CsrBtGattId	gattId	Application identifier

Table 3: Arguments for `CsrBtGattUnregisterReqSend` function

When GATT has processed the un-registration request, a `CSR_BT_GATT_UNREGISTER_CFM` primitive will be sent back to the application. The primitive members are described in Table 4.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_UNREGISTER_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 4: Members in a CSR_BT_GATT_UNREGISTER_CFM primitive

Please note that if GATT receives the `CSR_BT_GATT_UNREGISTER_REQ` primitive and the application has:

1. A BR/EDR or LE connection, or is creating one, GATT will ensure that this connection is released/cancelled, see section 3.3.
2. Added a local data base, GATT will ensure that it is removed, see section 3.6.15
3. Reserved/allocated a range of attribute handles, GATT will ensure that these are de-allocated, see section 3.6.16

Note, that if a Service has been removed from the database, the server application shall notify clients which it is bonded to. For more information, please refer to 3.6.17.

3.2 Advertise and Scan Procedures

This section describes how an application can send and receive advertising events, by using one of the following procedures:

- Advertising – broadcasting data by sending advertising events.
- Scan – receiving broadcast data contained in advertising events.

3.2.1 Advertising

A GATT application is able to send broadcast data to one or more LE devices that are scanning for broadcast data.

A GATT application can start advertising, by calling one of the following functions:

- *`CsrBtGattAdvertiseReqStartDataSend`*
- *`CsrBtGattAdvertiseReqStartSend`*

The GATT application can stop advertising again, by calling:

- *CsrBtGattAdvertiseReqStopSend*

The arguments for the three functions are described in Table 5, Table 6 and Table 7.

Note, an Advertising device can be connected to other LE devices.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattConnFlags	advertisingFlags	The advertising flags, which are defined in csr_bt_gatt_prim.h, can be set to: • CSR_BT_GATT_FLAGS_NONE - No special options. • CSR_BT_GATT_FLAGS_NONDISCOVERABLE - AD flags are non-discoverable. • CSR_BT_GATT_FLAGS_LIMITED_DISCOVERABLE - AD flags are limited discoverable. • CSR_BT_GATT_FLAGS_DISABLE_SCAN_RESPONSE - disable scan response. • CSR_BT_GATT_FLAGS_APPEND_DATA - append advertise data
CsrUint8	advertisingDataLength	Length of the advertising Data in octets.
CsrUint8	*advertisingData	The advertising data formatted as defined in [4], Section 11 – ADVERTISING AND SCAN RESPONSE DATA FORMAT.
CsrUint8	scanResponseDataLength	Length of the Scan Response Data in octets.
CsrUint8	*scanResponseData	The Scan Response Data formatted as defined in [4], Section 11 – ADVERTISING AND SCAN RESPONSE DATA FORMAT.

Table 5: Arguments for `CsrBtGattAdvertiseReqStartDataSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier

Table 6: Arguments for `CsrBtGattAdvertiseReqStartSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier

Table 7: Arguments for `CsrBtGattAdvertiseReqStopSend` function

Common to the above three functions are that they will all send a `CSR_BT_GATT_ADVERTISE_REQ` primitive to GATT, and when the procedure is completed, GATT returns a `CSR_BT_GATT_ADVERTISE_CFM` message back to the application. The parameters for `CSR_BT_GATT_ADVERTISE_CFM` are described in Table 8.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_ADVERTISE_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 8: Members in a CSR_BT_GATT_ADVERTISE_CFM primitive

A GATT application is able to change the advertise parameters, by calling:

- *CsrBtGattParamAdvertiseReqSend*

where the arguments for *CsrBtGattParamAdvertiseReqSend* are described in Table 9.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrUint16	advIntervalMin	Minimum advertising interval Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 second Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec
CsrUint16	advIntervalMax	Maximum advertising interval Range: 0x0020 to 0x4000 Default: N = 0x0800 (1.28 seconds) Time = N * 0.625 msec Time Range: 20 ms to 10.24 sec
CsrBtGattAdvChannel	advertisingChannelMap	The advertisingChannelMap parameter is a bit field that indicates the advertising channels that shall be used when transmitting advertising packets. At least one channel bit shall be set in the advertisingChannelMap parameter. The following bits are defined in csr_bt_gatt_prim.h: • CSR_BT_GATT_ADV_CHANNEL_37 - Enable channel 37. • CSR_BT_GATT_ADV_CHANNEL_38 - Enable channel 38. • CSR_BT_GATT_ADV_CHANNEL_39 - Enable channel 39. • CSR_BT_GATT_ADV_CHANNEL_ALL – Default all channels enable.

Table 9: Arguments for `CsrBtGattParamAdvertiseReqSend` function

CsrBtGattParamAdvertiseReqSend will send a `CSR_BT_GATT_PARAM_ADVERTISE_REQ` primitive to GATT, and when the procedure is completed GATT returns a `CSR_BT_GATT_PARAM_ADVERTISE_CFM` message back to the application. The parameters for `CSR_BT_GATT_PARAM_ADVERTISE_CFM` are described in Table 10.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_PARAM_ADVERTISE_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 10: Members in a CSR_BT_GATT_PARAM_ADVERTISE_CFM primitive

3.2.2 Scanning

A GATT application is able to receive broadcast data by using the scan procedure.

A GATT application can start scanning, by calling one of the following functions:

- `CsrBtGattScanReqStartSend`
- `CsrBtGattScanReqStartFilterSend`

The GATT application can stop scanning again by calling the function:

- `CsrBtGattScanReqStopSend`

The arguments for the three functions are described in Table 11, Table 12 and Table 13.

Note: A scanning device can be connected to other LE devices, but if the scanning device is a Slave in a LE-piconet the device cannot make connection requests.

Unless noted otherwise, units for all timing parameters are in Bluetooth slots. 1 BT slot = 0.625ms.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattScanFlags	flags	The scan flags, which are defined in csr_bt_gatt_prim.h.

Table 11: Arguments for CsrBtGattScanReqStartSend function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBool	enable	Toggle scanning on (True) or off (False)
CsrBtGattScanlags	Flags	The scan flags, which are defined in csr_bt_gatt_prim.h.
CsrUint8	filterAddrCount	Number of array entries in the *filterAddr* pointer
CsrBtTypeAddr	*filterAddr	List of Bluetooth addresses to pass through the GATT report filter
CsrUint8	filterDataCount	Number of array entries in the *filterData* pointer
CsrBtGattDataFilter	*filterData	List of data filters to pass GATT reports through

Table 12: Arguments for CsrBtGattScanReqStartFilterSend function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier

Table 13: Arguments for `CsrBtGattScanReqStopSend` function

The `CsrBtGattScanReqStartFilterSend` function allows the user to supply zero or more filters. Filters can be used for reducing the amount of reports an application receives:

- The *filterAddr* is an array of positive Bluetooth addresses, from which the application wants to receive reports. If the filter is *empty*, no address filtering will be done.

- The *filterData* allows the application to set up a set of data filters, which can specify what sort of report data it expects. The reports use the *Extended Inquiry Response* data format defined in the Bluetooth Core specification. Briefly, the EIR data format is a set of [length, tag, data] values, see Table 14. The data filter can be used for filtering on both 'tag' and 'data'.

Type	Member	Description
CsrUint8	adType	Data 'tag' type to pass through the filter. Tag value '0' matches all.
CsrUint8	interval	The space between each 'data' match attempt. For example, a value of 4 will try to match the 'data' pattern every 4 bytes in the report. A value of 0 disables interval matching and only looks at the beginning of the data.
CsrUint8 *	dataLength	Length of 'data' in bytes.
CsrUint8	data	Data pattern that must match every 'interval' byte.

Table 14: FilterData

Common to the three functions described in this section is that they will send a `CSR_BT_GATT_SCAN_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_SCAN_CFM` primitive, please see description in Table 15 and a `CSR_BT_GATT_REPORT_IND` primitive. The parameters for `CSR_BT_GATT_REPORT_IND` are described in Table 16.

Type	Member	Description
CsrBtGattPrim	Type	Signal identity – always set to CSR_BT_GATT_SCAN_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 15: Members in a CSR_BT_GATT_SCAN_CFM primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_REPORT_IND
CsrBtGattId	gattId	The application identifier
CsrUint8	eventType	The report event type received. This uses the CSR_BT_GATT_EVENT defines.
CsrBtTypeAddr	address	The device address of the peer device.
CsrBtTypeAddr	permanentAddress	Reserved for future use.
CsrUint8	lengthData	Length of 'data' in bytes.
CsrUint8	data	The actual report data received. The data is in the Bluetooth Core specification EIR data format.
CsrInt8	rssi	Received signal strength.

Table 16: Members in a CSR_BT_GATT_REPORT_IND primitive

The application may use the utilities function called *CsrBtGattUtilGetEirInfo*, which is defined in csr_bt_gatt_utils.h, to extract EIR type information from a CSR_BT_GATT_REPORT_IND primitive.

The timing parameters of low energy scanning can be controlled using the function:

- *CsrBtGattParamScanReqSend*.

The parameters are described in Table 17.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrUint16	scanInterval	The interval between scan attempts.
CsrUint16	scanWindow	The window length of a scan attempt.

Table 17: Arguments for CsrBtGattParamScanReqSend function

This function will send a CSR_BT_GATT_PARAM_SCAN_REQ primitive to GATT. When GATT receives this primitive, it sends back a CSR_BT_GATT_PARAM_SCAN_CFM message as confirmation. The parameters for CSR_BT_GATT_PARAM_SCAN_CFM are described in Table 18.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_PARAM_SCAN_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 18: Members in a CSR_BT_GATT_PARAM_SCAN_CFM primitive

3.3 Connection Procedures

This section describes how an application can setup a connection between two GATT devices and how an application can change the default connection parameters, by using one of the following procedures:

- Central – initiates the establishment of a LE connection.
- Peripheral – accepts the establishment of a LE connection.
- BR/EDR Connect – initiates the establishment of a BR/EDR connection.
- BR/EDR Accept Connect – accepts the establishment of a BR/EDR connection.
- Changing Connection Parameters

If the remote device supports BR/EDR/LE, the application shall either use the BR/EDR Connect or the BR/EDR Accept Connect procedure. If the remote device only supports LE the application shall either use the Central or Peripheral procedure. It is specified by a higher layer profile, if the application shall use one of the initiator procedures or one of the acceptor procedures.

Note, the application can find out if a peer device supports BR/EDR/LE or LE only by using the discovery procedure described in [3]. This procedure also retrieves the device address of the remote device.

3.3.1 Central

A GATT application can initiate the establishment of a LE connection with devices in the peripheral role, by calling the function:

- *CsrBtGattCentralReqSend*

The arguments for *CsrBtGattCentralReqSend* are described in Table 19.

Note, a device operating as Central will be in the Master Role, which means it supports multiple connections.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtTypeAddr	address	The device address of the peer device
CsrBtGattConnFlags	Flags	The connection flags, which are defined in csr_bt_gatt_prim.h, can be set to: • CSR_BT_GATT_FLAGS_NONE – No special options. • CSR_BT_GATT_FLAGS_WHITELIST – Connect to white list. • CSR_BT_GATT_FLAGS_NO_AUTO_SECURITY – Do not attempt to heighten security. • CSR_BT_GATT_FLAGS_NO_AUTO_SIGN_UPGRADE – Do not automatically allow *signed data* to *encrypted data* upgrades
CsrUint16	preferredMtu	The preferred packet size. 0 = no preference. Minimum/default is 23 bytes. If set to larger than 23 bytes and the preferredMtu size in the CSR_BT_GATT_CONNECT_IND message is 23 bytes, the application may receive a CSR_BT_GATT_MTU_CHANGED_IND message with the new negotiated packet size shortly after it has received a successful CSR_BT_GATT_CONNECT_IND message.

Table 19: Arguments for `CsrBtGattCentralReqSend` function

CsrBtGattCentralReqSend will send a CSR_BT_GATT_CENTRAL_REQ primitive to GATT. When GATT receives this primitive, it sends back a CSR_BT_GATT_CENTRAL_CFM message as confirmation, see Table

20. If CSR_BT_GATT_CENTRAL_CFM returns success, GATT will start establishing a LE connection, and the application will later receive a CSR_BT_GATT_CONNECT_IND message. This message indicates if the LE connection is established or not. The parameters for CSR_BT_GATT_CONNECT_IND are described in Table 21. Note if the application has set the 'preferredMtu' parameter higher than 23, GATT will Exchange the MTU with the peer device, just after the LE connection is established. If GATT succeeds to raise the MTU the application will receive a CSR_BT_GATT_MTU_CHANGED_IND message shortly after it has received a successful CSR_BT_GATT_CONNECT_IND message. The parameters for CSR_BT_GATT_MTU_CHANGED_IND message are described in Table 22.

The procedure for establishing a LE connection as Central, is illustrated in Figure 3.

The application is permitted to cancel this procedure after it has received a successful CSR_BT_GATT_CENTRAL_CFM message, by calling *CsrBtGattDisconnectReqSend*. When Cancelled/Disconnected the application will receive a CSR_BT_GATT_DISCONNECT_IND message. The Disconnect procedure is described in section 3.4.

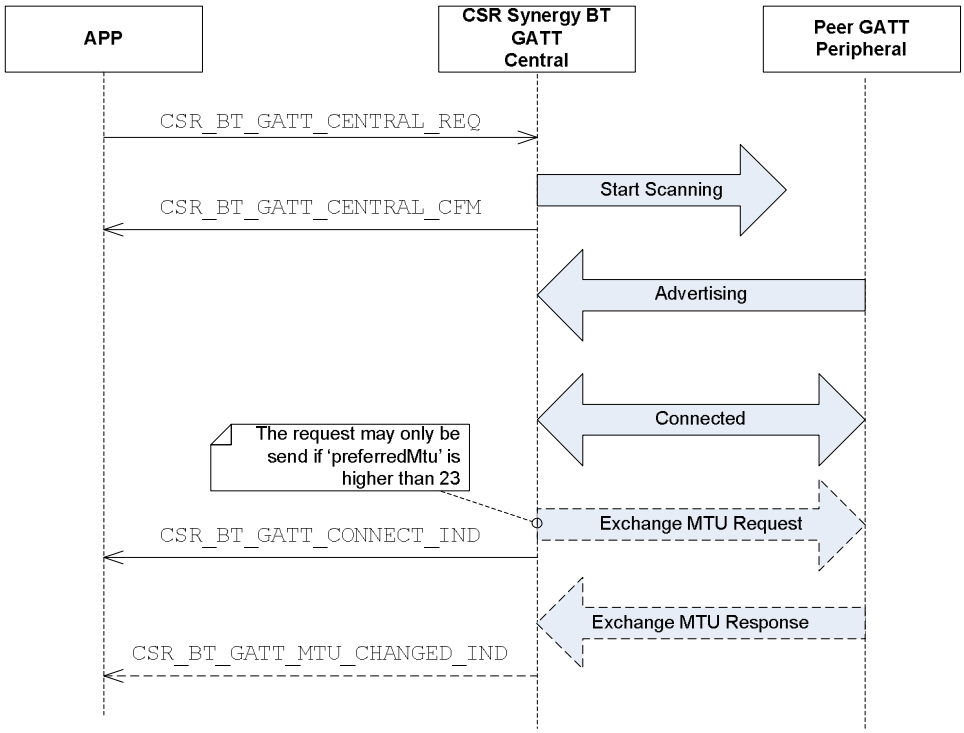


Figure 3: Establishing a LE connection as Central

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_CENTRAL_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 20: Members in a CSR_BT_GATT_CENTRAL_CFM primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_CONNECT_IND
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrBtGattConnInfo	connInfo	Connection info flags (radio type, etc.), which are defined in csr_bt_gatt_prim.h.
CsrBtTypeAddr	address	The device address of the peer device
CsrUint16	mtu	Maximum packet size

Table 21: Members in a CSR_BT_GATT_CONNECT_IND primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_MTU_CHANGED_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrUint16	mtu	MTU for the connection

Table 22: Members in a CSR_BT_GATT_MTU_CHANGED_IND primitive

3.3.2 Peripheral

A GATT application can accept the establishment of a LE connection with devices in the central role, by calling one of the following functions:

- *CsrBtGattPeripheralReqSend*
- *CsrBtGattPeripheralReqDataSend*

The arguments for the two functions are described in Table 23 and Table 24.

A device operating as Peripheral will be in the Slave Role, e.g. it can only support a single connection.

Note, that it is a break of Core Specification 4.0 conformance to use the peripheral functionality on a dual-mode device.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtTypeAddr	address	The device address of the peer device
CsrBtGattConnFlags	flags	The connection flags, which are defined in csr_bt_gatt_prim.h, can be set to: • CSR_BT_GATT_FLAGS_NONE – No special options • CSR_BT_GATT_FLAGS_WHITELIST – Connect to white list. • CSR_BT_GATT_FLAGS_UNDIRECTED – Use undirected connection • CSR_BT_GATT_FLAGS_ADVERTISE_TIMEOUT – Use undirected advertising times out • CSR_BT_GATT_FLAGS_NONDISCOVERABLE – AD flags are non-discoverable • CSR_BT_GATT_FLAGS_LIMITED_DISCOVERABLE – AD flags are limited discoverable • CSR_BT_GATT_FLAGS_NO_AUTO_SECURITY – Do not attempt to heighten security. • CSR_BT_GATT_FLAGS_NO_AUTO_SIGN_UPGRADE – Do not automatically allow *signed data* to *encrypted data* upgrades
CsrUint16	preferredMtu	The preferable packet size. 0 = no preference. Minimum/default is 23 bytes. If set to larger than 23 bytes and the preferredMtu size in the CSR_BT_GATT_CONNECT_IND message is 23 bytes, the application may receive a CSR_BT_GATT_MTU_CHANGED_IND message with the new negotiated packet size shortly after it has received a successful CSR_BT_GATT_CONNECT_IND message.

Table 23: Arguments for `CsrBtGattPeripheralReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtTypeAddr	address	The device address of the peer device
CsrBtGattConnFlags	flags	The connection flags, which are defined in csr_bt_gatt_prim.h, can be set to: • CSR_BT_GATT_FLAGS_NONE – No special options • CSR_BT_GATT_FLAGS_WHITELIST – Connect to white list. • CSR_BT_GATT_FLAGS_UNDIRECTED – Use undirected connection • CSR_BT_GATT_FLAGS_ADVERTISE_TIMEOUT – Use undirected advertising times out • CSR_BT_GATT_FLAGS_NONDISCOVERABLE – AD flags are non-discoverable • CSR_BT_GATT_FLAGS_LIMITED_DISCOVERABLE – AD flags are limited discoverable • CSR_BT_GATT_FLAGS_NO_AUTO_SECURITY – Do not attempt to heighten security. • CSR_BT_GATT_FLAGS_NO_AUTO_SIGN_UPGRADE – Do not automatically allow *signed data* to *encrypted data* upgrades
CsrUint16	preferredMtu	The preferable packet size. 0 = no preference. Minimum/default is 23 bytes. If set to larger than 23 bytes and the preferredMtu size in the CSR_BT_GATT_CONNECT_IND message is 23 bytes, the application may receive a CSR_BT_GATT_MTU_CHANGED_IND message with the new negotiated packet size shortly after it has received a successful CSR_BT_GATT_CONNECT_IND message.
CsrUint8	advDataLength	Length of the advertising data in octets
CsrUint8	*advData	The advertising data formatted as defined in [4], Section 11 – ADVERTISING AND SCAN RESPONSE DATA FORMAT
CsrUint8	scanRspDataLength	Length of the Scan Response Data in octets
CsrUint8	*scanRspData	The Scan Response Data formatted as defined in [4], Section 11 – ADVERTISING AND SCAN RESPONSE DATA FORMAT

Table 24: Arguments for `CsrBtGattPeripheralReqDataSend` function

Common for these two functions are that they will send a `CSR_BT_GATT_PERIPHERAL_REQ` primitive to GATT. When GATT receives this primitive it sends back a `CSR_BT_GATT_PERIPHERAL_CFM` message as confirmation, see Table 25. If `CSR_BT_GATT_PERIPHERAL_CFM` returns success, GATT will accept the establishment of a LE connection, and the application will later receive a `CSR_BT_GATT_CONNECT_IND` message. This message indicates if the LE connection is established or not. The parameters for `CSR_BT_GATT_CONNECT_IND` are described in Table 21. Note, if the application has set the 'preferredMtu' parameter higher than 23, GATT will Exchange the MTU with the peer device, just after the LE connection is established. If GATT succeeds to raise the MTU the application will receive a `CSR_BT_GATT_MTU_CHANGED_IND` message shortly after it has received a successful `CSR_BT_GATT_CONNECT_IND` message. The parameters for the `CSR_BT_GATT_MTU_CHANGED_IND` message are described in Table 22.

The procedure for establishing a LE connection as Peripheral, is illustrated in Figure 4.

The application is permitted to cancel this procedure after it has received a successful `CSR_BT_GATT_PERIPHERAL_CFM` message, by calling *`CsrBtGattDisconnectReqSend`*. When Cancelled/Disconnected the application will receive a `CSR_BT_GATT_DISCONNECT_IND` message. The Disconnect procedure is described in section 3.4.

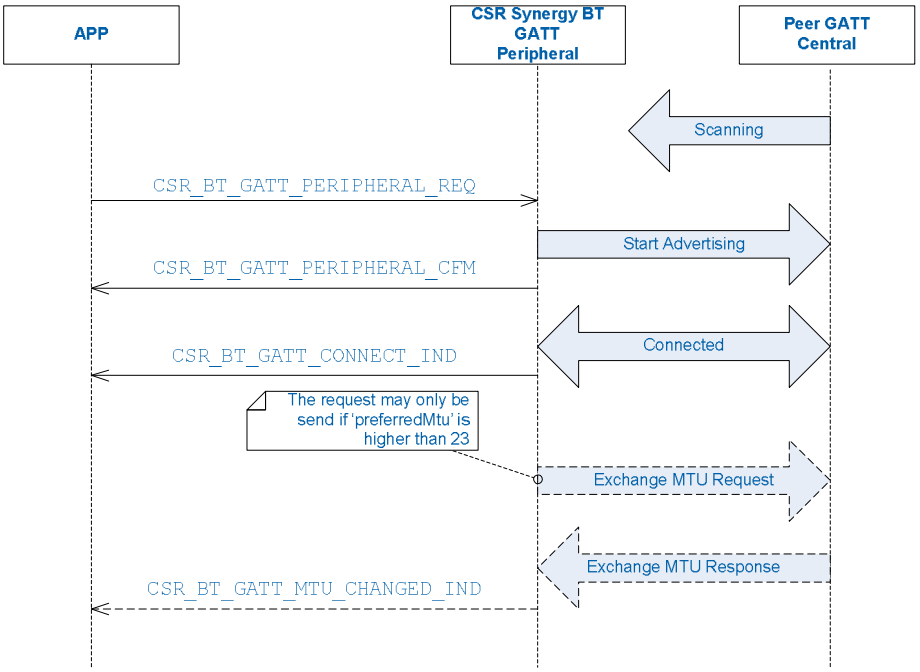


Figure 4: Accept Establishment of a LE connection as Peripheral

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_PERIPHERAL_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 25: Members in a CSR_BT_GATT_PERIPHERAL_CFM primitive

3.3.3 Bredr Connect

A GATT application can initiate the establishment of a BR/EDR connection, by calling the function:

- *CsrBtGattBredrConnectReqSend*

The arguments for *CsrBtGattBredrConnectReqSend* are described in Table 26.

Note a device running BR/EDR connection supports multiple connections.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtTypeAddr	address	The device address of the peer device
CsrBtGattConnFlags	flags	Reserved for future used, shall be set to CSR_BT_GATT_FLAGS_NONE. The connection flags are defined in csr_bt_gatt_prim.h.

Table 26: Arguments for `CsrBtGattBredrConnectReqSend` function

CsrBtGattBredrConnectReqSend will send a CSR_BT_GATT_BREDR_CONNECT_REQ primitive to GATT. When GATT receives this primitive, it sends back a CSR_BT_GATT_BREDR_CONNECT_CFM message as confirmation, see Table 27 . If CSR_BT_GATT_BREDR_CONNECT_CFM returns success, GATT will start establishing a BR/EDR connection, and the application will later receive a CSR_BT_GATT_CONNECT_IND message. This message indicates if the BR/EDR connection is established or not. The parameters for CSR_BT_GATT_CONNECT_IND are described in Table 21.

The procedure for establishing a BR/EDR connection, is illustrated in Figure 5.

The application is permitted to cancel this procedure after it has received a successful CSR_BT_GATT_BREDR_CONNECT_CFM message, by calling *CsrBtGattDisconnectReqSend*. When Cancelled/Disconnected the application will receive a CSR_BT_GATT_DISCONNECT_IND message. The Disconnect procedure is described in section 3.4.

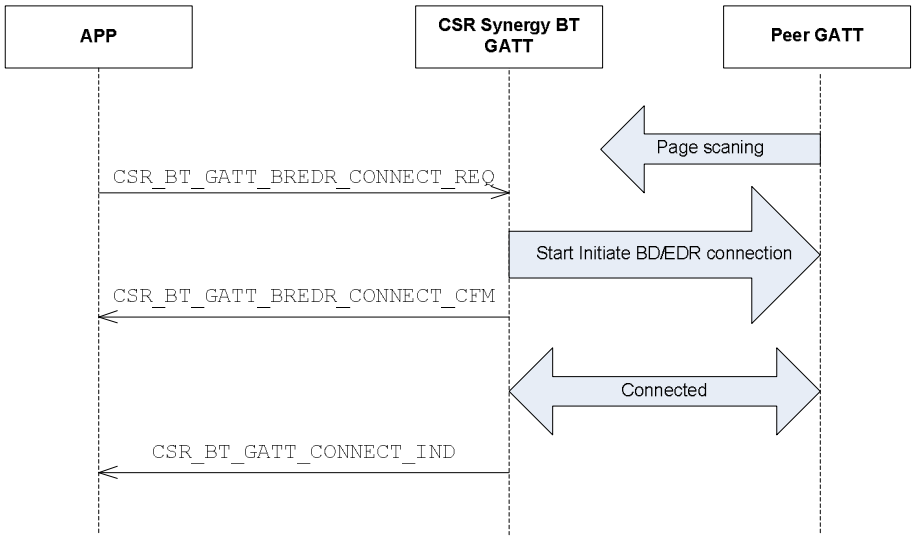


Figure 5: Establishing a BR/EDR connection

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_BRDR_CONNECT_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 27: Members in a CSR_BT_GATT_BRDR_CONNECT_CFM primitive

3.3.4 Bredr Accept

A GATT application can accept the establishment of a BR/EDR connection, by calling the function:

- *CsrBtGattBredrAcceptReqSend*

The arguments for *CsrBtGattBredrAcceptReqSend* are described in Table 28.

Note, a device running BR/EDR connection supports multiple connections.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattConnFlags	flags	Reserved for future used, shall be set to CSR_BT_GATT_FLAGS_NONE. The connection flags are defined in csr_bt_gatt_prim.h.

Table 28: Arguments for `CsrBtGattBredrAcceptReqSend` function

CsrBtGattBredrAcceptReqSend will send a CSR_BT_GATT_BREDR_ACCEPT_REQ primitive to GATT. When GATT receives this primitive, it sends back a CSR_BT_GATT_BREDR_ACCEPT_CFM message as confirmation, see Table 29. If CSR_BT_GATT_BREDR_ACCEPT_CFM returns success, GATT will accept the establishment of a BR/EDR connection, and the application will later receive a CSR_BT_GATT_CONNECT_IND message. This message indicates if the BR/EDR connection is established or not. The parameters for CSR_BT_GATT_CONNECT_IND are described in Table 21.

Note that the connection identifier returned in CSR_BT_GATT_BREDR_ACCEPT_CFM will be CSR_BT_CONN_ID_INVALID (=0) as no connection is yet established. The actual CsrBtConnId will be returned in CSR_BT_GATT_CONNECT_IND. Although the CsrBtConnId assumes the invalid value, it is still possible to cancel the BR/EDR connection accept procedure using CSR_BT_GATT_DISCONNECT_REQ.

The procedure for accepting the establishment of a BR/EDR connection, is illustrated in Figure 6.

The application is permitted to cancel this procedure after it has received a successful CSR_BT_GATT_BREDR_ACCEPT_CFM message, by calling *CsrBtGattDisconnectReqSend*. When Cancelled/Disconnected the application will receive a CSR_BT_GATT_DISCONNECT_IND message. The Disconnect procedure is described in section 3.4.

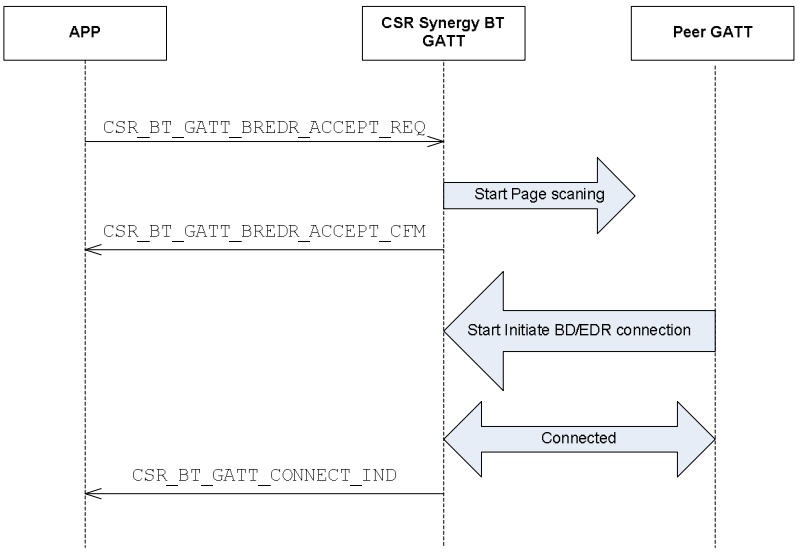


Figure 6: Accept Establishment of a BR/EDR connection

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_BRDR_ACCEPT_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.

Type	Member	Description
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier. Always zero for BR/EDR connection accept.

Table 29: Members in a CSR_BT_GATT_BRDR_ACCEPT_CFM primitive

3.3.5 Changing Connection Parameters

Both the peripheral and central of a connection may attempt to change the connection timing parameters, for example to reduce latency or to achieve lower power consumption. The connection parameters can be changed in two ways by calling the functions:

- `CsrBtGattParamConnectionReqSend`
- `CsrBtGattParamConUpdateReqSend`

The function `CsrBtGattParamConnectionReqSend` is used for setting the *default* connection parameters and `CsrBtGattParamConUpdateReqSend` is used for changing the parameters for an existing connection. The arguments for the two functions are described in Table 30 and Table 31.

Unless noted otherwise, units for all timing parameters are in Bluetooth slot - 1 BT slot = 0.625ms.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrUint16	scanInterval	The scan interval when initiating a connection as central Range: 0x0004 to 0x4000
CsrUint16	scanWindow	The scan window when initiating a connection as central Range: 0x0004 to 0x4000. Shall be less than or equal to the 'scanInterval' parameter
CsrUint16	connIntervalMin	Minimum connection interval Range: 0x0006 to 0x0C80. Shall be less than or equal to the 'connIntervalMax' parameter.
CsrUint16	connIntervalMax	Maximum connection interval Range: 0x0006 to 0x0C80. Shall be greater than or equal to the 'connIntervalMin' parameter.
CsrUint16	connLatency	Connection latency Range: 0x0000 to 0x01F3. Shall be less than or equal to the 'connLatencyMax' parameter
CsrUint16	supervisionTimeout	Supervision timeout for the LE Link. Shall be greater than or equal to the 'supervisionTimeoutMin' parameter and it shall be less than or equal to the 'supervisionTimeoutMax' parameter.
CsrUint16	connAttemptTimeout	Connection initiation timeout
CsrUint16	advIntervalMin	The minimum peripheral connection advertise interval
CsrUint16	advIntervalMax	The maximum peripheral connection advertise interval
CsrUint16	connLatencyMax	Maximum accepted connection latency. Maximum value is 0x01F3
CsrUint16	supervisionTimeoutMin	Minimum accepted link supervision timeout. Minimum value is 0x000A
CsrUint16	supervisionTimeoutMax	Maximum accepted link supervision timeout. Maximum value is 0x0C80

Table 30: Arguments for `CsrBtGattParamConnectionReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier

Type	Argument	Description
CsrUint16	connIntervalMin	Minimum connection interval. Range: 0x0006 to 0x0C80. Shall be less than or equal to the 'connIntervalMax' parameter.
CsrUint16	connIntervalMax	Maximum connection interval. Range: 0x0006 to 0x0C80. Shall be greater than or equal to the 'connIntervalMin' parameter
CsrUint16	connLatency	Slave latency for the connection in number of connection events. Range: 0x0000 to 0x01F3
CsrUint16	supervisionTimeout	Supervision timeout for the LE Link. Range: 0x000A to 0x0C80
CsrUint16	minimumCeLength	Information parameter about the minimum length of connection needed for this LE connection. How this value is used is outside the scope of this specification. Range: 0x0000 – 0xFFFF
CsrUint16	maximumCeLength	Information parameter about the maximum length of connection needed for this LE connection. How this value is used is outside the scope of this specification. Range: 0x0000 – 0xFFFF

Table 31: Arguments for `CsrBtGattParamConUpdateReqSend` function

CsrBtGattParamConnectionReqSend will send a CSR_BT_GATT_PARAM_CONNECTION_REQ primitive to GATT. When GATT receives this primitive, it sends back a CSR_BT_GATT_PARAM_CONNECTION_CFM message as confirmation. The parameters for CSR_BT_GATT_PARAM_CONNECTION_CFM are described in Table 32.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_PARAM_CONNECTION_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 32: Members in a CSR_BT_GATT_PARAM_CONNECTION_CFM primitive

CsrBtGattParamConUpdateReqSend will send a CSR_BT_GATT_PARAM_CONN_UPDATE_REQ primitive to GATT. When GATT receives this primitive, it sends back a CSR_BT_GATT_PARAM_CONN_UPDATE_CFM message as confirmation. The parameters for CSR_BT_GATT_PARAM_CONN_UPDATE_CFM are described in Table 33.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_PARAM_CON_UPDATE_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Type	Member	Description
CsrBtConnId	btConnId	Connection identifier

Table 33: Members in a CSR_BT_GATT_PARAM_CON_UPDATE_CFM primitive

Please note if GATT is Master of a LE connection and a peer Slave requests to update the connection parameters to an interval that is outside the range of which the application has defined as default, GATT will automatically reject this request. Please also note if more than one application is connected to the same physical link and one of the applications requests to change the connection parameters runtime, by calling *CsrBtGattParamConUpdateReqSend*, GATT will try to change the connection parameters. Please also note it is only the Master that can change the connection parameter runtime, the slave may request the Master to do it.

The application may decide if a request (from another local application or from a peer Slave) to change the connection parameters shall be accepted or not by subscribing for CSR_BT_GATT_EVENT_MASK_SUBSCRIBE_PARAM_CONN_UPDATE_IND events, see section 3.18. If the applications want to control if a request to change the connection parameters shall be accepted or not, it is recommended that all applications subscribe for this event.

If an application subscribes for these events, it will receive a CSR_BT_GATT_PARAM_CONN_UPDATE_IND message whenever another local application or a peer Slave requests to update the connection parameter into an interval that is outside the range of which it has registered to GATT. The parameters for CSR_BT_GATT_PARAM_CONN_UPDATE_IND are described in Table 34.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_PARAM_CONN_UPDATE_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrUint16	connIntervalMax	Connection interval max
CsrUint16	connIntervalMin	Connection interval min
CsrUint16	connLatency	Connection latency
CsrUint16	supervisionTimeout	Supervision timeout
CsrBool	incoming	Flag indicating peer-initiated (TRUE) or locally-indicated (FALSE)
CsrUint16	identifier	Used for identifying the ParamConnUpdate

Table 34: Members in a CSR_BT_GATT_PARAM_CONN_UPDATE_IND primitive

The application shall response to a CSR_BT_GATT_PARAM_CONN_UPDATE_IND by calling *CsrBtGattParamConnUpdateResSend*. Please note, the only case where an application does not need to respond is if it receives a CSR_BT_GATT_DISCONNECT_IND before it has called *CsrBtGattParamConnUpdateResSend*. The arguments for *CsrBtGattParamConnUpdateResSend* is described in Table 35.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrUint16	identifier	Used to identify the ParamConnUpdate signal
CsrBool	accept	TRUE – if paramters are acceptable, FALSE – non-acceptable

Table 35: Arguments for CsrBtGattParamConnUpdateResSend function

If the applications have subscribed for CSR_BT_GATT_EVENT_MASK_SUBSCRIBE_PARAM_CONN_UPDATE_IND events it will also receive the CSR_BT_GATT_PARAM_CONN_CHANGED_IND message whenever the actual connection parameters have changed. The parameters for CSR_BT_GATT_PARAM_CONN_CHANGED_IND are described inTable 36 .

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_PARAM_CONN_CHANGED_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrUint16	interval	Connection interval (in slots, i.e. x * 0.625ms)
CsrUint16	latency	Connection latency (in slots, i.e. x * 0.625ms)
CsrUint16	timeout	Supervision timeout (in 10ms units)

Table 36: Members in a CSR_BT_GATT_PARAM_CONN_CHANGED_IND primitive

3.4 Disconnect Procedures

This section describes how an application can release/cancel a LE or BR/EDR connection between two GATT devices, by calling the function:

- *CsrBtGattDisconnectReqSend*

The arguments for *CsrBtGattDisconnectReqSend* are described in Table 37.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier

Table 37: Arguments for `CsrBtGattDisconnectReqSend` function

This function will send a CSR_BT_GATT_DISCONNECT_REQ primitive to GATT. Later, when GATT has released or cancelled the connection, it returns a CSR_BT_GATT_DISCONNECT_IND message to the application. The parameters for CSR_BT_GATT_DISCONNECT_IND are described in Table 38.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DISCONNECT_IND
CsrBtGattId	gattId	The application identifier
CsrBtReasonCode	reasonCode	The reason code of the operation. Possible values depend on the value of reasonSupplier. If eg. the reasonSupplier == CSR_BT_SUPPLIER_GATT then the possible reason codes can be found in csr_bt_gatt_prim.h. . All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	reasonSupplier	This parameter specifies the supplier of the reason given in reasonCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrBtGattConnInfo	connInfo	Connection info flags (radio type, etc.), which are defined in csr_bt_gatt_prim.h.
CsrBtTypeAddr	address	The device address of the peer device

Table 38: Members in a CSR_BT_GATT_DISCONNECT_IND primitive

3.5 White List Procedures

The application may use white lists while advertising, scanning or setting up a LE connection. The White list tells the controller which remote devices are allowed to communicate with the local device. I.e. when using white list, transmissions from devices that are not in the white list will be ignored by the controller.

Note, since device filtering occurs in the controller it may have a significant impact on power consumption by filtering (or ignoring) advertising packets, scan requests or connection requests from being sent to the higher layers for handling. Also, note that white list filtering does not work when using the BR/EDR Radio.

In the following sections it is described how an application can:

- Add one or more devices to the white list
- Read the white list
- Clear the entire white list

Note, only white lists which must be shared between the applications exist. GATT does not coordinate this.

3.5.1 White List Add

The application can add one or more devices to the white list, by calling the function:

- *CsrBtGattWhitelistAddReqSend*

The arguments for *CsrBtGattWhitelistAddReqSend* are described in Table 39.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrUint16	addressCount	Number of address items in ‘addess’
CsrBtTypeAddr	*address	An allocated list of device addresses

Table 39: Arguments for `CsrBtGattWhiteListAddReqSend` function

This function will send a `CSR_BT_GATT_WHITELIST_ADD_REQ` primitive to GATT. Later, when GATT has added the devices to the controller white list, it returns a `CSR_BT_GATT_WHITELIST_ADD_CFM` message to the application. The parameters for `CSR_BT_GATT_WHITELIST_ADD_CFM` are described in Table 40.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_WHITELIST_ADD_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 40: Members in a CSR_BT_GATT_WHITELIST_ADD_CFM primitive

3.5.2 White List Read

The application can read the white list, by calling the function:

- *CsrBtGattWhitelistReadReqSend*

The arguments for *CsrBtGattWhitelistReadReqSend* are described in Table 41.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier

Table 41: Arguments for `CsrBtGattWhiteListReadReqSend` function

This function will send a CSR_BT_GATT_WHITELIST_READ_REQ primitive to GATT. Later, when GATT has read the white list, it returns a CSR_BT_GATT_WHITELIST_READ_CFM message to the application. The parameters for CSR_BT_GATT_WHITELIST_READ_CFM are described in Table 42.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_WHITELIST_READ_CFM
CsrBtGattId	gattId	The application identifier
CsrUint16	addressCount	Number of address items in 'address'
CsrBtTypeAddr	*address	An allocated list of device addresses.

Table 42: Members in a CSR_BT_GATT_WHITELIST_READ_CFM primitive

3.5.3 White List Clear

The application can clear the white list stored in the controller, by calling the function:

- *CsrBtGattWhitelistClearReqSend*

The arguments for *CsrBtGattWhitelistClearReqSend* are described in Table 43.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier

Table 43: Arguments for `CsrBtGattWhiteListClearReqSend` function

This function will send a CSR_BT_GATT_WHITELIST_CLEAR_REQ primitive to GATT. Later, when GATT has cleared the white list, it returns a CSR_BT_GATT_WHITELIST_CLEAR_CFM message to the application. The parameters for CSR_BT_GATT_WHITELIST_CLEAR_CFM are described in Table 44.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_WHITELIST_CLEAR_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 44: Members in a CSR_BT_GATT_WHITELIST_CLEAR_CFM primitive

3.6 Database Procedures

If an application has to act as a server is needs to create a database. In the following section it is described how this can be done. Figure 7 illustrates the step a Server application shall go though in order to do this.

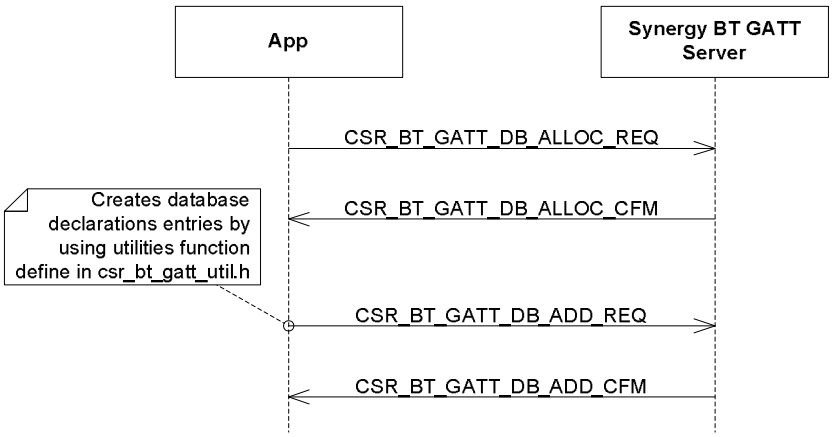


Figure 7: Initial step for creating a local database

To make it easier for the application to build a database GATT provides a set of utilities functions, which are defined in csr_bt_gatt_utils.h. These utilities functions are also described in the following sections.

Note, an application shall not create the GAP and the GATT services. These two standard services, and the characteristics exposed under them, are created by GATT, as part of its initialization procedure. Also note that all 128-Bit UUID in the following utilities functions shall be represented in Little-Endian.

3.6.1 Database Structure

A data base shall consist of one or more Services, where each Service consists of one or more Characteristics and may reference to other Services, as illustrated in Figure 8. According to [1] it is defined that:

- A Service is defined by its service definition which shall contain a service declaration and may contain include definitions and characteristic definitions. The Service definition may have zero or more include definitions. All characteristic definitions shall immediately follow the last include definition. If no include definition exists all characteristic definitions shall instead immediately follow the service declaration.

- An included service is a method to reference another service definition in the data base. The data base shall not contain a service definition with an include definition to another service that references the original service.

- A Characteristic is a value used in a service along with properties and configuration information about how the value can be addressed and information about how it can be represented. A Characteristic definition shall contain a Characteristic Declaration and a Characteristic Value Declaration. It may also contain x- number of Characteristic Descriptor Declarations, which contain related information about the Characteristic Value.

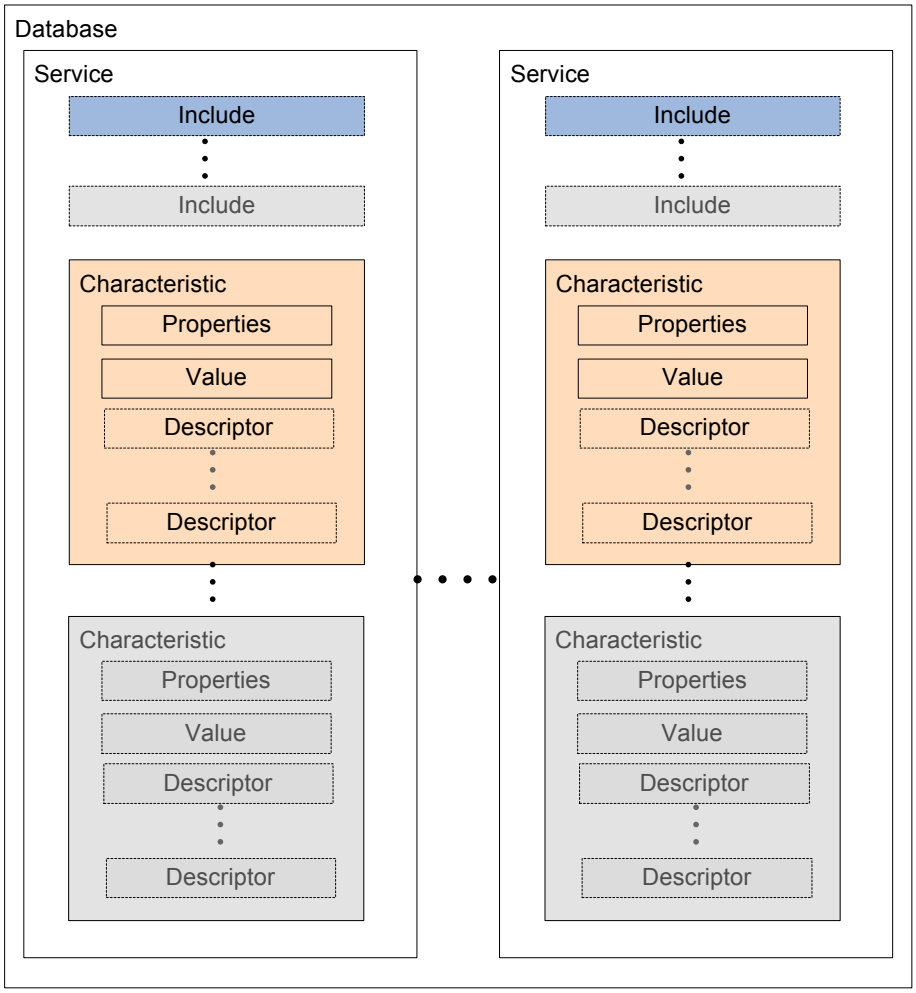


Figure 8: Data base structure

3.6.2 Database Permission

When creating the database it is possible to configure the permissions. By default GATT is designed to handle all access to the database by the client, however it is possible for the application when creating the database to add certain permissions. The permissions can be added when using the database utility functions described in section 3.6.4 to 3.6.13. The different permissions are listed in Table 45, and will be discussed in the following:

Flags	Short description
CSR_BT_GATT_ATTR_FLAGS_NONE	Default
CSR_BT_GATT_ATTR_FLAGS_DYNLEN	Allow write with different length
CSR_BT_GATT_ATTR_FLAGS_IRQ_READ	Attribute is owned by the application
CSR_BT_GATT_ATTR_FLAGS_IRQ_WRITE	Attribute can now be written without permission by app.
CSR_BT_GATT_ATTR_FLAGS_READ_ENCRYPTION	Connection needs to be encrypted for reading
CSR_BT_GATT_ATTR_FLAGS_READ_AUTHENTICATION	Client needs to be authenticated, i.e. bonded, to read the attribute
CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION	Connection needs to be encrypted for writing
CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION	Client needs to be authenticated, i.e. bonded, to write to the attribute
CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION	Client needs to be authorised to access attribute
CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE	Accessing attribute needs encryption and a certain encryption key size
CSR_BT_GATT_ATTR_FLAGS_DISABLE_LE	Attribute cannot be accessed using LE radio
CSR_BT_GATT_ATTR_FLAGS_DISABLE_BREDR	Attribute cannot be accessed using BR/EDR

Table 45: Possible permissions

CSR_BT_GATT_ATTR_FLAGS_NONE

As default, no additional parameters are added.

CSR_BT_GATT_ATTR_FLAGS_DYNLEN

By default all entries in the database do not accept to be overwritten by a value which changes its lengths. Except if CSR_BT_GATT_ATTR_FLAGS_DYNLEN is set, GATT will return a CSR_BT_GATT_RESULT_INVALID_LENGTH error.

CSR_BT_GATT_ATTR_FLAGS_READ_ENCRYPTION

When reading an attribute guarded with the CSR_BT_GATT_ATTR_FLAGS_READ_ENCRYPTION, GATT will deny access to the attribute if the connection to the client is not encrypted. It is then the job of the client to request an encryption of the link, before reading the attribute again.

CSR_BT_GATT_ATTR_FLAGS_READ_AUTHENTICATION

When reading an attribute guarded with the CSR_BT_GATT_ATTR_FLAGS_READ_AUTHENTICATION, GATT will deny access to the attribute if the client is not bonded. It is then the job of the client to bond with the server, before reading the attribute again.

CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION

When writing an attribute guarded with the CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION, GATT will deny access to the attribute if the connection to the client is not encrypted. It is then the job of the client to request an encryption of the link, before trying to write to the attribute again.

CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION

When writing an attribute guarded with the CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION, GATT will deny access to the attribute if the client is not bonded. It is then the job of the client to bond with the server before writing the attribute again.

CSR_BT_GATT_ATTR_FLAGS_DISABLE_LE

Some services should not be allowed on LE radio. By disabling the LE radio, GATT will ensure that a client using a LE link and accessing the attribute will receive an error.

CSR_BT_GATT_ATTR_FLAGS_DISABLE_BREDR

Some services should not be allowed on BR/EDR radio. By disabling the BR/EDR radio, GATT will ensure that a client using a BR/EDR link and accessing the attribute will receive an error.

CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION

The application can request that all use of an attribute should be authorised by the application. If this flag is set for an attribute, GATT will ask the application for permission every time a client accesses the attribute. Note, if this permission flag is set the application will receive a `CSR_BT_GATT_DB_ACCESS_READ_IND` or `CSR_BT_GATT_DB_ACCESS_WRITE_IND` message. These messages are described in the following sections.

CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE

It is possible to have special requirements for the size of the encryption key. However, there is no way to store information about which size is needed for the individual attributes in the database. GATT will therefore ask the application to confirm if the encryption key size is sufficient. Note, if this permission flag is set the application will receive a `CSR_BT_GATT_DB_ACCESS_READ_IND` or `CSR_BT_GATT_DB_ACCESS_WRITE_IND` message. These messages are described in the following sections.

CSR_BT_GATT_ATTR_FLAGS_IRQ_READ

The application can also ask to own the attribute, e.g. if an application wants to supply different values of the individual clients of the same attribute. In order to own the attribute, the CSR_BT_GATT_ATTR_FLAGS_IRQ_READ flag needs to be set. In this case, GATT will ask the application for permission and value every time a client wants to read the attribute, and if the client wants to write a value, GATT will supply the value to the application. It is then the task of the application to store this value and response to GATT weather the write was a success. Note, if this permission flag is set the application will receive a `CSR_BT_GATT_DB_ACCESS_READ_IND` or `CSR_BT_GATT_DB_ACCESS_WRITE_IND` message. These messages are described in the following sections.

CSR_BT_GATT_ATTR_FLAGS_IRQ_WRITE

Similarly to the owning, the attribute GATT also supports to partly own the attribute, in terms of all writes. To obtain this, CSR_BT_GATT_ATTR_FLAGS_IRQ_WRITE flag needs to be set. Here the values are still stored in the GATT database and the application is only asked for write permission. E.g. the application will receive a `CSR_BT_GATT_DB_ACCESS_WRITE_IND` message. This message is described in the following section.

Read Access Indication

As seen in the sections above, GATT needs to ask the application for support when determining the needed permission. In some cases where a Read Request from a Client requires permission from the application, GATT will send a `CSR_BT_GATT_DB_ACCESS_READ_IND` message, see Table 46. After receiving this message, the application shall respond by calling the function:

- *`CsrBtGattDbReadAccessResSend`*

As illustrated in Figure 9, this function will send a `CSR_BT_GATT_DB_ACCESS_RES` primitive to GATT, which either accepts or rejects the Read Request initiate by the Client. The arguments for *`CsrBtGattDbReadAccessResSend`* are described in Table 47.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DB_ACCESS_READ_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	attrHandle	The handle of the attribute
CsrUint16	offset	The offset of the first octet to be accessed
CsrUint16	maxRspValueLength	The maximum length that the value of the attribute must have
CsrBtGattAccessCheck	check	Special conditions that need to be checked. Bits can be: CSR_BT_GATT_ACCESS_CHECK_NONE CSR_BT_GATT_ACCESS_CHECK_AUTHORISATION CSR_BT_GATT_ACCESS_CHECK_ENCR_KEY_SIZE Note: Multiple bits can be set.
CsrBtGattConnInfo	connInfo	Information about the nature of the connection to the peer.
CsrBtTypedAddr	address	Peer address

Table 46: Members in a CSR_BT_GATT_DB_ACCESS_READ_IND primitive

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	attrHandle	The attribute handle received in the CSR_BT_GATT_DB_ACCESS_READ_IND message
CsrBtGattDbAccessRspCode	responseCode	Available response codes are described in Table 51 and defined in csr_bt_gatt_prim.h.
CsrUint16	valueLength	Length of the attribute value which has been read by the application
CsrUint8	*value	An allocated pointer of the attribute value

Table 47: Arguments for CsrBtGattDbReadAccessResSend function

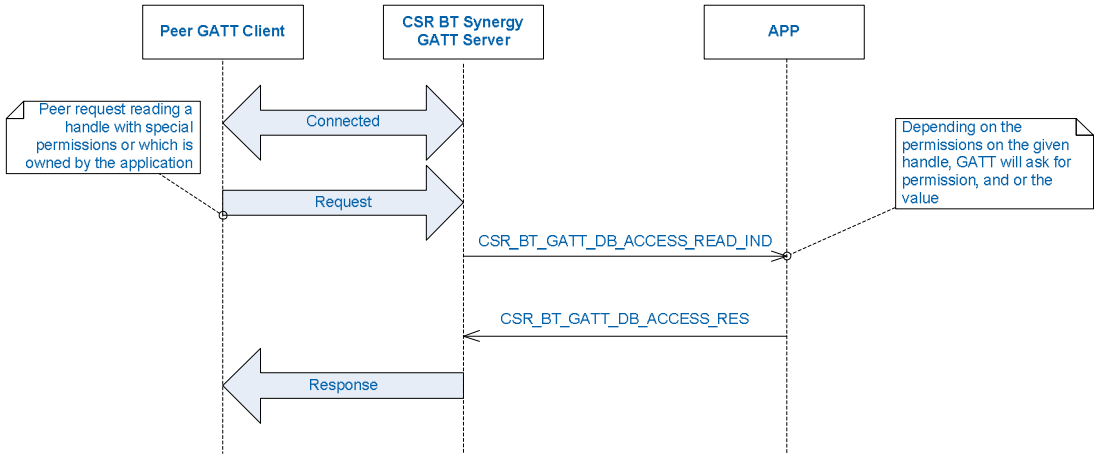


Figure 9: Accept Read Request from client

Write Access Indication

As seen in the sections above, GATT needs to ask the application for support when determining the needed permission. In some cases where a Write Request from a Client requires permission from the application, GATT will send a `CSR_BT_GATT_DB_ACCESS_WRITE_IND` message, see Table 48. After receiving this message, the application shall respond by calling the function, which arguments are described in Table 47:

- *`CsrBtGattDbWriteAccessResSend`*

In the `CSR_BT_GATT_DB_ACCESS_WRITE_IND` message, the check flags described in Table 48 are set. These flags indicate which permission checks the application needs to perform.

Because permission checks are performed before the actual data to write has been transferred, the application may receive `CSR_BT_GATT_DB_ACCESS_WRITE_IND` without any data (i.e. *writeUnit* is NULL) but with the *check* member set to e.g. CSR_BT_GATT_ACCESS_CHECK_AUTHORISATION. This is illustrated on Figure 9.

As the ATT protocol allows non-continuous data write, it is necessary to combine data into an array of *write units* when presented to the application. The `CsrBtGattAttrWritePairs` structure allows data, offset and lengths to be set together when dealing with a single attribute handle. A single `CSR_BT_GATT_DB_ACCESS_WRITE_IND` will only ever write to a single attribute handle – but the write may be split in to smaller chunks in case the peer attempts a non-continuous write. GATT will attempt to combine disparate write requests in to as few write units as possible -- but if gaps are present between write requests to the same handle, multiple write units for a single handle is necessary.

Furthermore, ATT allows multiple writes to multiple attribute handles to be grouped together in to a single atomic transaction. As GATT supports the notion of multiple server applications, it is necessary to tell all affected server applications when an atomic write transaction has finished (either successfully or not). Such an atomic transaction is in effect when the *check* flag has the `CSR_BT_GATT_ACCESS_CHECK_RELIABLE_WRITE` bit set. When this bit is set, a write operation will finish when the application receives the `CSR_BT_GATT_DB_ACCESS_COMPLETE_IND`. Only when this signal has been received should data be committed to the database. Table 50 describes the `CSR_BT_GATT_DB_ACCESS_COMPLETE_IND` signal. The atomic write procedure is illustrated on Figure 10.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DB_ACCESS_WRITE_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattAccessCheck	check	Special conditions that need to be checked. Bits can be: CSR_BT_GATT_ACCESS_CHECK_NONE CSR_BT_GATT_ACCESS_CHECK_AUTHORISATION CSR_BT_GATT_ACCESS_CHECK_ENCR_KEY_SIZE CSR_BT_GATT_ACCESS_CHECK_RELIABLE_WRITE Note: Multiple bits can be set.
CsrBtGattConnInfo	connInfo	Connection info flags: CSR_BT_GATT_CONNINFO_LE CSR_BT_GATT_CONNINFO_BREDR
CsrBtTypedAddr	address	Peer Bluetooth address.
CsrUint16	writeUnitCount	Number of entries in the *writeUnit* array
CsrBtGattAttrWritePairs*	writeUnit	Array of elements that peer is attempting to write. The structure contains the following members: • `CsrBtGattHandle attrHandle` The attribute handle being written to. This value always matches the primitive *attrHandle*. • `CsrUint16 offset` Start offset of the data • `CsrUint16 valueLength` Length of the data in *value* • `CsrUint8 *value` Pointer to the data itself.
CsrBtGattHandle	attrHandle	The handle of the attribute being written to. If data is present in the *writeUnit* member, these elements will always match this handle.

Table 48: Members in a CSR_BT_GATT_DB_ACCESS_WRITE_IND primitive

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The attribute handle received in the CSR_BT_GATT_DB_ACCESS_WRITE_IND message
CsrBtGattDbAccessRspCode	responseCode	Available response codes are described in Table 51 and defined in csr_bt_gatt_prim.h.

Table 49: Arguments for `CsrBtGattDbWriteAccessResSend` function

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DB_ACCESS_COMPLETE_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattConnInfo	connInfo	Connection info flags: CSR_BT_GATT_CONNINFO_LE CSR_BT_GATT_CONNINFO_BREDR
CsrBtGattHandle	attrHandle	The handle of the attribute.
CsrBool	commit	True only if all prepare writes succeeded

Table 50: Members in a CSR_BT_GATT_DB_ACCESS_COMPLETE_IND primitive

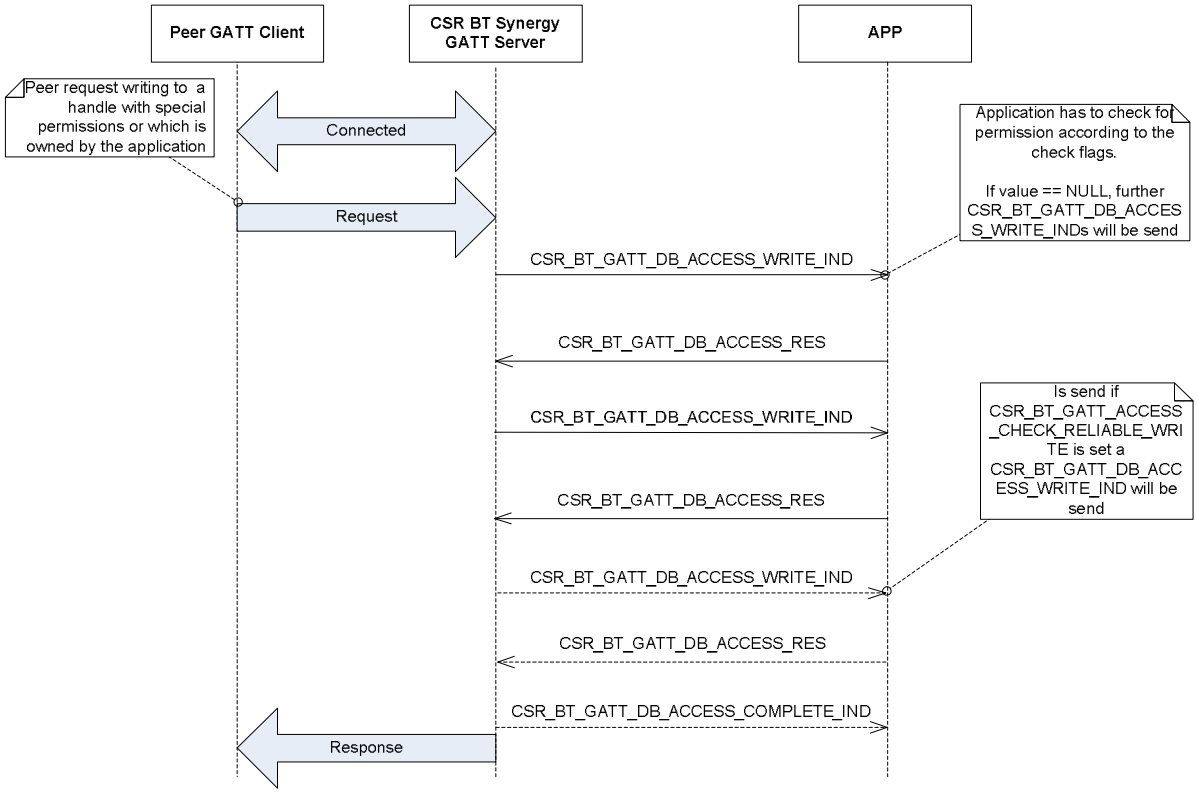


Figure 10: Accept Write Request from client

CsrBtGattDbAccessRspCode	Description
CSR_BT_GATT_ACCESS_RES_SUCCESS	The operation is accepted
CSR_BT_GATT_ACCESS_RES_INVALID_HANDLE	The attribute handle given is not valid.
CSR_BT_GATT_ACCESS_RES_READ_NOT_PERMITTED	The attribute cannot be read.
CSR_BT_GATT_ACCESS_RES_WRITE_NOT_PERMITTED	The attribute cannot be written.
CSR_BT_GATT_ACCESS_RES_INVALID_PDU	The attribute PDU was invalid.
CSR_BT_GATT_ACCESS_RES_INSUFFICIENT_AUTHENTICATION	The attribute requires authentication before it can be read or written.
CSR_BT_GATT_ACCESS_RES_REQUEST_NOT_SUPPORTED	Attribute server does not support the request received from the client.
CSR_BT_GATT_ACCESS_RES_INVALID_OFFSET	Offset specified was past the end of the attribute.
CSR_BT_GATT_ACCESS_RES_INSUFFICIENT_AUTHORISATION	The attribute requires authorization before it can be read or written.
CSR_BT_GATT_ACCESS_RES_PREPARE_QUEUE_FULL	Too many prepare writes have been queued.
CSR_BT_GATT_ACCESS_RES_ATTR_NOT_FOUND	No attribute found within the given attribute handle range.
CSR_BT_GATT_ACCESS_RES_NOT_LONG	The attribute cannot be read or written using the Read Blob Request
CSR_BT_GATT_ACCESS_RES_INSUFFICIENT_ENCR_KEY_SIZE	The Encryption Key Size used for encrypting this link is insufficient.
CSR_BT_GATT_ACCESS_RES_INVALID_LENGTH	The attribute value length is invalid for the operation.
CSR_BT_GATT_ACCESS_RES_UNLIKELY_ERROR	The attribute request that was requested has encountered an error that was unlikely, and therefore could not be completed as requested.
CSR_BT_GATT_ACCESS_RES_INSUFFICIENT_ENCRYPTION	The attribute requires encryption before it can be read or written.
CSR_BT_GATT_ACCESS_RES_UNSUPPORTED_GROUP_TYPE	The attribute type is not a supported grouping attribute as defined by a higher layer specification.
CSR_BT_GATT_ACCESS_RES_INSUFFICIENT_RESOURCES	Insufficient Resources to complete the request

Table 51: Database Access Response Codes

3.6.3 Allocate Attribute Handles

Before an application can begin to make its own database it shall reserve/allocate a range of attribute handles, by calling the function:

- *CsrBtGattDbAllocReqSend*

The arguments for *CsrBtGattDbAllocReqSend* are described in Table 38.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrUint16	numOfAttrHandles	Number of attribute handles
CsrUint16	preferredStartHandle	The StartHandle the application prefers, 0 = no preference.

Table 52: Arguments for `CsrBtGattDbAllocReqSend` function

This function will send a CSR_BT_GATT_DB_ALLOC_REQ primitive to GATT. Later, when GATT has reserved/allocated a range of attribute handles to the application, it returns a CSR_BT_GATT_DB_ALLOC_CFM message. The parameters for CSR_BT_GATT_DB_ALLOC_CFM are described in Table 38.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DB_ALLOC_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtGattHandle	start	Start attribute handle
CsrBtGattHandle	end	End attribute handle
CsrUint16	preferredStartHandle	The preferred StartHandle given in CsrBtGattAllocReqSend

Table 53: Members in a CSR_BT_GATT_DB_ALLOC_CFM primitive

After getting a range of attribute handles the application can start to build a database, by using the utilities functions described in the following sections.

3.6.4 Creating a Service Declaration

Four utilities functions exist for creating a Service Declaration:

- *CsrBtGattUtilCreatePrimaryServiceWith16BitUuid*
- *CsrBtGattUtilCreatePrimaryServiceWith128BitUuid*
- *CsrBtGattUtilCreateSecondaryServiceWith16BitUuid*
- *CsrBtGattUtilCreateSecondaryServiceWith128BitUuid*

The function *CsrBtGattUtilCreatePrimaryServiceWith16BitUuid*, which arguments are described in Table 55, creates a Primary Service Declaration with the Attribute Value set to a 16-Bit service UUID, as illustrated in Table 54.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for “Primary Service”	16- bit Bluetooth UUID	Read Only, No Authentication, No Authorization

Table 54: Service Declaration for `CsrBtGattUtilCreatePrimaryServiceWith16BitUuid`

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Primary Service Declaration must be place in the local database. Note the 'attrHandle' parameter is automatically incremented by 1.
CsrBtUuid16	service Uuid16	A 16-Bit Service UUID.
CsrBool	leOnly	If the Service Definition shall be supported over BR/EDR 'leOnly' shall be set to FALSE otherwise TRUE. If set to FALSE GATT will automatically generate and publish a generic SDP record. In order for GATT to do this a complete Service definition SHALL be created, i.e. all Include Definitions, Characteristic Definitions, and all Characteristic Descriptor Definitions SHALL be created before CSR_BT_GATT_DB_ADD_REQ is called. Note, If the application for some reason cannot Create a complete Service Definition and the Service shall be supported over BR/EDR the application shall set 'leOnly' to TRUE. The application then needs to generate and publish the SDP record by using the CM - Connection Manager API.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 55: Arguments for `CsrBtGattUtilCreatePrimaryServiceWith16BitUuid` function

The function *`CsrBtGattUtilCreatePrimaryServiceWith128BitUuid`*, of which arguments are described in Table 57, creates a Primary Service Declaration with the Attribute Value set to a 128-Bit service UUID, as illustrated in Table 56.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2800 – UUID for "Primary Service"	128- bit Bluetooth UUID	Read Only, No Authentication, No Authorization

Table 56: Service Declaration for `CsrBtGattUtilCreatePrimaryServiceWith128BitUuid`

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Primary Service Declaration must be placed in the local database. Note the 'attrHandle' parameter is automatically incremented by 1.
CsrBtUuid128	service Uuid128	A 128-Bit Service UUID.
CsrBool	leOnly	If the Service Definition shall be supported over BR/EDR 'leOnly' shall be set to FALSE otherwise TRUE. If set to FALSE GATT will automatically generate and publish a generic SDP record. In order for GATT to do this a complete Service definition SHALL be created, i.e. all Include Definitions, Characteristic Definitions, and all Characteristic Descriptor Definitions SHALL be created before CSR_BT_GATT_DB_ADD_REQ is called. Note, If the application for some reason cannot Create a complete Service Definition and the Service shall be supported over BR/EDR the application shall set 'leOnly' to TRUE. The application then needs to generate and publish the SDP record by using the CM - Connection Manager API.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 57: Arguments for `CsrBtGattUtilCreatePrimaryServiceWith128BitUuid` function

The function *CsrBtGattUtilCreateSecondaryServiceWith16BitUuid*, of which arguments are described in Table 59, creates a Secondary Service Declaration with the Attribute Value set to a 16-Bit service UUID, as illustrated in Table 58.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2801 for “Secondary Service”	16- bit Bluetooth UUID	Read Only, No Authentication, No Authorization

Table 58: Service Declaration for `CsrBtGattUtilCreateSecondaryServiceWith16BitUuid`

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Secondary Service Declaration must be placed in the local database. Note the ‘attrHandle’ parameter is automatically incremented by 1.
CsrBtUuid16	serviceUuid16	A 16-Bit Service UUID
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 59: Arguments for `CsrBtGattUtilCreateSecondaryServiceWith16BitUuid` function

The function *CsrBtGattUtilCreateSecondaryServiceWith128BitUuid*, of which arguments are described in Table 61, creates a Secondary Service Declaration with the Attribute Value set to a 128-Bit service UUID as illustrated in Table 60.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permission
0xNNNN	0x2801 for “Secondary Service”	128- bit Bluetooth UUID	Read Only, No Authentication, No Authorization

Table 60: Service Declaration for `CsrBtGattUtilCreateSecondaryServiceWith128BitUuid`

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Secondary Service Declaration must be placed in the local database. Note the ‘attrHandle’ parameter is automatically incremented by 1.
CsrBtUuid128	uuid128	A 128-Bit Service UUID
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 61: Arguments for `CsrBtGattUtilCreateSecondaryServiceWith128BitUuid` function

3.6.5 Creating a Include Declaration

A Service may contain zero or more include definitions, which shall (if any) immediately follow the Service Declaration. Two utilities functions exist for creating an Include Declaration:

- *CsrBtGattUtilCreateIncludeDefinitionWithUuid*
- *CsrBtGattUtilCreateIncludeDefinitionWithoutUuid*

The function *CsrBtGattUtilCreateIncludeDefinitionWithUuid*, of which arguments are described in Table 63, creates an Include Declaration with the a 16-Bit service UUID, as illustrated in Table 62.

Attribute Handle	Attribute Type	Attribute Value			Attribute Permission
0xNNNN	0x2802 – UUID for “Include”	Included Service Attribute Handle	End Group Handle	Service UUID	Read Only, No Authentication, No Authorization

Table 62: Include Declaration with Service UUID

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Include Declaration must be placed in the local database. Note the ‘attrHandle’ parameter is automatically incremented by 1.
CsrBtGattHandle	inclServiceAttrHandle	The attribute handle/index of where the included service starts
CsrBtGattHandle	endGroupHandle	The attribute handle/index of where the included service stops
CsrBtUuid16	serviceUuid	The UUID of the included service
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 63: Arguments for CsrBtGattUtilCreateIncludeDefinitionWithUuid function

The function *CsrBtGattUtilCreateIncludeDefinitionWithoutUuid*, of which arguments are described in Table 65, creates an Include Declaration without any service UUID, as illustrated in Table 64. Note a Service UUID shall only be present if it is 16-Bit Bluetooth UUID.

Attribute Handle	Attribute Type	Attribute Value		Attribute Permission
0xNNNN	0x2802 – UUID for “Include”	Included Service Attribute Handle	End Group Handle	Read Only, No Authentication, No Authorization

Table 64: Include Declaration without Service UUID

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Include Declaration must be placed in the local database. Note the ‘attrHandle’ parameter is automatically incremented by 1.
CsrBtGattHandle	inclServiceAttrHandle	The attribute handle/index of where the included service starts
CsrBtGattHandle	endGroupHandle	The attribute handle/index of where the included service stops
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 65: Arguments for CsrBtGattUtilCreateIncludeDefinitionWithoutUuid function

3.6.6 Creating a Characteristic and a Characteristic Value Declaration

A Service may also contain zero or more characteristic definitions, which shall (if any) immediately follow the last include definition. If no include definition exists all characteristic definitions shall instead immediately follow the service declaration. Within a Service definition, some characteristics may be mandatory and those characteristics shall be located before any optional characteristic within the Service definition.

A Characteristic Definition shall contain a Characteristic Declaration, and a Characteristic Value Declaration. A Characteristic Definition may also contain Characteristic Descriptors declarations. How an application can create Characteristic Descriptors declarations is explained in the following sections.

Two utilities functions exist for creating the mandatory part of a Characteristic Definition, e.g. a Characteristic Declaration, and a Characteristic Value Declaration.

- `CsrBtGattUtilCreateCharacDefinitionWith16BitUuid`
- `CsrBtGattUtilCreateCharacDefinitionWith128BitUuid`

These functions, of which arguments are described in Table 68 and Table 69, create a Characteristic Declaration and a Characteristic Value Declaration with a 16-Bit or 128-Bit characteristic UUID, as illustrated in Table 66 and Table 67. Note the Characteristic Declaration Attribute Value, e.g. the Characteristic Properties, the Characteristic Value Attribute Handle and the Characteristic UUID, shall not change while the server has a trusted relationship (bonded) with any client. If these values change the server shall notify the clients. For more information, please refer to section 3.6.17.

Attribute Handle	**Attribute Types**	**Attribute Value**			**Attribute Permissions**
0xNNNN	0x2803 – UUID for “Characteristic”	Characteristic Properties	Characteristic Value Attribute Handle	16 or 128-Bit Characteristic UUID.	Read Only, No Authentication, No Authorization

Table 66: Characteristic Declaration

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 67: Characteristic Value Declaration

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Characteristic Declaration and the Characteristic Value Declaration must be placed in the local database. The Characteristic Value Declaration will be placed right after (index + 1) the Characteristic Declaration. Note the 'attrHandle' parameter is automatically incremented by 2.
CsrBtGattPropertiesBits	properties	The Characteristic Properties bits determine how the Characteristic Value can be use. The Properties bits, which is describe in Table 70, are define in csr_bt_gatt_prim.h.
CsrBtUuid16	uuid16	A 16-Bit UUID for the Characteristic Value
CsrBtGattAttrFlags	attrValueFlags	The attribute flag field defines how the Characteristic Value Declaration can be accessed. The attribute flags are defined in csr_bt_gatt_prim.h and can be set to: CSR_BT_GATT_ATTR_FLAGS_NONE CSR_BT_GATT_ATTR_FLAGS_DYNLEN CSR_BT_GATT_ATTR_FLAGS_IRQ_READ CSR_BT_GATT_ATTR_FLAGS_IRQ_WRITE CSR_BT_GATT_ATTR_FLAGS_READ_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_READ_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION CSR_BT_GATT_ATTR_FLAGS_DISABLE_LE CSR_BT_GATT_ATTR_FLAGS_DISABLE_BREDR Note multiple attribute flags can be set. For more information refer to section.3.6.2.
CsrUint16	attrValueLength	The length of the Attribute Value in octets
const CsrUint8	*attrValue	The Value of the characteristic. Note, this value shall be const data, i.e. not an allocated pointer.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 68: Arguments for CsrBtGattUtilCreateCharacDefinitionWith16BitUuid function

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Characteristic Declaration and the Characteristic Value Declaration must be placed in the local database. The Characteristic Value Declaration will be placed right after (index + 1) the Characteristic Declaration. Note the 'attrHandle' parameter is automatically incremented by 2.
CsrBtGattPropertiesBits	properties	The Characteristic Properties bits determine how the Characteristic Value can be use. The Properties bits, which is describe in Table 70, are define in csr_bt_gatt_prim.h.
CsrBtUuid128	uuid128	A 128-Bit UUID for the Characteristic Value
CsrBtGattAttrFlags	attrValueFlags	The attribute flag field defines how the Characteristic Value Declaration can be accessed. The attribute flags are defined in csr_bt_gatt_prim.h and can be set to: CSR_BT_GATT_ATTR_FLAGS_NONE CSR_BT_GATT_ATTR_FLAGS_DYNLEN CSR_BT_GATT_ATTR_FLAGS_IRQ_READ CSR_BT_GATT_ATTR_FLAGS_IRQ_WRITE CSR_BT_GATT_ATTR_FLAGS_READ_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_READ_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION CSR_BT_GATT_ATTR_FLAGS_DISABLE_LE CSR_BT_GATT_ATTR_FLAGS_DISABLE_BREDR Note multiple attribute flags can be set. For more information refer to section 3.6.2.
CsrUint16	attrValueLength	The length of the Attribute Value in octets
const CsrUint8	*attrValue	The Value of the characteristic. Note, this value shall be const data, i.e. not an allocated pointer.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 69: Arguments for CsrBtGattUtilCreateCharacDefinitionWith128BitUuid function

The 'properties' parameter is a bit field that determines how the Characteristic Value can be used, or how the Characteristic Descriptors can be accessed. The properties bits are defined in Table 70. If the properties bits are

set the action described is permitted. Note multiple Characteristic Properties can be set, and the bits shall be set according to the procedures this characteristic supports, without regard to security requirements.

Properties	Value	Description
Broadcast	CSR_BT_GATT_CHARAC_PROPERTIES_BROADCAST (0x01)	If set, permits broadcasts in the Characteristic Value using the Server Characteristic Configuration Declaration. For more information refer to section 3.6.10
Read	CSR_BT_GATT_CHARAC_PROPERTIES_READ (0x02)	If set, permits reads of the Characteristic Value using procedures defined in section 4.8 in the Generic Attribute Profile Specification [1] See also section 3.11 in this document.
Write Without Response	CSR_BT_GATT_CHARAC_PROPERTIES_WRITE_WITHOUT_RESPONSE (0x04)	If set, permits writes of the Characteristic Value without response using procedures defined in section 4.9.1 the Generic Attribute Profile Specification[1] See also section 3.13.1 in this document.
Write	CSR_BT_GATT_CHARAC_PROPERTIES_WRITE (0x08)	If set, permits writes of the Characteristic Value with response using procedures defined in section 4.9.3 or 4.9.4. in the Generic Attribute Profile Specification[1] See also section 3.13.1 in this document.
Notify	CSR_BT_GATT_CHARAC_PROPERTIES_NOTIFY (0x10)	If set, permits notifications of a Characteristic Value without acknowledgement using the procedure defined in section 4.10. in the Generic Attribute Profile Specification [1]. See also section 3.6.9 and section 3.14.3 in this document.
Indicate	CSR_BT_GATT_CHARAC_PROPERTIES_INDICATE (0x20)	If set, permits indications of a Characteristic Value with acknowledgement using the procedure defined in section 4.11 in the Generic Attribute Profile Specification [1] See also section 3.6.9 and section 3.14.3 in this document.
Authenticated Signed Writes	CSR_BT_GATT_CHARAC_PROPERTIES_AUTH_SIGNED_WRITES (0x40)	If set, permits signed writes to the Characteristic Value using the procedure defined in section 4.9.2 in the Generic Attribute Profile Specification [1]. See also section 3.13.2 in this document.
Extended Properties	CSR_BT_GATT_CHARAC_PROPERTIES_EXTENDED_PROPERTIES (0x80)	If set, additional characteristic properties are defined in the Characteristic Extended Properties Descriptor defined in section 3.3.3.1 in the Generic Attribute Profile Specification[1]. See also section 3.6.7 in this document.

Table 70: Characteristic Properties bit field

3.6.7 Creating a Characteristic Extended Properties Declaration

The Characteristic Extended Properties declaration is a descriptor that defines additional Characteristic Properties. If the 'Extended Properties' (0x80) bit of the Characteristic Properties is set, see Table 70, then this characteristic descriptor shall exist. This characteristic descriptor may occur in any position within the characteristic definition after the Characteristic Value.

The utilities function

- *CsrBtGattUtilCreateCharacExtProperties*

of which arguments are described in Table 72, creates a Characteristic Extended Properties declaration, as illustrated in Table 71. Note, only one Characteristic Extended Properties declaration shall exist in a characteristic definition.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2900 – UUID for "Characteristic Extended Properties"	Characteristic Extended Properties Bit Field	Read Only, No Authentication, No Authorization

Table 71: Characteristic Extended Properties declaration

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle I.e. the index of where the Characteristic Extended Properties Declaration must be placed in the local database. Note the 'attrHandle' parameter is automatically incremented by 1.
CsrBtGattExtProperties Bits	extProperties	The Extended Properties bits, which are described in Table 73, are defined in csr_bt_gatt_prim.h.
CsrBtGattDb	**tail	The tail pointer, a reference pointer pointing to the last element in the linked list.

Table 72: Arguments for CsrBtGattUtilCreateCharacExtProperties function

The 'extProperties' parameter is a bit field that describes additional properties on how the Characteristic Value can be used, or how the characteristic descriptors can be accessed. If the bits defined in Table 73 are set, the action described is permitted. Note multiple Characteristic Properties can be set.

Properties	Value	Description
Reliable Write	CSR_BT_GATT_CHARAC_EXT_PROPERTIES_RELIABLE_WRITE (0x0001)	If set, permits reliable write of the Characteristic Value. For more information refer to section 3.13.4.
Writable Auxiliaries	CSR_BT_GATT_CHARAC_EXT_PROPERTIES_WRITE_AUX (0x0002)	If set, permits writes to Characteristic User Description declaration defined in section 3.6.8.
Reserved for Future Use	0xFFFC	Reserved for Future Use

Table 73: Characteristic Extended Properties bit field

3.6.8 Creating a Characteristic User Description Declaration

The Characteristic User Description declaration is an optional characteristic descriptor that defines a UTF-8 string of variable size that is a user textual description of the Characteristic Value. If the Writable Auxiliary bit of the Characteristic Extended Properties is set, see section 3.6.7, then this characteristic descriptor shall exist and the 'attrPermission' parameter shall set the CSR_BT_GATT_PERM_FLAGS_WRITE flag.

The utilities function

- *CsrBtGattUtilCreateCharacUserDescription*

of which arguments are described in Table 75, creates a User Description declaration, as illustrated in Table 74.

Note this characteristic descriptor may occur in any position within the characteristic definition after the Characteristic Value, and only one Characteristic User Description declaration shall exist in a characteristic definition.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2901 – UUID for "Characteristic User Description"	Characteristic User Description UTF-8 String	Higher layer profile or implementation specific

Table 74: Characteristic User Description declaration

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Characteristic Extended User Descriptor Declaration must be placed in the local database. Note the 'attrHandle' parameter is automatically incremented by 2.
CsrUtf8String	*userDescription	A NULL terminated UTF8 string describing the Charateristic Value. Note this value shall be const data, i.e. not an allocated pointer.
GattPermFlags	attrPermission	The permission flag field is used for determining whether read or write access is permitted or not. The permission flags are defined in csr_bt_gatt_prim.h and can be set to: CSR_BT_GATT_PERM_FLAGS_READ CSR_BT_GATT_PERM_FLAGS_WRITE
GattAttrFlags	attrFlags	The attribute flag field defines how User Description Declaration can be accessed. The attribute flags are defined in csr_bt_gatt_prim.h can be set to: CSR_BT_GATT_ATTR_FLAGS_NONE CSR_BT_GATT_ATTR_FLAGS_DYNLEN CSR_BT_GATT_ATTR_FLAGS_IRQ_READ CSR_BT_GATT_ATTR_FLAGS_IRQ_WRITE CSR_BT_GATT_ATTR_FLAGS_READ_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_READ_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE. Note, multiple attribute flags can be set. For more information refer to section 3.6.2
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 75: Arguments for CsrBtGattUtilCreateCharacUserDescription function

3.6.9 Creating a Client Characteristic Configuration Declaration

The Client Characteristic Configuration declaration is an optional characteristic descriptor that defines how a specific client may configure a characteristic. Only one Client Characteristic Configuration Declaration may exist within a Characteristic definition, and it may occur in any position after the Characteristic Value.

In order for the Server application to accept that the Client sets this descriptor value, the Server application shall set either the 'Notify' (0x10), the 'Indicate' (0x20), or both bits when creating the Characteristic Declaration, see section 3.6.6 Table 70.

The utilities function

- *CsrBtGattUtilCreateClientCharacConfiguration*

ofwhich arguments are described in Table 77, creates a Client Characteristic Configuration declaration, as illustrated in Table 76.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2902 – UUID for "Client Characteristic Configurations"	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 76: Client Characteristic Configuration declaration

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Characteristic Extended User Descriptor Declaration must be place in the local database. Note the 'attrHandle' parameter is automatically incremented by 1.
CsrBtGattAttrFlags	attrFlags	The attribute flag field defines how the Client Characteristic Configuration Declaration can be accessed. The attribute flags are defined in csr_bt_gatt_prim.h can be set to: CSR_BT_GATT_ATTR_FLAGS_NONE CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE. Note, multiple attribute flags can be set. For more information refer to section 3.6.2.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 77: Arguments for CsrBtGattUtilCreateClientCharacConfiguration function

Note, multiple clients can connect to the server, and as only a single instance of the Client Characteristic Configuration can exist, the application has to own and handle the value in order to provide an individual Client Characteristic Configuration Descriptor Value for each connected client. I.e. reads of this Value shall only show the Client Characteristic Configuration Descriptor Value for that client and write only affects the configuration of that client. The Client Characteristic Configuration Descriptor Value shall also be persistent across connections for bonded devices. For non-bonded devices the Client Characteristic Configuration Descriptor Value shall be responded with CSR_BT_GATT_CLIENT_CHARAC_CONFIG_DEFAULT, i.e. the Client Characteristic Configuration Declaration is not in use.

As the Client Characteristic Configuration descriptor Value can have many different Values and it shall be persistent across connections this Value is always own by the application, e.g. if a Client tries to read the Client Characteristic Configuration Declaration the application will receive a CSR_BT_GATT_DB_ACCESS_READ_IND message which it shall respond to by calling *CsrBtGattDbReadAccessResSend*. Similarly, if a Client tries to write to the Client Characteristic Configuration Declaration the application will receive a CSR_BT_GATT_DB_ACCESS_WRITE_IND message, which it shall respond to by calling *CsrBtGattDbWriteAccessResSend*. For more information about these functions and indication messages, refer to section 3.6.2.

If the client has configured the server to send a notification or an indication to the client the server application shall send an event to the client when the Characteristic Value is changed. The GATT Server is able to send event messages to a client, by calling one of the following functions:

- *CsrBtGattNotificationEventReqSend*
- *CsrBtGattIndicationEventReqSend*

where *CsrBtGattNotificationEventReqSend* sends a Characteristic Value notification, and *CsrBtGattIndicationEventReqSend* sends a Characteristic Value indication, to the Client. The arguments for the two functions are described in Table 78.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	attrHandle	The characteristic value handle being notified
CsrUint16	valueLength	Length of the attribute value that must be sent to Client
CsrUint8	*value	An allocated pointer of the attribute value

Table 78: Arguments for `CsrBtGattNotificationEventReqSend` and `CsrBtGattIndicationEventReqSend` functions

Common to all these functions are that they will send a CSR_BT_GATT_EVENT_SEND_REQ primitive to GATT. When the procedure is completed, GATT returns a CSR_BT_GATT_EVENT_SEND_CFM message to the application. The parameters for CSR_BT_GATT_EVENT_SEND_CFM are described in Table 79.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_EVENT_SEND_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier.

Table 79: Members in a CSR_BT_GATT_EVENT_SEND_CFM primitive

Figure 11 illustrates what a server application shall do when a client has configured the server to send a notification or an indication to it.

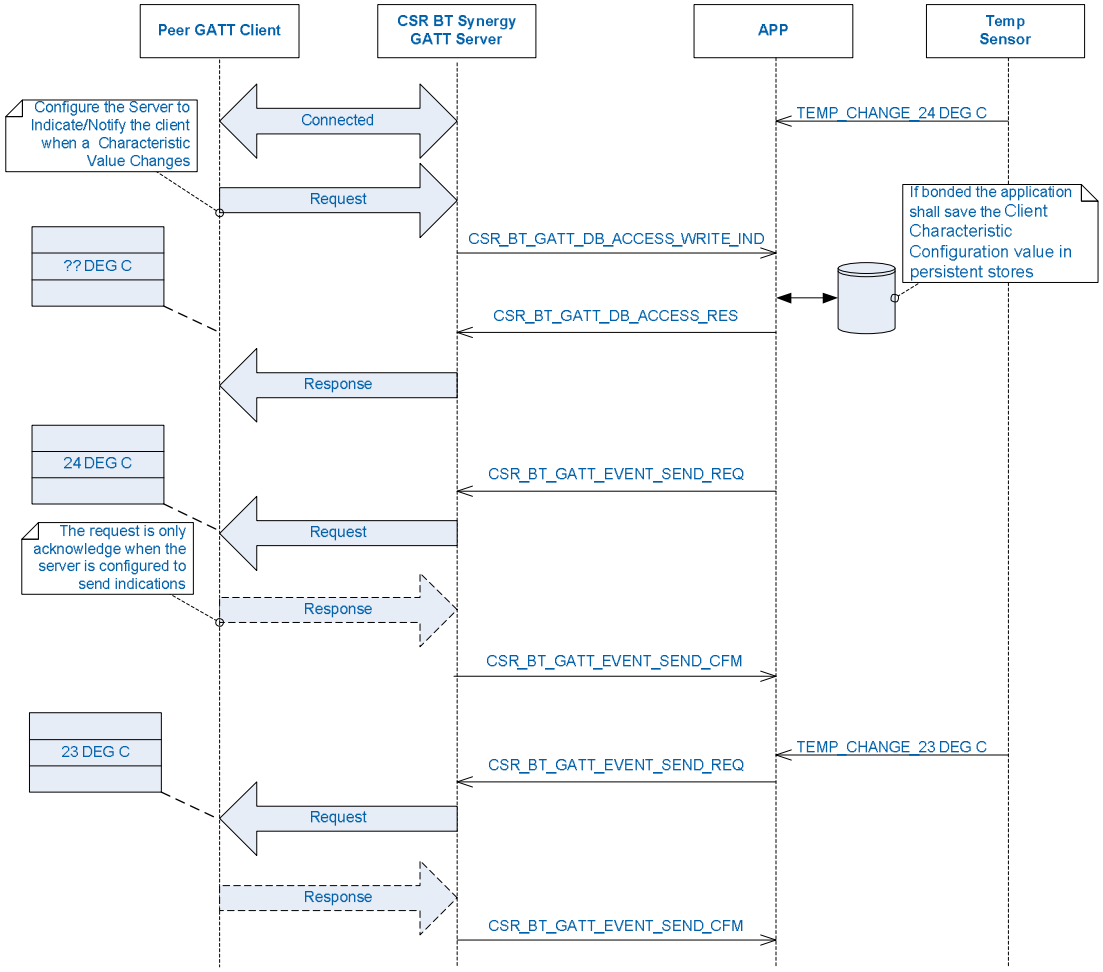


Figure 11: Notifying/Indicate a Characteristic Value from a Server to a Client

Note, if the connection is released and the client has configured the server to send a notification or indication to the client, the server shall re-establish the connection with that client when an event operation causes a notification or an indication to the client.

If the server is disconnected, but intends to become a Peripheral in a connection, it shall use one of the procedures described in section 3.3.2 or section 3.3.3 to setup the connection. The other way around, if the server is disconnected, however intends to become a Central in the connection it shall use one of the procedures described in section 3.3.1 or section 3.3.4 to setup the connection.

If the server cannot re-establish the connection, then the notification or indication for this event shall be discarded and no further connection re-establishment shall occur, until another event occurs.

3.6.10 Creating a Server Characteristic Configuration Declaration

The Server Characteristic Configuration Declaration is an optional Characteristic descriptor that defines how the characteristic may be configured for the server. Only one Server Characteristic Configuration Declaration may exist within a Characteristic definition, and it may occur in any position after the Characteristic Value.

In order for the Server application to accept that the Client sets this descriptor value, the Server application shall set the 'Broadcast' (0x01) bit when creating the Characteristic Declaration, see section 3.6.6 Table 70.

The utilities function

- `CsrBtGattUtilCreateServerCharacConfiguration`

of which arguments are described in Table 81, creates a Server Characteristic Configuration declaration, as illustrated in Table 80.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2903 – UUID for “Server 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 80: Server Characteristic Configuration declaration

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Characteristic Extended User Descriptor Declaration must be placed in the local database. Note the ‘attrHandle’ parameter is automatically incremented by 1.
CsrBtGattAttrFlags	attrFlags	The attribute flag field defines how the Server Characteristic Configuration Declaration can be accessed. The attribute flags are defined in csr_bt_gatt_prim.h can be set to: CSR_BT_GATT_ATTR_FLAGS_NONE CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE. Note multiple attribute flags can be set. For more information refer to section 3.6.2.
CsrBtGattSrvConfigBits	configurationBits	The Server Characteristic Configuration bits, which is describe is Table 163, are define in csr_bt_gatt_prim.h.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 81: Arguments for CsrBtGattUtilCreateServerCharacConfiguration function

Note, multiple clients may write this configuration descriptor to control the configuration of this characteristic on the server for all clients. As there is only one instantiation of the Server Characteristic Configuration for all clients, reads of the Server Characteristic Configuration shows the configuration of all clients and writes affects the configuration for all clients.

As the Server application is responsible of executing/controlling the broadcast procedure the application will receive a `CSR_BT_GATT_DB_ACCESS_WRITE_IND` message, every time a Client tries to write to the Server Characteristic Configuration Declaration. The application shall respond the `CSR_BT_GATT_DB_ACCESS_WRITE_IND` message by calling CsrBtGattDbWriteAccessResSend. For more information about this function and the indication message, refer to section 3.6.2.

If the client configures the server to start broadcasting the Characteristic Value, the server application shall start advertising the Characteristic Value, as illustrated in Figure 12, by calling *`CsrBtGattAdvertiseReqStartDataSend`*, which is described in section 3.2.1.

Likewise, if a Client configures the server to stop broadcasting the Characteristic Value the server application shall stop advertising by calling *`CsrBtGattAdvertiseReqStopSend`*, which is also described in section 3.2.1. Note a server application may advertise more than one Characteristic Value. If a Client configures the server to stop broadcasting one of these Characteristic Values it shall of course keep advertising the other ones.

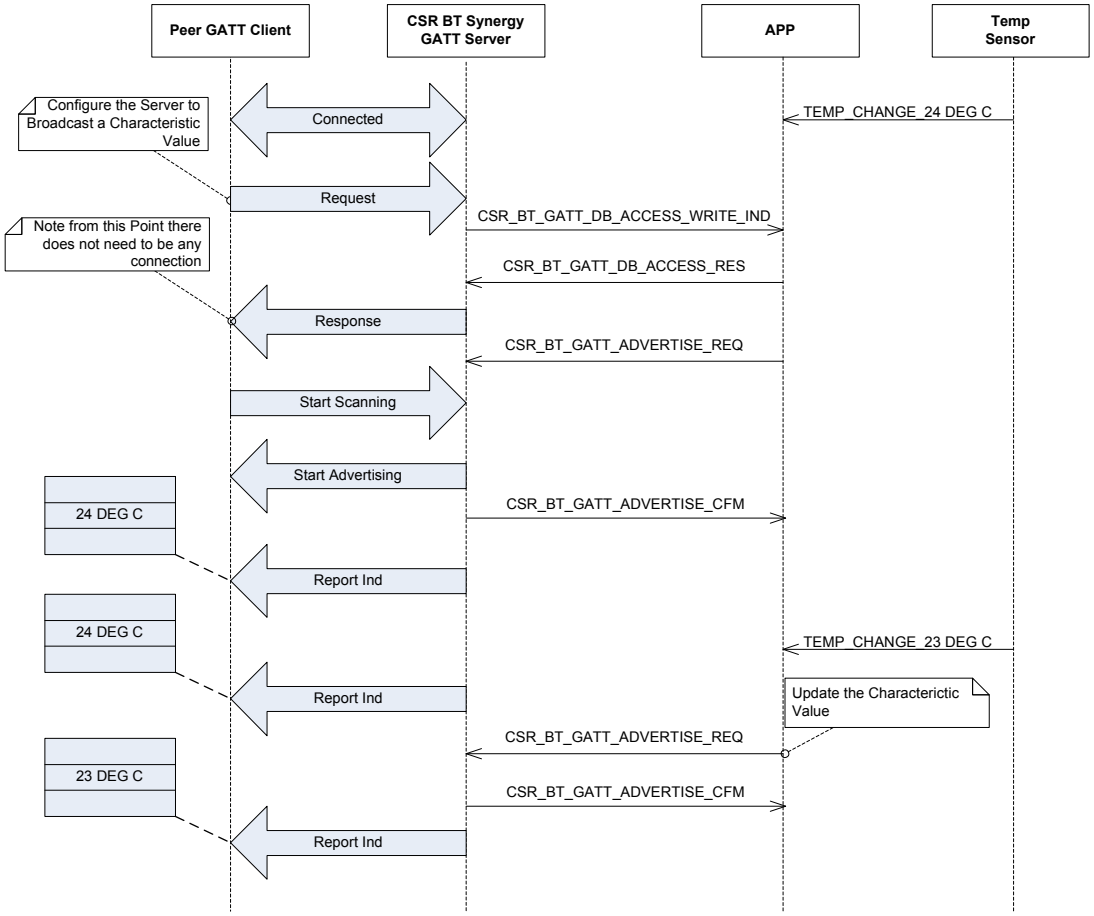


Figure 12: Broadcasting a Characteristic Value from a Server to a Client

3.6.11 Creating a Characteristic Presentation Format Declaration

The Characteristic Presentation Format declaration is an optional characteristic descriptor that defines the format of the Characteristic Value.

The utilities function

- `CsrBtGattUtilCreateCharacPresentationFormat`

of which arguments are described in Table 83, creates a Presentation Format declaration, as illustrated in Table 82.

Note this characteristic descriptor may occur in any position within the characteristic definition after the Characteristic Value, and if more than one Characteristic Presentation Format declaration exist, in the characteristic definition, then a Characteristic Aggregate Format declaration shall also exist as part of the characteristic definition. For information about how the Characteristic Aggregate Format declaration is created, refer to section 3.6.12.

Attribute Handle	Attribute Type	Attribute Value					Attribute Permissions
0xNNNN	0x2904 – UUID for “Characteristic Format”	Format	Exponent	Unit	Name Space	Description	Read only, No Authentication, No Authorization

Table 82: Characteristic Format declaration

Type	Member	Description
CsrBtGattDb	*head	The head pointer. A reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Characteristic Presentation Format Declaration must be place in the local database. Note the 'attrHandle' parameter is automatically incremented by 1.
CsrBtGattFormats	format	One octet that defines the format of the Value of this characteristic. The formats are defined in csr_bt_gatt_prim.h, and can be set to the values defined in Table 84.
CsrInt8	exponent	One octet that determines how the value of this characteristic is further formatted. The exponent field is only used with integer format types, see Table 84. The exponent field is a signed integer and is used for determining how the value of this characteristic is further formatted after the formula: actual value = Characteristic Value * $10^{Exponent}$ I.e. if the Exponent is 2 and the Characteristic Value is 23, the actual value would be 2300.
CsrUint16	unit	The unit is 2 octets and describes the unit of this characteristic. For more information refer to the Assigned Numbers Specification [2].
CsrUint8	namespace	The namespace is one octet long and is the namespace of the description parameter. For more information please refer to the Assigned Numbers Specification [2].
CsrUint16	description	The Description is 2 octets long and enumerated value. For more information refer to the Assigned Numbers document [2].
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 83: Arguments for `CsrBtGattUtilCreateCharacPresentationFormat` function

Format	Short Name	Description	Exponent/Value
CSR_BT_GATT_CHARAC_FORMAT_RFU (0x00)	rfu	Reserved for Future Used	No
CSR_BT_GATT_CHARAC_FORMAT_BOOLEAN (0x01)	Boolean	Unsigned 1-bit; 0 = false, 1 = true	No
CSR_BT_GATT_CHARAC_FORMAT_2BIT (0x02)	2bit	Unsigned 2-bit integer	No
CSR_BT_GATT_CHARAC_FORMAT_NIBBLE (0x03)	nibble	Unsigned 4-bit integer	No
CSR_BT_GATT_CHARAC_FORMAT_UINT8* (0x04)	uint8	Unsigned 8-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_UINT12 (0x05)	uint12	Unsigned 12-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_UINT16 (0x06)	uint16	Unsigned 16-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_UINT24 (0x07)	uint24	Unsigned 24-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_UINT32 (0x08)	uint32	Unsigned 32-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_UINT48 (0x09)	uint48	Unsigned 48-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_UINT64 (0x0A)	uint64	Unsigned 64-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_UINT128 (0x0B)	uint128	Unsigned 128-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_SINT8 (0x0C)	sint8	Signed 8-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_SINT12 (0x0D)	sint12	Signed 12-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_SINT16 (0x0E)	sint16	Signed 16-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_SINT24 (0x0F)	sint24	Signed 24-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_SINT32 (0x10)	sint32	Signed 32-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_SINT48 (0x11)	sint48	Signed 48-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_SINT64 (0x12)	sint64	Signed 64-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_SINT128 (0x13)	sint128	Signed 128-bit integer	Yes
CSR_BT_GATT_CHARAC_FORMAT_FLOAT32 (0x14)	float32	IEEE-754 32-bit floating point	No
CSR_BT_GATT_CHARAC_FORMAT_FLOAT64 (0x15)	float64	IEEE-754 64-bit floating point	No
CSR_BT_GATT_CHARAC_FORMAT_SFLOAT (0x16)	SFLOAT	IEEE-11073 16-bit SFLOAT	No
CSR_BT_GATT_CHARAC_FORMAT_FLOAT (0x17)	FLOAT	IEEE-11073 32-bit FLOAT	No
CSR_BT_GATT_CHARAC_FORMAT_DUINT16 (0x18)	duint16	IEEE-20601 format	No
CSR_BT_GATT_CHARAC_FORMAT_UTF8S (0x19)	uf8s	UTF-8 string	No
CSR_BT_GATT_CHARAC_FORMAT_UTF16S (0x1A)	utf16s	UTF-16 sting	No
CSR_BT_GATT_CHARAC_FORMAT_STRUCT (0x1B)	struct	Opaque structure	No
0x1C – 0xFF	rfu	Reserved for Future Use	No

Table 84: Characteristic Format types

3.6.12 Creating a Characteristic Aggregate Format Declaration

The Characteristic Aggregate Format declaration is an optional characteristic descriptor that defines the format of an aggregated Characteristic Value.

The utilities function

- *CsrBtGattUtilCreateCharacAggregateFormat*

of which arguments are described in Table 86, creates a Aggregate Format declaration, as illustrated in Table 85.

Only one Aggregate Format declaration may exist within a Characteristic definition, and it may occur in any position after the Characteristic Value. Note if more than one Characteristic Presentation Format declarations exist, see section 3.6.11, within the characteristic definition this Characteristic declaration shall also exist.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	0x2905 – UUID for "Characteristic Aggregate Format"	List of *Attribute Handles* for the Characteristic Presentation Format Declarations	Read Only, No authentication, No authorization

Table 85: Characteristic Aggregate Format declaration

Type	Member	Description
CsrBtGattDb	*head	The head pointer. A reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the Characteristic Aggregate Format Declaration must be placed in the local database. Note the 'attrHandle' parameter is automatically incremented by 1.
CsrUint16	handlesCount	The number of Attribute Handles in the list.
const CsrBtGattHandle	*handles	A list of Attribute Handles of Characteristic Presentation Format declarations. where each Attribute handle points to a Characteristic Presentation Format declaration. Note, this value shall be const data, i.e. not an allocated pointer.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 86: Arguments for `CsrBtGattUtilCreateCharacAggregateFormat` function

3.6.13 Creating a Profile Define Descriptor

A server application may define its own descriptors. A Profile Defined descriptor may exist within a Characteristic definition, and it may occur in any position after the Characteristic Value. Two utilities functions exist for creating a Profile Defined descriptor:

- *CsrBtGattUtilCreateDbEntryFromUuid16*
- *CsrBtGattUtilCreateDbEntryFromUuid128*

These functions, of which arguments are described in Table 88 and Table 89, create a Profile Defined descriptor with a 16-Bit or 128-Bit characteristic descriptor UUID, as illustrated in Table 87.

Attribute Handle	Attribute Type	Attribute Value	Attribute Permissions
0xNNNN	16 or 128-Bit characteristic descriptor UUID	Profile defined	Higher layer profile or implementation specific

Table 87: A Profile Define Descriptor declaration

Type	Member	Description
CsrBtGattDb	*head	The head pointer. A reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the profile define descriptor must be place in the local database. Note the 'attrHandle' parameter is automatic incremented by 1.
CsrBtUuid16	uuid16	A 16-Bit characteristic descriptor UUID
CsrBtGattPermFlags	attrPermission	The permission flag field is use to determined whether read or write access is permitted or not. The following permission flags are defined in csr_bt_gatt_prim.h and can be set to: CSR_BT_GATT_PERM_FLAGS_NONE CSR_BT_GATT_PERM_FLAGS_READ CSR_BT_GATT_PERM_FLAGS_WRITE_CMD CSR_BT_GATT_PERM_FLAGS_WRITE_REQ CSR_BT_GATT_PERM_FLAGS_WRITE CSR_BT_GATT_PERM_FLAGS_AUTH_SIGNED_WRITES
CsrBtGattAttrFlags		The attribute flag field defines how the Profile Define descriptor can be accessed. The following attribute flags are defined in csr_bt_gatt_prim.h and can be set to: CSR_BT_GATT_ATTR_FLAGS_NONE CSR_BT_GATT_ATTR_FLAGS_DYNLEN CSR_BT_GATT_ATTR_FLAGS_IRQ_READ CSR_BT_GATT_ATTR_FLAGS_IRQ_WRITE CSR_BT_GATT_ATTR_FLAGS_READ_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_READ_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE CSR_BT_GATT_ATTR_FLAGS_DISABLE_LE CSR_BT_GATT_ATTR_FLAGS_DISABLE_BREDR Please note multiple attribute flags can be set. For more information, refer to 3.6.2.
CsrUint16	attrValueLength	The length of the Attribute Value in octets
const CsrUint8	*attrValue	The attribute value. Note this value shall be const data, i.e. not an allocated pointer.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 88: Arguments for CsrBtGattUtilCreateDbEntryFromUuid16 function

Type	Member	Description
CsrBtGattDb	*head	The head pointer, a reference pointer which points to the first element in the linked list.
CsrBtGattHandle	*attrHandle	The Attribute Handle. I.e. the index of where the profile define descriptor must be placed in the local database. Note the 'attrHandle' parameter is automatic incremented by 1.
CsrBtUuid128	uuid128	A 128-Bit characteristic descriptor UUID
CsrBtGattPermFlags	attrPermission	The permission flag field is used for determining whether read or write access is permitted or not. The following permission flags are defined in csr_bt_gatt_prim.h and can be set to: CSR_BT_GATT_PERM_FLAGS_NONE CSR_BT_GATT_PERM_FLAGS_READ CSR_BT_GATT_PERM_FLAGS_WRITE_CMD CSR_BT_GATT_PERM_FLAGS_WRITE_REQ CSR_BT_GATT_PERM_FLAGS_WRITE CSR_BT_GATT_PERM_FLAGS_AUTH_SIGNED_WRITES
CsrBtGattAttrFlags		The attribute flag field defines how the Profile Define descriptor can be access. The following attribute flags are defined in csr_bt_gatt_prim.h and can be set to: CSR_BT_GATT_ATTR_FLAGS_NONE CSR_BT_GATT_ATTR_FLAGS_DYNLEN CSR_BT_GATT_ATTR_FLAGS_IRQ_READ CSR_BT_GATT_ATTR_FLAGS_IRQ_WRITE CSR_BT_GATT_ATTR_FLAGS_READ_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_READ_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_WRITE_ENCRYPTION CSR_BT_GATT_ATTR_FLAGS_WRITE_AUTHENTICATION CSR_BT_GATT_ATTR_FLAGS_AUTHORISATION CSR_BT_GATT_ATTR_FLAGS_ENCR_KEY_SIZE CSR_BT_GATT_ATTR_FLAGS_DISABLE_LE CSR_BT_GATT_ATTR_FLAGS_DISABLE_BREDR Please note multiple attribute flags can be set. For more information refer to 3.6.2.
CsrUint16	attrValueLength	The length of the Attribute Value in octets
const CsrUint8	*attrValue	The attribute value. Note this value shall be const data, i.e. not an allocated pointer.
CsrBtGattDb	**tail	The tail pointer, a reference pointer which points to the last element in the linked list.

Table 89: Arguments for CsrBtGattUtilCreateDbEntryFromUuid128 function

3.6.14 Database Add

When the application has built a database it can add it to GATT by calling:

- *CsrBtGattDbAddReqSend*.

The arguments for *CsrBtGattDbAddReqSend* are described in Table 90.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattDb	*db	The local database

Table 90: Arguments for `CsrBtGattDbAddReqSend` function

This function will send a CSR_BT_GATT_DB_ADD_REQ primitive to GATT. Later, when GATT has stored the database, it returns a CSR_BT_GATT_DB_ADD_CFM message to the application. The parameters for CSR_BT_GATT_DB_ADD_CFM are described in Table 91.

Note, that if a Service has been added to the database, the server application shall notify the clients which it is bonded to. For more information refer to 3.6.17.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DB_ADD_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 91: Members in a CSR_BT_GATT_DB_ADD_CFM primitive

3.6.15 Database Remove

A Server application can remove handle entries from its database by calling:

- *CsrBtGattDbRemoveReqSend*

The arguments for this function are described in Table 92.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	start	First handle number to be removed
CsrBtGattHandle	end	Last handle number to be removed

Table 92: Arguments for `CsrBtGattDbRemoveReqSend` function

This function will send a CSR_BT_GATT_DB_REMOVE_REQ primitive to GATT. Later, when GATT has removed the handle entries from the database, it returns a CSR_BT_GATT_DB_REMOVE_CFM message to the application. The parameters for CSR_BT_GATT_DB_REMOVE_CFM are described in Table 93.

Note, that if a Service has been removed from the database, the server application shall notify clients which it is bonded to. For more information refer to 3.6.17

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DB_REMOVE_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrUint16	numOfAttr	Number of attributes removed

Table 93: Members in a CSR_BT_GATT_DB_REMOVE_CFM primitive

3.6.16 De-allocate Attribute Handles

A server application is able to free/de-allocate the range of attribute handles that it has previously allocated, see section 3.6.3, by calling:

- *CsrBtGattDbDeallocReqSend*

The arguments for *CsrBtGattDbDeallocReqSend* are described in Table 92.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier

Table 94: Arguments for `CsrBtGattDbDeallocReqSend` function

Please note that if GATT receives the CSR_BT_GATT_DB_DEALLOC_REQ primitive and the application still has some attribute handles register in the local database, GATT will ensure that these handle entries are removed from the database, see section 3.6.15. Later, when GATT has de-allocated the attribute handles, it returns a CSR_BT_GATT_DB_DEALLOC_CFM message to the application. The parameters for CSR_BT_GATT_DB_DEALLOC_CFM are described in Table 95. Also note that if a Service has been removed from the database, the server application shall notify clients which it is bonded to. For more information refer to 3.6.17.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DB_DEALLOC_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtGattHandle	start	Start attribute handle
CsrBtGattHandle	end	End attribute handle

Table 95: Members in a CSR_BT_GATT_DB_DEALLOC_CFM primitive

3.6.17 Service Change

Some Clients may read the database once and use the information across reconnections without reading the database again. This means that if a Service has been added, modified, or removed from the database, the server application shall notify clients which it is bonded to. The Server application can notify a client about a service change by calling the function:

- *CsrBtGattServiceChangedEventReqSend*

of which arguments are described in Table 96. This function will send a CSR_BT_GATT_EVENT_SEND_REQ primitive to GATT. When the procedure is completed, GATT returns a CSR_BT_GATT_EVENT_SEND_CFM message to the application, see Table 79.

Note, if a service change occurs while one or more bonded Clients are not connected, the application shall send a service change when the client reconnects to the server.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier

Type	Argument	Description
CsrBtGattHandle	startHandle	Start of Affected Attribute Handle Range
CsrBtGattHandle	endHandle	End of Affected Attribute Handle Range

Table 96: Arguments for `CsrBtGattServiceChangedEventReqSend` function

3.7 Primary Service Discovery Procedures

A Client application is able to Discover Primary Services on a GATT Server (Peer/Local) by calling one of the following functions:

- *CsrBtGattDiscoverAllPrimaryServicesReqSend*
- *CsrBtGattDiscoverAllPrimaryServicesLocalReqSend*
- *CsrBtGattDiscoverPrimaryServicesBy16BitUuidReqSend*
- *CsrBtGattDiscoverPrimaryServicesBy16BitUuidLocalReqSend*
- *CsrBtGattDiscoverPrimaryServicesBy128BitUuidReqSend*
- *CsrBtGattDiscoverPrimaryServicesBy128BitUuidLocalReqSend*

Where the function *CsrBtGattDiscoverAllPrimaryServicesReqSend* is used for discovering all primary services on a peer server device and *CsrBtGattDiscoverAllPrimaryServicesLocalReqSend* discovers all primary services on the local device. The arguments for the two functions are described in Table 97 and Table 98.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier

Table 97: Arguments for `CsrBtGattDiscoverAllPrimaryServicesReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier

Table 98: Arguments for `CsrBtGattDiscoverAllPrimaryServicesLocalReqSend` function

The last four functions are used for discovering a specific primary service when only a Service UUID is known. The functions *CsrBtGattDiscoverPrimaryServicesBy16BitUuidReqSend* and *CsrBtGattDiscoverPrimaryServicesBy128BitUuidReqSend* are used for discovering a primary service which matches the given Service UUID (16 or 128-Bit) on a peer server device and the functions *CsrBtGattDiscoverPrimaryServicesBy16BitUuidLocalReqSend* and *CsrBtGattDiscoverPrimaryServicesBy128BitUuidLocalReqSend* do the same just on the local device. The arguments for the four functions are described in Table 99, Table 100, Table 101, and Table 102.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtUuid16	uuid16	The 16-bit service UUID to look for

Table 99: Arguments for `CsrBtGattDiscoverPrimaryServicesBy16BitUuidReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier

Type	Argument	Description
CsrBtConnId	btConnId	Connection identifier
CsrBtUuid128	uuid128	The 128-bit service UUID to look for

Table 100: Arguments for `CsrBtGattDiscoverPrimaryServicesBy128BitUuidReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtUuid16	uuid16	The 16-bit service UUID to look for

Table 101: Arguments for `CsrBtGattDiscoverPrimaryServicesBy16BitUuidLocalReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtUuid128	uuid128	The 128-bit service UUID to look for

Table 102: Arguments for `CsrBtGattDiscoverPrimaryServicesBy128BitUuidLocalReqSend` function

Common to all these functions are that they will all send a `CSR_BT_GATT_DISCOVER_SERVICES_REQ` primitive to GATT. The procedure for Discover Primary Services, including both an initiator and a responder side, is illustrated in Figure 13. Every time a Primary Service is discovered, matching the search criteria, a `CSR_BT_GATT_DISCOVER_SERVICES_IND` message is sent to the application. When the procedure is completed, GATT returns a `CSR_BT_GATT_DISCOVER_SERVICES_CFM` message to the application. The parameters for `CSR_BT_GATT_DISCOVER_SERVICES_IND` are described in Table 103 and the parameters for `CSR_BT_GATT_DISCOVER_SERVICES_CFM` are described in Table 104. The definition of a Service Declaration is described in section 3.6.4.

Please note, if the Client application is discovering Primary Services from the local device it does not need to be connected. Once the Primary Services are discovered, additional information about each service can be found by using other procedures, including characteristic discovery, see section 3.9, and relationship discovery to find other related primary or secondary services, see section 3.8.

The Client may cancel this procedure if a desired Primary Service is found before this procedure has finished. This is done by calling *`CsrBtGattCancelReqSend`*, of which parameters are described in Table 170. When the discovery procedure is cancelled, the `CSR_BT_GATT_DISCOVER_SERVICES_CFM` message is returned to the Client application as confirmation.

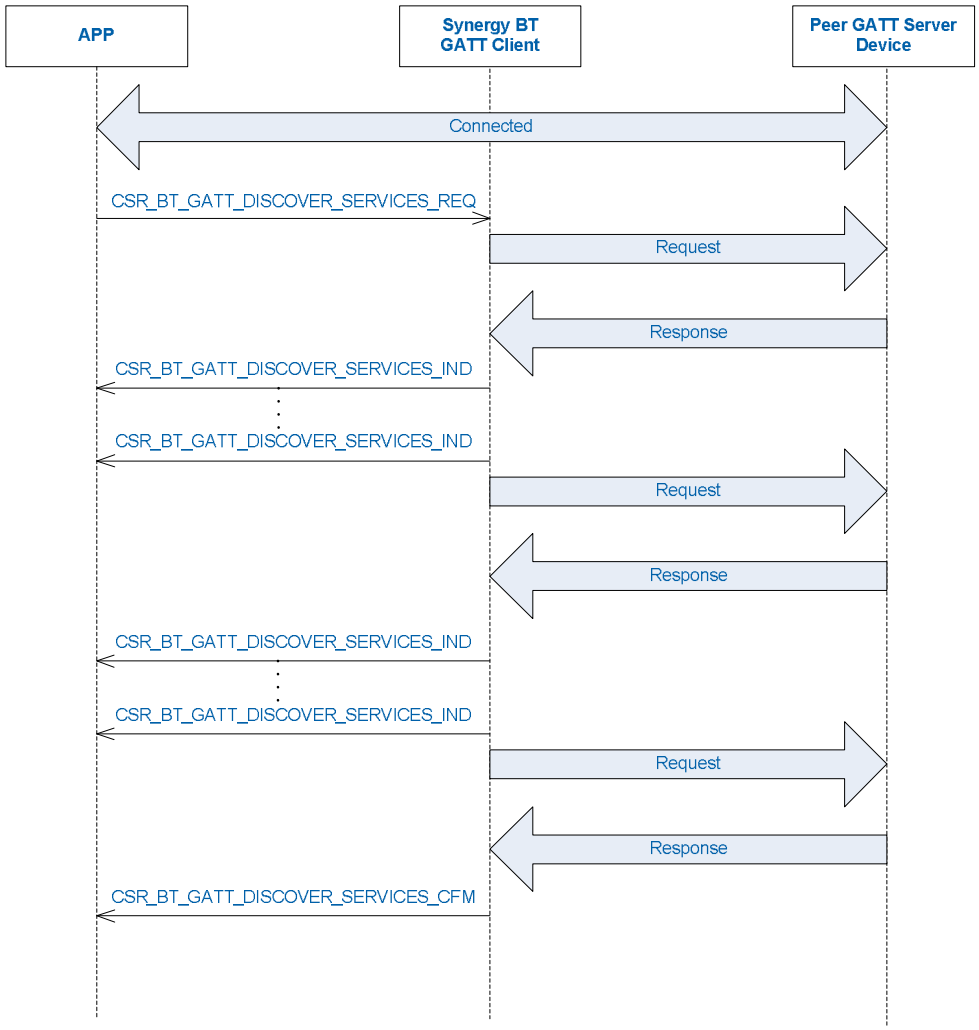


Figure 13: Discover Primary Services

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DISCOVER_SERVICES_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier.
CsrBtGattHandle	endHandle	The end group handle
CsrBtUuid	uuid	A 16 or 128-Bit Service UUID, which is presented in Little-Endian. If uuid.length equals CSR_BT_UUID16_SIZE it is a 16-Bit Service UUID, otherwise it is a 128-Bit Service UUID. In case that it is a 16-Bit UUID, it is placed in the two first octets. The application is able to use the macro CSR_GET_UINT16_FROM_LITTLE_ENDIAN to convert into a CsrUint16. (CSR_GET_UINT16_FROM_LITTLE_ENDIAN(msg->uuid.uuid))

Table 103: Members in a CSR_BT_GATT_DISCOVER_SERVICES_IND primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DISCOVER_SERVICES_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values that are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 104: Members in a CSR_BT_GATT_DISCOVER_SERVICES_CFM primitive

3.8 Relationship Discovery Procedures

After a Client application has discovered the primary services on the server, it is able to discover service relationships to other services by calling one of the following functions:

- *CsrBtGattFindInclServicesReqSend*
- *CsrBtGattFindInclServicesLocalReqSend*

Where the function *CsrBtGattFindInclServicesReqSend* is used for finding include service declarations within a service definition on a peer server device and *CsrBtGattFindInclServicesLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 105 and Table 106.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	startHandle	The start handle of the specified service
CsrBtGattHandle	endHandle	The end handle of the specified service

Table 105: Arguments for `CsrBtGattFindIncludeServicesReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	startHandle	The start handle of the specified service
CsrBtGattHandle	endHandle	The end handle of the specified service

Table 106: Arguments for `CsrBtGattFindIncludeServicesLocalReqSend` function

Common to these functions are that they will all send a CSR_BT_GATT_FIND_INCL_SERVICES_REQ primitive to GATT. The procedure for finding included service declarations within a service definition, including both an initiator and a responder side, is as illustrated in Figure 14. Every time an included service declaration within a service definition is discovered a CSR_BT_GATT_FIND_INCL_SERVICES_IND message is sent to the application. When the procedure is completed, GATT returns a CSR_BT_GATT_FIND_INCL_SERVICES_CFM message to the application. The parameters for CSR_BT_GATT_DISCOVER_SERVICES_IND are described in Table 103 and the parameters for CSR_BT_GATT_DISCOVER_SERVICES_CFM are described in Table 104. Please note, if the Client application is discovering Service Relationships from the local device it does not need to be connected. The definition of an Include Declaration is described in section 3.6.5.

The Client application is permitted to cancel this procedure early, if a desired included service is found prior to discovering all the included services on the server. This can be done by calling `CsrBtGattCancelReqSend`, of which parameters are described in Table 170. As confirmation the application will still receive the `CSR_BT_GATT_DISCOVER_SERVICES_CFM` message.

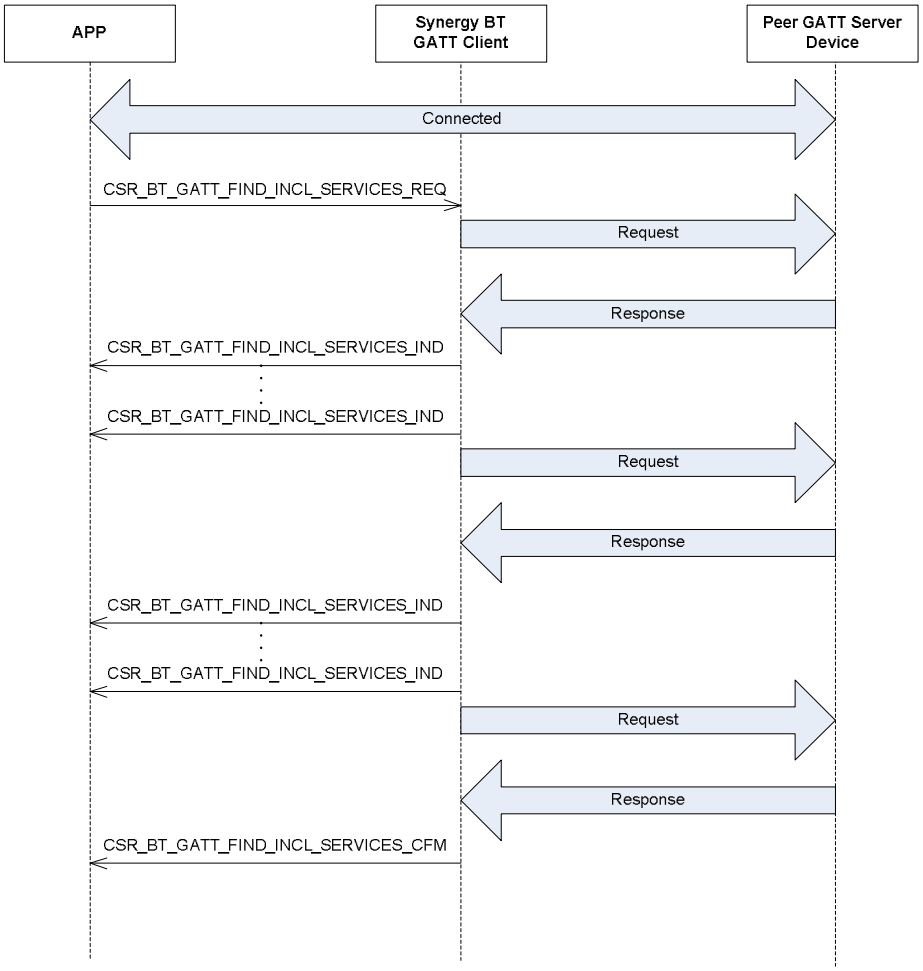


Figure 14: Discover Service Relationship

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_FIND_INCL_SERVICES_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	attrHandle	Attribute handle of the Include Service
CsrBtGattHandle	startHandle	Starting handle of the Include Service
CsrBtGattHandle	endGroupHandle	The end group handle
CsrBtUuid	uuid	A 16 or 128-Bit Service UUID, which is presented in Little-Endian. If uuid.length equals CSR_BT_UUID16_SIZE it is a 16-Bit Service UUID, otherwise it is a 128-Bit Service UUID. In case that it is a 16-Bit UUID it is placed in the two first octets. The application is able to use the macro CSR_GET_UINT16_FROM_LITTLE_ENDIAN to convert into a CsrUint16. (CSR_GET_UINT16_FROM_LITTLE_ENDIAN(msg->uuid.uuid))

Table 107: Members in a CSR_BT_GATT_FIND_INCL_SERVICES_IND primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_FIND_INCL_SERVICES_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 108: Members in a CSR_BT_GATT_FIND_INCL_SERVICES_CFM primitive

3.9 Characteristic Discovery Procedures

After a Client application has discovered the primary services on the server, it is able to discover Service Characteristics on a GATT Server (Peer/Local) by calling one of the following functions:

- *CsrBtGattDiscoverAllCharacOfAServiceReqSend*
- *CsrBtGattDiscoverAllCharacOfAServiceLocalReqSend*
- *CsrBtGattDiscoverCharacBy16BitUuidReqSend*
- *CsrBtGattDiscoverCharacBy16BitUuidLocalReqSend*
- *CsrBtGattDiscoverCharacBy128BitUuidReqSend*
- *CsrBtGattDiscoverCharacBy128BitUuidLocalReqSend*

The function `CsrBtGattDiscoverAllCharacOfAServiceReqSend` is used by a client for finding all the characteristic declarations within a service definition on a server when only the service handle range is known on a peer server device and *CsrBtGattDiscoverAllCharacOfAServiceLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 109 and Table 110.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	startHandle	The start handle of the specified service
CsrBtGattHandle	endHandle	The end handle of the specified service

Table 109: Arguments for `CsrBtGattDiscoverAllCharacOfAServiceReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	startHandle	The start handle of the specified service
CsrBtGattHandle	endHandle	The end handle of the specified service

Table 110: Arguments for `CsrBtGattDiscoverAllCharacOfAServiceLocalReqSend` function

The last four functions are used for discovering service characteristics on a server when only the service handle ranges are known and the characteristic UUID is known. The functions `CsrBtGattDiscoverCharacBy16BitUuidReqSend` and `CsrBtGattDiscoverCharacBy128BitUuidReqSend` are used for discovering a service characteristics which matches the given Service UUID (16 or 128-Bit) on a peer server device and the functions `CsrBtGattDiscoverCharacBy16BitUuidLocalReqSend` and `CsrBtGattDiscoverCharacBy128BitUuidLocalReqSend` do the same just on the local device. The arguments for the four functions are described in Table 111, Table 112, Table 113, and Table 114.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	startHandle	The start handle of the specified service
CsrBtGattHandle	endHandle	The end handle of the specified service
CsrBtUuid16	uuid16	The 16-bit characteristic UUID

Table 111: Arguments for `CsrBtGattDiscoverCharacBy16BitUuidReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	startHandle	The start handle of the specified service
CsrBtGattHandle	endHandle	The end handle of the specified service
CsrBtUuid128	uuid128	The 128-bit characteristic UUID

Table 112: Arguments for `CsrBtGattDiscoverCharacBy128BitUuidReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	startHandle	The start handle of the specified service
CsrBtGattHandle	endHandle	The end handle of the specified service
CsrBtUuid16	uuid16	The 16-bit characteristic UUID

Table 113: Arguments for `CsrBtGattDiscoverCharacBy16BitUuidLocalReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	startHandle	The start handle of the specified service
CsrBtGattHandle	endHandle	The end handle of the specified service
CsrBtUuid128	uuid128	The 128-bit characteristic UUID

Table 114: Arguments for `CsrBtGattDiscoverCharacBy128BitUuidLocalReqSend` function

Common to all these functions are that they will all send a `CSR_BT_GATT_DISCOVER_CHARAC_REQ` primitive to GATT. The procedure for Discover Service Characteristics, including both an initiator and a responder side, as illustrated in Figure 15. Every time a characteristic declaration has been discovered, matching the search criteria, a `CSR_BT_GATT_DISCOVER_CHARAC_IND` message is sent to the application. When the procedure is completed, GATT returns a `CSR_BT_GATT_DISCOVER_CHARAC_CFM` message. The parameters for `CSR_BT_GATT_DISCOVER_CHARAC_IND` are described in Table 115 and the parameters for `CSR_BT_GATT_DISCOVER_CHARAC_CFM` are described in Table 116. The definition of a Characteristic Declaration is described in section 3.6.6.

Please note, if the Client application is discovering Service Characteristics from the local device it does not need to be connected. Once the Service Characteristics are discovered it is possible to discover the characteristics descriptors, see section 3.10.

The Client may cancel this procedure if a desired Service Characteristics is found before the procedure has finished. This is done by calling *`CsrBtGattCancelReqSend`*, of which parameters are described in Table 170. When the discovery procedure is cancelled, the `CSR_BT_GATT_DISCOVER_CHARAC_CFM` message is returned to the application as confirmation.

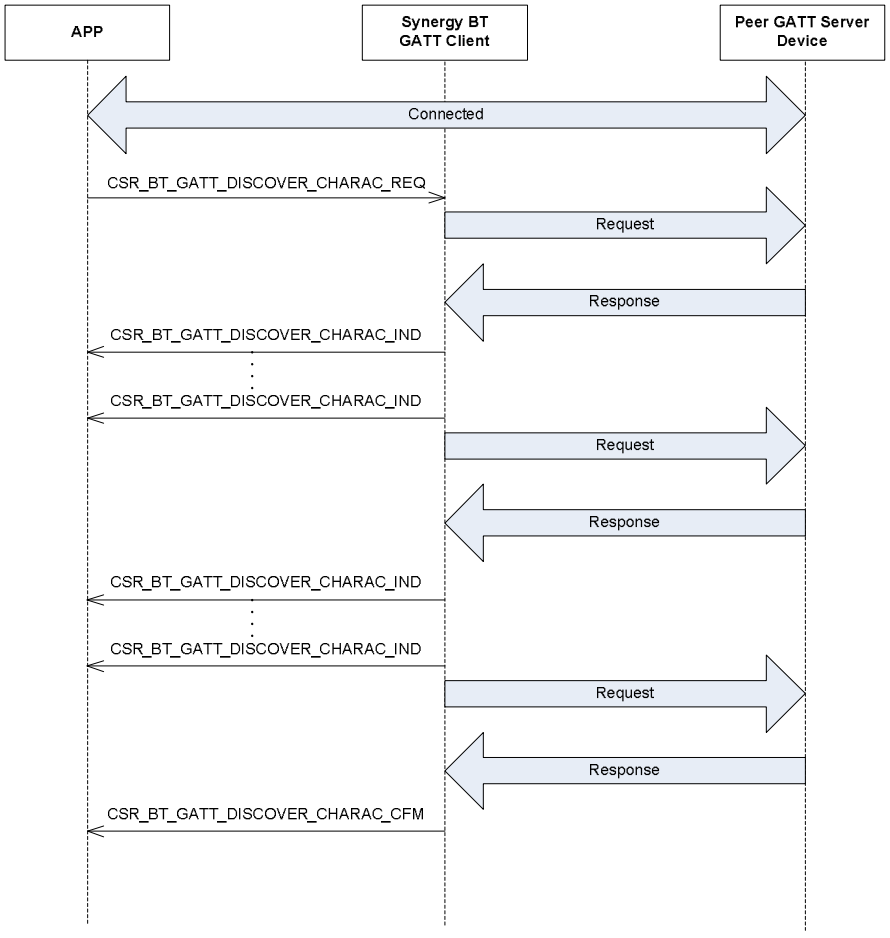


Figure 15: Discover Service Characteristics

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DISCOVER_CHARAC_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	declarationHandle	Handle for the characteristic declaration
CsrBtGattPropertiesBits	property	Characteristic Property, which is a bit field that determines how the Characteristic Value can be used, or how the characteristic Descriptors can be addressed. The declaration of a Characteristic Value and the Characteristic descriptors is described in section 3.6 where also the CsrBtGattPropertiesBits are described, see Table 70. Note, multiple property bits may be set and valid values are defined in csr_bt_gatt_prim.h.
CSrBtUuid	uuid	A Characteristic UUID, that describes the type of the characteristic value, which can be 16 or 128-Bit. The UUID is represented in Little-Endian. If uuid.length equals CSR_BT_UUID16_SIZE it is a 16-Bit UUID, otherwise it is a 128-Bit UUID. If case that it is a 16-Bit UUID it is placed in the two first octets. The application is able to use the macro CSR_GET_UINT16_FROM_LITTLE_ENDIAN to convert into a CsrUint16. (CSR_GET_UINT16_FROM_LITTLE_ENDIAN(msg->uuid.uuid))
CsrBtGattHandle	valueHandle	Characteristic value Handle which is the handle of the Attribute containing the Value of this characteristic.

Table 115: Members in a CSR_BT_GATT_DISCOVER_CHARAC_IND primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DISCOVER_CHARAC_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 116: Members in a CSR_BT_GATT_DISCOVER_CHARAC_CFM primitive

3.10 Characteristic Descriptor Discovery Procedures

After a Client application has found the characteristics on the server, it is able to discover all the Characteristic Descriptors within a characteristic definition on a GATT Server (Peer/Local) by calling one of the following functions:

- *CsrBtGattDiscoverAllCharacDescriptorsReqSend*
- *CsrBtGattDiscoverAllCharacDescriptorsLocalReqSend*

Where the function *CsrBtGattDiscoverAllCharacDescriptorsReqSend* is used for finding all descriptor declarations within a service definition on a peer server device and *CsrBtGattDiscoverAllCharacDescriptorsLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 117 and Table 118.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	startHandle	The handle of the specified characteristic value + 1
CsrBtGattHandle	endHandle	The end handle of the specified characteristic

Table 117: Arguments for `CsrBtGattDiscoverAllCharacDescriptorsReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	startHandle	The handle of the specified characteristic value + 1
CsrBtGattHandle	endHandle	The end handle of the specified characteristic

Table 118: Arguments for `CsrBtGattDiscoverAllCharacDescriptorsLocalReqSend` function

Common to these functions are that they will all send a `CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_REQ` primitive to GATT. The procedure for discovering all the Characteristic Descriptors within a characteristic definition, including both an initiator and a responder side, as illustrated in Figure 16. Every time a Characteristic Descriptor within a characteristic definition is discovered a `CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_IND` message is sent to the application. When the procedure is completed, GATT returns a `CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_CFM` message. The parameters for `CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_IND` are described in Table 119 and the parameters for `CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_CFM` are described in Table 120. Please note, if the Client application is discovering Characteristic Descriptors from the local device it does not need to be connected. The definition of the define Characteristic Descriptors is described in section 3.6.7, 3.6.8, 3.6.9, 3.6.10, 3.6.11, 3.6.12 and section 3.6.13.

The Client application is permitted to cancel this procedure early if a desired Characteristic Descriptor is found prior to discovering all the Characteristic Descriptors on the server. This can be done by calling *CsrBtGattCancelReqSend*, of which parameters are described in Table 170. As confirmation, the application will still receive the `CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_CFM` message.

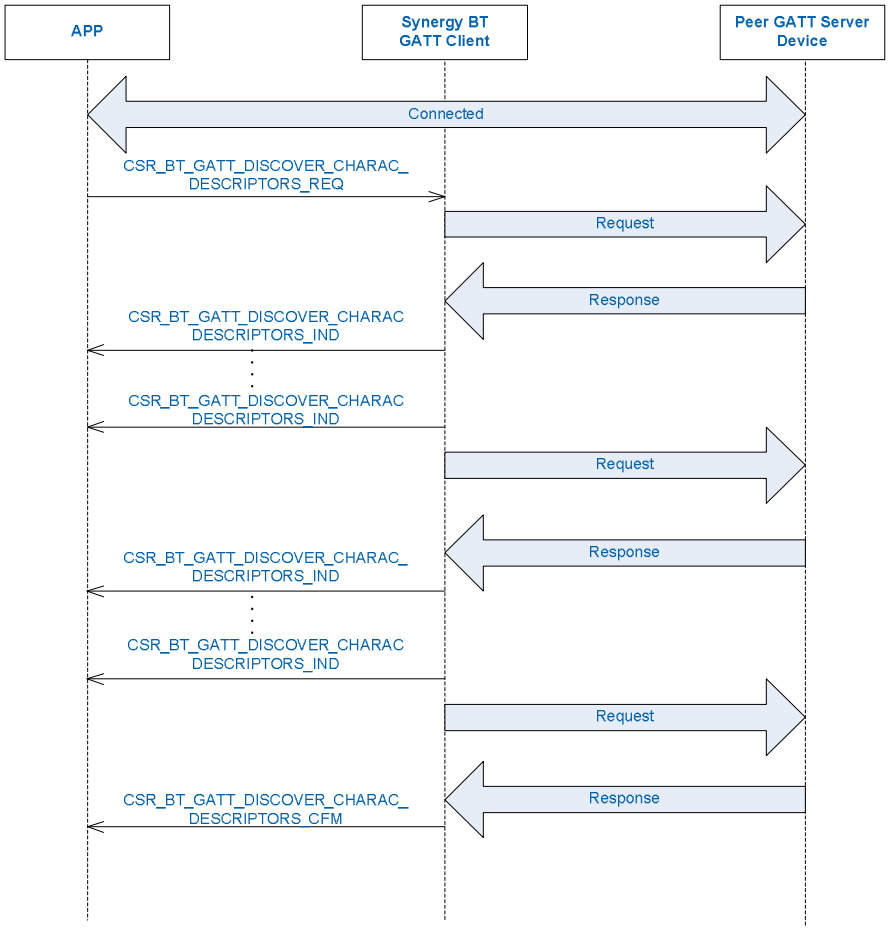


Figure 16: Discover All Characteristics Descriptors

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CSrBtUuid	uuid	A Characteristic Descriptor UUID, that describes the type of the characteristic descriptor value, which can be 16 or 128-Bit. The UUID is represented in Little-Endian. If uuid.length equals CSR_BT_UUID16_SIZE it is a 16-Bit UUID, otherwise it is a 128-Bit UUID. In case that it is a 16-Bit UUID it is placed in the two first octets. The application is able to use the macro CSR_GET_UINT16_FROM_LITTLE_ENDIAN to convert into a CsrUint16. (CSR_GET_UINT16_FROM_LITTLE_ENDIAN(msg->uuid.uuid))
CsrBtGattHandle	descriptorHandle	The handle of the Characteristic Descriptor declaration

Table 119: Members in a CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_IND primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 120: Members in a CSR_BT_GATT_DISCOVER_CHARAC_DESCRIPTORS_CFM primitive

3.11 Read Characteristic Value Procedures

A Client application can read a Characteristic Value from a GATT Server (Peer/Local) in three ways.

1. It can read it directly if it knows the handle of the Characteristic Value,
2. It can read it by using the Characteristic UUID, or
3. it can read multiple Characteristic Values if it knows the Characteristic Value handles.

3.11.1 Read Characteristic Value

A Client application is able to read Characteristic Value from a GATT Server (Peer/Local) when the Client knows the handle of the Characteristic Value, by calling one of the following functions:

- *CsrBtGattReadReqSend*
- *CsrBtGattReadLocalReqSend*

The function *CsrBtGattReadReqSend* is used for reading the Characteristic Value from a peer server and *CsrBtGattReadLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 121and Table 122.

Note, the function *CsrBtGattReadReqSend* shall only be used if the peer server has set the Characteristic Properties 'Read' (0x02) bit. The Characteristic Properties bit field is part of the Characteristic Declaration Attribute Value, see Table 66, which is described in section 3.9 and the definition of a Characteristic Value Declaration is described in Table 67.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	Handle	The Characteristic Value Handle
CsrUint16	Offset	The offset of the first octet that shall be read

Table 121: Arguments for `CsrBtGattReadReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	Handle	The Characteristic Value Handle

Table 122: Arguments for `CsrBtGattReadLocalReqSend` function

Common to these two functions are that they will all send a `CSR_BT_GATT_READ_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_READ_CFM` which includes the read Characteristic Value, see Figure 17. The parameters for `CSR_BT_GATT_READ_CFM` are described in Table 123. Please note, if the Client application is reading the Characteristic Value from the local device, it does not need to be connected.

The Client application is permitted to cancel this read procedure early. This can be done by calling *`CsrBtGattCancelReqSend`*, of which parameters are described in Table 170. As confirmation, the application will still receive the `CSR_BT_GATT_READ_CFM` message.

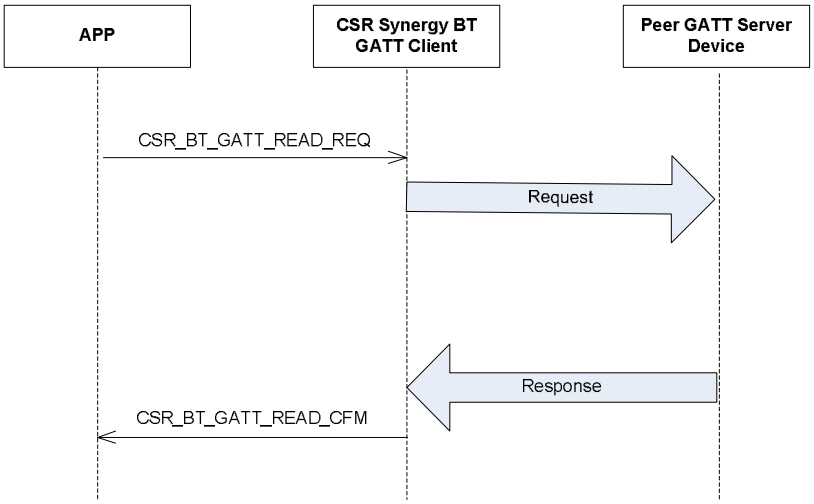


Figure 17: Read Characteristic Value

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrUint16	valueLength	Length of the Characteristic Value in octets
CsrUint8	*value	Pointer to the Characteristic Value

Table 123: Members in a CSR_BT_GATT_READ_CFM primitive

3.11.2 Read Characteristic Value by UUID

A Client application can read a Characteristic Value from a server when it only knows the characteristic UUID and does not know the handle of the characteristic, by using one the functions:

- *CsrBtGattReadBy16BitUuidReqSend*
- *CsrBtGattReadBy128BitUuidReqSend*
- *CsrBtGattReadBy16BitUuidLocalReqSend*
- *CsrBtGattReadBy128BitUuidLocalReqSend*

Where the two first functions read a Characteristic Value matching the given characteristic UUID (16 or 128-Bit) from a peer server device and the last two functions do the same just on the local device. The arguments for the four functions are described in Table 124, Table 125, Table 126, and Table 127.

Note, the functions *CsrBtGattReadBy16BitUuidReqSend* and *CsrBtGattReadBy128BitUuidReqSend* shall only be used if the peer server has set the Characteristic Properties 'Read' (0x02) bit. The Characteristic Properties bit field is part of the Characteristic Declaration Attribute Value, see Table 66, which is described in section 3.9 and the definition of a Characteristic Value Declaration is described in Table 67.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	startHandle	The start handle
CsrBtGattHandle	endHandle	The end handle. NOTE: The Start and End handle parameters shall be set to the range over which this read is to be performed. This is typically the handle range for the service to which the characteristic belongs.
CsrBtUuid16	uuid16	The known 16-bit characteristic UUID

Table 124: Arguments for `CsrBtGattReadBy16BitUuidReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	startHandle	The start handle
CsrBtGattHandle	endHandle	The end handle. NOTE: The Start and End handle parameters shall be set to the range over which this read is to be performed. This is typically the handle range for the service to which the characteristic belongs.
CsrBtUuid128	uuid128	The known 128-bit characteristic UUID

Table 125: Arguments for `CsrBtGattReadBy128BitUuidReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	startHandle	The start handle
CsrBtGattHandle	endHandle	The end handle. NOTE: The Start and End handle parameters shall be set to the range over which this read is to be performed. This is typically the handle range for the service to which the characteristic belongs.
CsrBtUuid16	uuid16	The known 16-bit characteristic UUID

Table 126: Arguments for `CsrBtGattReadBy16BitUuidLocalReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	startHandle	The start handle
CsrBtGattHandle	endHandle	The end handle. NOTE: The Start and End handle parameters shall be set to the range over which this read is to be performed. This is typically the handle range for the service to which the characteristic belongs.
CsrBtUuid128	uuid128	The known 128-bit characteristic UUID

Table 127: Arguments for `CsrBtGattReadBy128BitUuidLocalReqSend` function

Common for these four functions are that they will all send a `CSR_BT_GATT_READ_BY_UUID_REQ` primitive to GATT. The procedure for reading a Characteristic Value by UUID is illustrated in Figure 18. Every time a Characteristic Value has been read, within the range over which this reading procedure shall be performed, a `CSR_BT_GATT_READ_BY_UUID_IND` message is sent to the application. When the procedure is completed, GATT returns a `CSR_BT_GATT_READ_BY_UUID_CFM` message to the application. The parameters for `CSR_BT_GATT_READ_BY_UUID_IND` are described in Table 128 and the parameters for `CSR_BT_GATT_READ_BY_UUID_CFM` are described in Table 129. Please note if the Client application is reading from the local device it does not need to be connected.

The Client application is permitted to cancel this procedure early if a desired Characteristic Value is read prior to reading the entire handle range on the server. This can be done by calling *`CsrBtGattCancelReqSend`*, of which parameters are described in Table 170. As confirmation, the application will still receive the `CSR_BT_GATT_READ_BY_UUID_CFM` message.

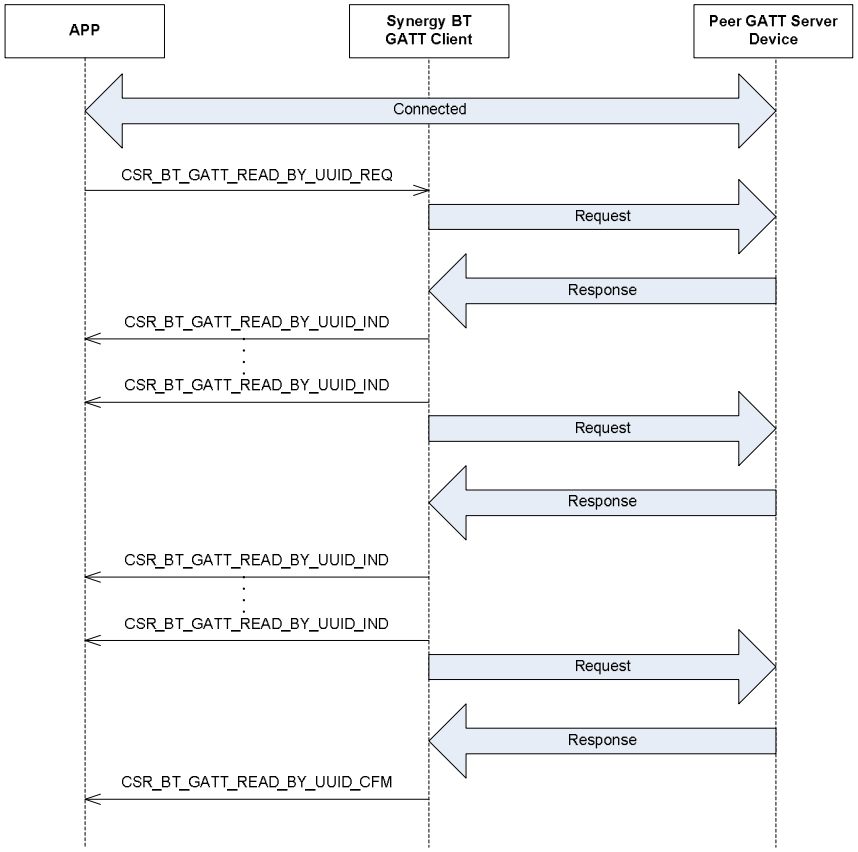


Figure 18: Read Characteristic Value by UUID

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_BY_UUID_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	valueHandle	Characteristic Value Handle
CsrUint16	valueLength	Length of the Characteristic Value in octets
CsrUint8	*value	Pointer to the Characteristic Value

Table 128: Members in a CSR_BT_GATT_READ_BY_UUID_IND primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_BY_UUID_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 129: Members in a CSR_BT_GATT_READ_BY_UUID_CFM primitive

3.11.3 Read Multiple Characteristic Values

A Client application is able to read multiple Characteristic Values from a GATT Server (Peer/Local) when the Client knows the Characteristic Value handles, by calling one of the following functions:

- *CsrBtGattReadMultiReqSend*
- *CsrBtGattReadMultiLocalReqSend*

The function *CsrBtGattReadMultiReqSend* is used for reading multiple Characteristic Values from a peer server and *CsrBtGattReadMultiLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 130 and Table 131.

Note, the function *CsrBtGattReadMultiReqSend* shall only be used if the peer server has set the Characteristic Properties 'Read' (0x02) bit. The Characteristic Properties bit field is part of the Characteristic Declaration Attribute Value, see Table 66, which is described in section 3.9 and the definition of a Characteristic Value Declaration is described in Table 67. Also note that a GATT client should not request to read multiple Characteristic Values when the set of value parameters of the response is equal to (mtu - 1) octets in length, since it is not possible to determine if the last Characteristic Value was read, or additional Characteristic Values exist, or were truncated by the GATT Server.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrUint16	numOfHandles	Number of attribute handles that must be read
CsrBtGattHandle	*handles	An allocated pointer of two or more attribute handles

Table 130: Arguments for `CsrBtGattReadMultiReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrUint16	numOfHandles	Number of attribute handles that must be read
CsrBtGattHandle	*handles	An allocated pointer of two or more attribute handles

Table 131: Arguments for `CsrBtGattReadMultiLocalReqSend` function

Common to these two functions are that they will all send a `CSR_BT_GATT_READ_MULTI_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_READ_MULTI_CFM` which includes the read Characteristic Values, please refer to Figure 19. The parameters for `CSR_BT_GATT_READ_MULTI_CFM` are described in Table 132. Please note, if the Client application is reading multiple Characteristic Values from the local device, it does not need to be connected.

The Client application is permitted to cancel this read procedure early. This can be done by calling *`CsrBtGattCancelReqSend`*, of which parameters are described in Table 170. As confirmation, the application will still receive the `CSR_BT_GATT_READ_MULTI_CFM` message.

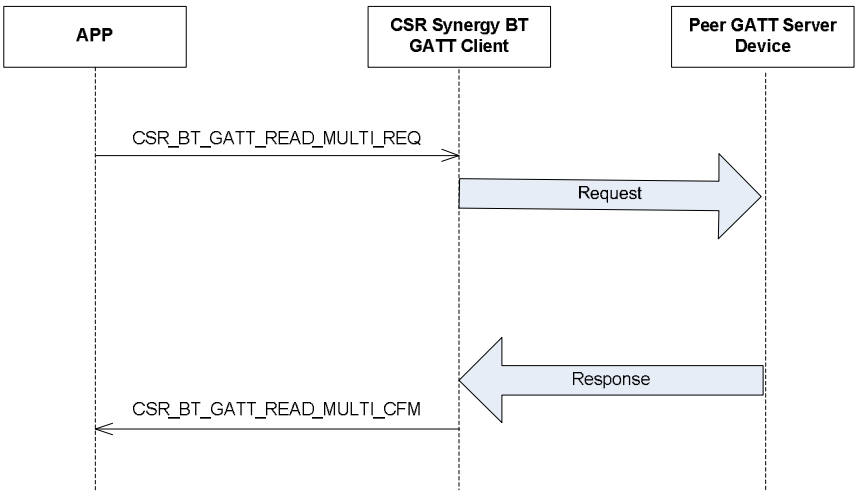


Figure 19: Read Characteristic Value by UUID

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_MULTI_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrUint16	valueLength	Length of the Characteristic Values in octets
CsrUint8	*value	Pointer to the Characteristic Values

Table 132: Members in a CSR_BT_GATT_READ_MULTI_CFM primitive

3.12 Read Characteristic Descriptor Procedures

A Client application is able to read seven different Characteristic Descriptors from a GATT Server (Peer/Local) when the client knows the handle of these descriptors.

3.12.1 Read Extended Properties

A Client application is able to read a Characteristic Extended Properties declaration, by calling one of the following functions:

- *CsrBtGattReadExtendedPropertiesReqSend*
- *CsrBtGattReadExtendedPropertiesLocalReqSend*

The function *CsrBtGattReadExtendedPropertiesReqSend* reads the Extended Properties declaration from a peer server and *CsrBtGattReadExtendedPropertiesLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 133 and Table 134.

Note the function *CsrBtGattReadExtendedPropertiesReqSend* shall only be used if the peer server has enabled the Characteristic Properties 'Extended Properties' (0x80) bit. The Characteristic Properties bit field is part of the Characteristic Declaration Attribute Value, see Table 66, which is described in section 3.9 and the definition of a Characteristic Value Declaration is described in Table 67.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle of the Characteristic Extended Properties declaration

Table 133: Arguments for `CsrBtGattReadExtendedPropertiesReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle of the Characteristic Extended Properties declaration

Table 134: Arguments for `CsrBtGattReadExtendedPropertiesLocalReqSend` function

Common to these two functions are that they will send a CSR_BT_GATT_READ_REQ primitive to GATT and as confirmation the application will receive a CSR_BT_GATT_READ_EXTENDED_PROPERTIES_CFM which includes the read Extended Properties declaration, as illustrated in Figure 20. The parameters for CSR_BT_GATT_READ_EXTENDED_PROPERTIES_CFM are described in Table 135. Please note, if the Client application is reading the Extended Properties declaration from a local device, it does not need to be connected.

The Client application is permitted to cancel this read procedure early. This can be done by calling *CsrBtGattCancelReqSend*, which parameters are described in Table 170. As confirmation, the application will still receive the CSR_BT_GATT_READ_EXTENDED_PROPERTIES_CFM message.

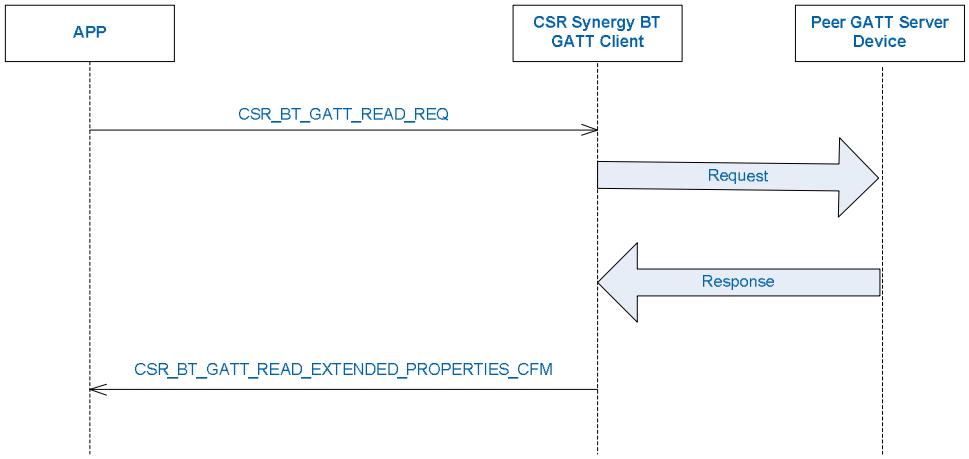


Figure 20: Read Extended Properties declaration

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_EXTENDED_PROPERTIES_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrBtGattReadExtended PropertiesBits	extProperties	The Characteristic Extended Properties bit field, which is a bit field that describes additional properties on how the Characteristic Value can be used, or how the characteristic Descriptors can be accessed. The declaration of an Extended Properties Descriptor is described in section 3.6 where also the CsrBtGattReadExtendedPropertiesBits are described, see Table 73. Note multiple extended property bits may be set and valid values are defined in csr_bt_gatt_prim.h.

Table 135: Members in a CSR_BT_GATT_READ_EXTENDED_PROPERTIES_CFM primitive

3.12.2 Read User Description

A Client application is able to read a Characteristic User Description declaration, by calling one of the following functions:

- *CsrBtGattReadUserDescriptionReqSend*
- *CsrBtGattReadUserDescriptionLocalReqSend*

The function *CsrBtGattReadUserDescriptionReqSend* reads the User Description declaration from a peer server and *CsrBtGattReadUserDescriptionLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 136 and Table 137.

Note, the function *CsrBtGattReadUserDescriptionReqSend* shall only be used if the peer server has enabled the 'Writeable Auxiliary' (0x002) bit, which is part of Characteristic Extended Properties bit field. The Extended Properties bit field is described in Table 73 and is part of the Extended Properties declaration, which

can be read by using the functions described in section 3.12.1.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle of the Characteristic User Description declaration

Table 136: Arguments for `CsrBtGattReadUserDescriptionReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle of the Characteristic User Description declaration

Table 137: Arguments for `CsrBtGattReadUserDescriptionLocalReqSend` function

Common to these two functions are that they will send a `CSR_BT_GATT_READ_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_READ_USER_DESCRIPTION_CFM`, as illustrated in Figure 21. The parameters for `CSR_BT_GATT_READ_USER_DESCRIPTION_CFM` are described in Table 138. Please note, if the Client application is reading from a local device, it does not need to be connected.

The Client application is permitted to cancel this read procedure early. This can be done by calling *`CsrBtGattCancelReqSend`*, of which parameters are described in Table 170. As confirmation, the application will still receive the `CSR_BT_GATT_READ_USER_DESCRIPTION_CFM` message.

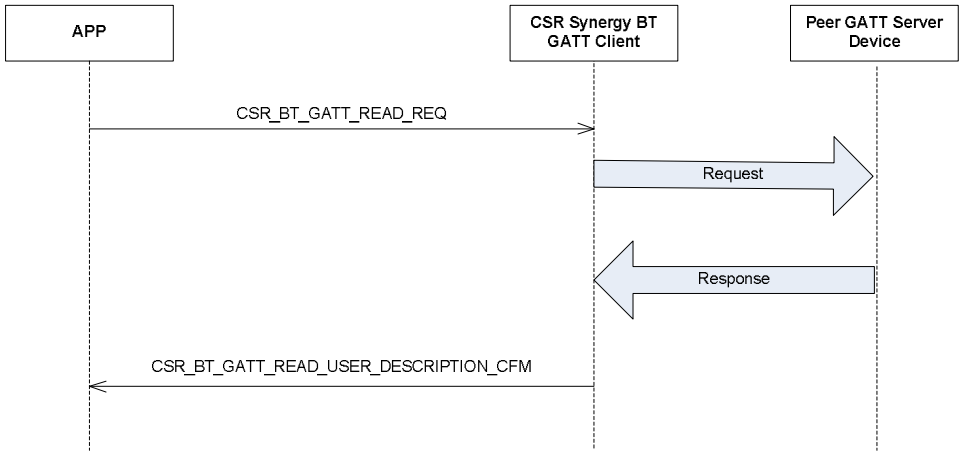


Figure 21: Read User Description declaration

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_USER_DESCRIPTION_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrUtf8String	*usrDescription	A zero terminate Characteristic User Description UTF-8 String

Table 138: Members in a CSR_BT_GATT_READ_USER_DESCRIPTION_CFM primitive

3.12.3 Read Client Configuration

A Client application is able to read a Client Characteristic Configuration declaration from a peer GATT server, by calling the function:

- *CsrBtGattReadClientConfigurationReqSend*

The arguments for this function is described in Table 139.

Note, the function *CsrBtGattReadClientConfigurationReqSend* shall only be used if the peer server has enabled either the 'Notify' (0x10) bit or the 'Indicate' (0x20) bit, which is part of Characteristic Properties bit field. The Characteristic Properties bit field is described in Table 70 and is part of the Characteristic declaration, which can be found by using the functions described in section 3.9. Also note that it is not possible to read the local value of the Client Characteristic Configuration declaration because the value is always shown by the application. For more information, please refer to section 3.6.9.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle of the Client Characteristic Configuration declaration

Table 139: Arguments for `CsrBtGattReadClientConfigurationReqSend` function

CsrBtGattReadClientConfigurationReqSend will send a CSR_BT_GATT_READ_REQ primitive to GATT and as confirmation the application will receive a CSR_BT_GATT_READ_CLIENT_CONFIGURATION_CFM, as illustrated in Figure 22. The parameters for CSR_BT_GATT_READ_CLIENT_CONFIGURATION_CFM are described in Table 140.

The Client application is permitted to cancel this read procedure early. This can be done by calling *CsrBtGattCancelReqSend*, of which parameters are described in Table 170. As confirmation, the application will still receive the CSR_BT_GATT_READ_CLIENT_CONFIGURATION_CFM message.

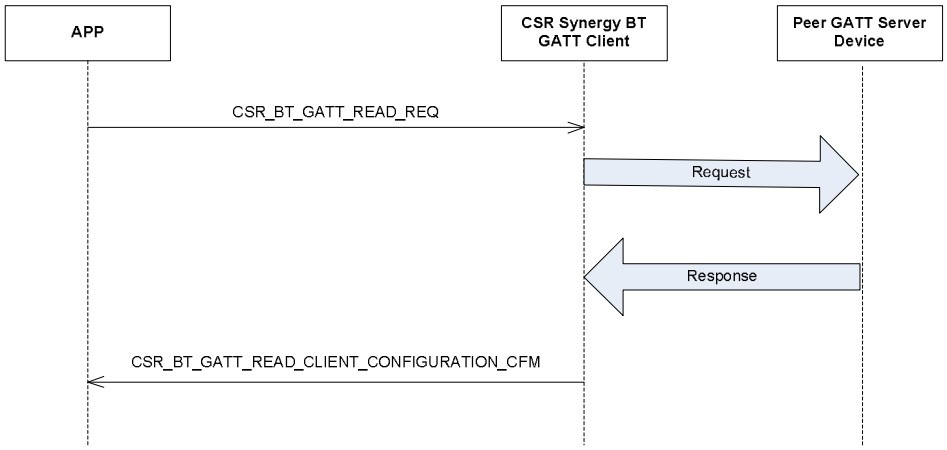


Figure 22: Read Client Configuration declaration

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_CLIENT_CONFIGURATION_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrBtGattCliConfBits	configuration	Client Characteristic Configuration bits, which are described in Table 165.

Table 140: Members in a CSR_BT_GATT_READ_CLIENT_CONFIGURATION_CFM primitive

3.12.4 Read Server Configuration

A Client application is able to read a Server Characteristic Configuration declaration, by calling one of the following functions:

- *CsrBtGattReadServerConfigurationReqSend*
- *CsrBtGattReadServerConfigurationLocalReqSend*

The function *CsrBtGattReadServerConfigurationReqSend* reads the Server Characteristic Configuration declaration from a peer server and *CsrBtGattReadServerConfigurationLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 141 and Table 142.

The function *CsrBtGattReadServerConfigurationReqSend* shall only be used if the peer server has enabled the 'Broadcast' (0x01) bit, which is part of Characteristic Properties bit field. The Characteristic Properties bit field is described in Table 70 and is part of the Characteristic declaration, which can be found by using the functions described in section 3.9.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle of the Server Characteristic Configuration declaration

Table 141: Arguments for `CsrBtGattReadServerConfigurationReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle of the Server Characteristic Configuration declaration

Table 142: Arguments for `CsrBtGattReadServerConfigurationLocalReqSend` function

Common to these two functions are that they will send a `CSR_BT_GATT_READ_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_READ_SERVER_CONFIGURATION_CFM`, as illustrated in Figure 23. The parameters for `CSR_BT_GATT_READ_SERVER_CONFIGURATION_CFM` are described in Table 143. Please note if the Client application is reading from a local device, it does not need to be connected.

The Client application is permitted to cancel this read procedure early. This can be done by calling *CsrBtGattCancelReqSend*, which parameters are described in Table 170. As confirmation, the application will still receive the CSR_BT_GATT_READ_SERVER_CONFIGURATION_CFM message.

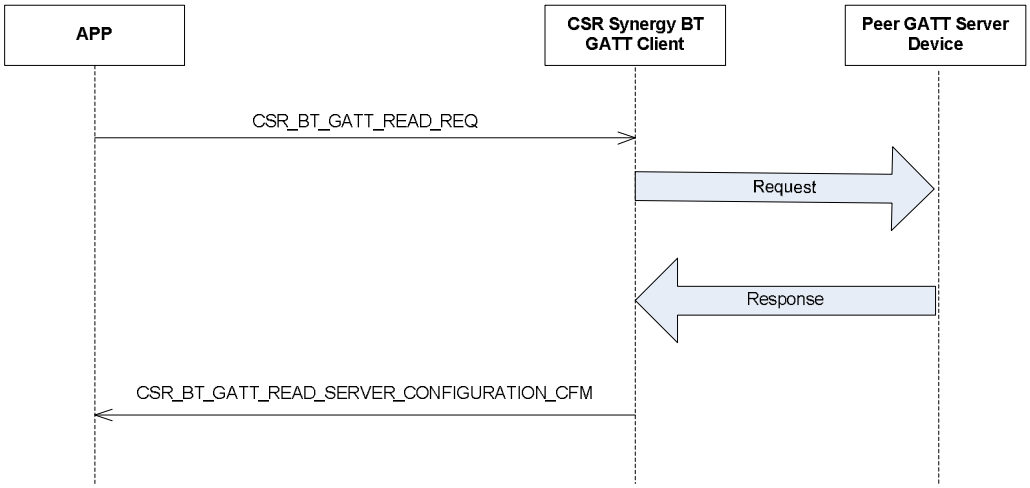


Figure 23: Read Server Configuration declaration

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_SERVER_CONFIGURATION_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrBtGattSrvConfigBits	configuration	Server Characteristic Configuration bits, which are described in Table 163.

Table 143: Members in a CSR_BT_GATT_READ_SERVER_CONFIGURATION_CFM primitive

3.12.5 Read Presentation Format

A Client application is able to read a Characteristic Presentation Format declaration, which is an optional declaration that defines the format of the Characteristic Value, by calling one of the following functions:

- *CsrBtGattReadPresentationFormatReqSend*
- *CsrBtGattReadPresentationFormatLocalReqSend*

The function *CsrBtGattReadPresentationFormatReqSend* reads the Presentation Format declaration from a peer server and *CsrBtGattReadPresentationFormatLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 144 and Table 145.

More than one Characteristic Presentation Format declaration may exist within a characteristic definition. If so a Characteristic Aggregate Format declaration shall also exist as part of the characteristic definition.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle of the Characteristic Presentation Formant declaration

Table 144: Arguments for `CsrBtGattReadPresentationFormatReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle of the Characteristic Presentation Formant declaration

Table 145: Arguments for `CsrBtGattReadPresentationFormatLocalReqSend` function

Common to these two functions are that they will send a `CSR_BT_GATT_READ_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_READ_PRESENTATION_FORMAT_CFM`, as illustrated on Figure 24. The parameters for `CSR_BT_GATT_READ_PRESENTATION_FORMAT_CFM` are described in Table 146. Please note, if the Client application is reading from a local device, it does not need to be connected.

The Client application is permitted to cancel this read procedure early. This can be done by calling *`CsrBtGattCancelReqSend`*, of which parameters are described in Table 170. As confirmation, the application will still receive the `CSR_BT_GATT_READ_PRESENTATION_FORMAT_CFM` message.

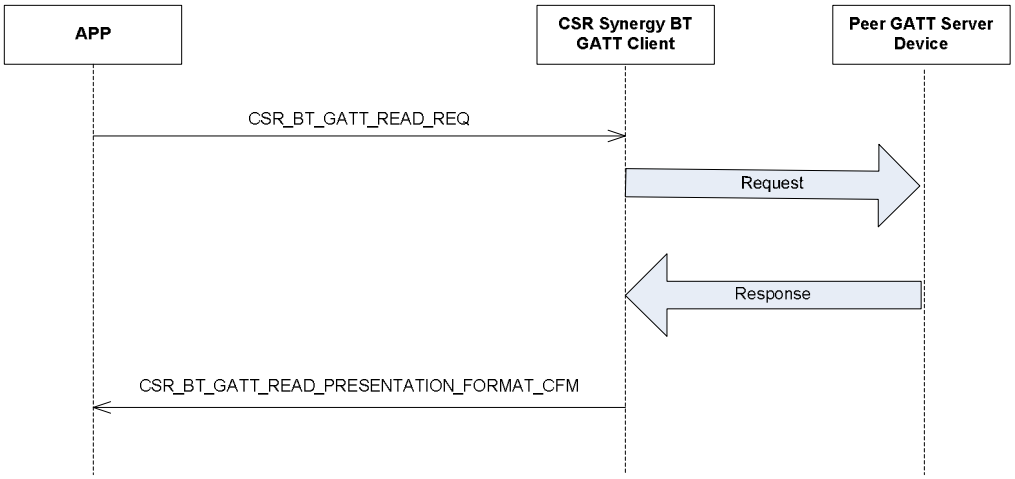


Figure 24: Read Presentation Format declaration

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_PRESENTATION_FORMAT_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrBtGattFormats	format	Format of the value of this characteristic, which are described in Table 84
CsrUint8	exponent	Exponent field that determines how the Characteristic Value is further formatted. The parameter is only used on integer format types that are present in Table 84. Actual value = Characteristic Value * $10^{exponent}$
CsrUint16	unit	The unit is a UUID defined in the Assigned Numbers Specification [2].
CsrUint8	nameSpace	The name space field indentify defined in the Assigned Numbers Specification [2].
CsrUint16	description	The description is an enumerated value as defined in the Assigned Numbers Specification [2].

Table 146: Members in a CSR_BT_GATT_READ_PRESENTATION_FORMAT_CFM primitive

3.12.6 Read Aggregate Format

A Client application is able to read a Characteristic Aggregate Format declaration, which is an optional declaration that defines the format of an aggregated Characteristic Value, by calling one of the following functions:

- *CsrBtGattReadAggregateFormatReqSend*
- *CsrBtGattReadAggregateFormatLocalReqSend*

Where the function *CsrBtGattReadAggregateFormatReqSend* reads the Aggregate Format declaration from a peer server and *CsrBtGattReadAggregateFormatLocalReqSend* does the same just on the local device. The arguments for the two functions are described in Table 147 and Table 148.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle of the Characteristic Aggregate Formant declaration

Table 147: Arguments for `CsrBtGattReadAggregateFormatReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle of the Characteristic Aggregate Formant declaration

Table 148: Arguments for `CsrBtGattReadAggregateFormatLocalReqSend` function

Common to these two functions are that they will send a CSR_BT_GATT_READ_REQ primitive to GATT and as confirmation the application will receive a CSR_BT_GATT_READ_AGGREGATE_FORMAT_CFM, as illustrated in Figure 25: Read Aggregate Format declaration. The parameters for CSR_BT_GATT_READ_AGGREGATE_FORMAT_CFM are described in Table 149. Please note if the Client application is reading from a local device, it does not need to be connected.

The Client application is permitted to cancel this read procedure early. This can be done by calling *CsrBtGattCancelReqSend*, of which parameters are described in Table 170. As confirmation, the application will still receive the CSR_BT_GATT_READ_AGGREGATE_FORMAT_CFM message.

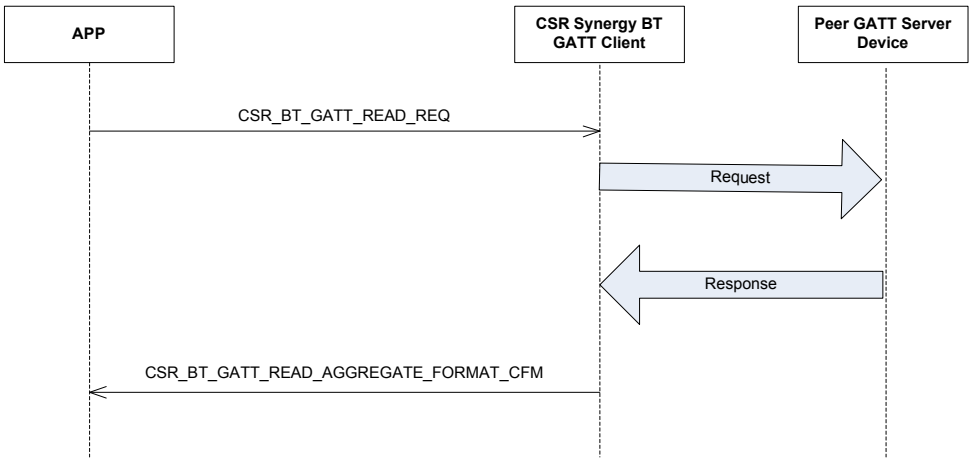


Figure 25: Read Aggregate Format declaration

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_AGGREGATE_FORMAT_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrUint16	handlesCount	Number of attribute handles
CsrBtGattHandle	*handles	List of Attribute Handles for the Characteristic Presentation Format Declarations

Table 149: Members in a CSR_BT_GATT_READ_AGGREGATE_FORMAT_CFM primitive

3.12.7 Read Profile Defined Descriptor

A Client application is able to read a Read a Characteristic Descriptor defined by a higher layer Profile, by calling one of the following functions:

- *CsrBtGattReadProfileDefinedDescriptorReqSend*
- *CsrBtGattReadProfileDefinedDescriptorLocalReqSend*

The function *CsrBtGattReadProfileDefinedDescriptorReqSend* reads a Profile defined Characteristic descriptor from a peer server and *CsrBtGattReadProfileDefinedDescriptorLocalReqSend* does the same only on the local device. The arguments for the two functions are described in Table 150 and Table 151.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle the Profile Defined Descriptor declaration
CsrUint16	offset	The offset of the first octet that shall be read

Table 150: Arguments for `CsrBtGattReadProfileDefinedDescriptorReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle the Profile Defined Descriptor declaration

Table 151: Arguments for `CsrBtGattReadProfileDefinedDescriptorLocalReqSend` function

Common to these two functions are that they will send a `CSR_BT_GATT_READ_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_READ_PROFILE_DEFINED_DESCRIPTOR_CFM`, as illustrated in Figure 26. The parameters for `CSR_BT_GATT_READ_PROFILE_DEFINED_DESCRIPTOR_CFM` are described in Table 152. Please note, if the Client application is reading from a local device, it does not need to be connected.

The Client application is permitted to cancel this read procedure early. This can be done by calling *CsrBtGattCancelReqSend*, of which parameters are described in Table 170. As confirmation, the application will still receive the `CSR_BT_GATT_READ_PROFILE_DEFINED_DESCRIPTOR_CFM` message.

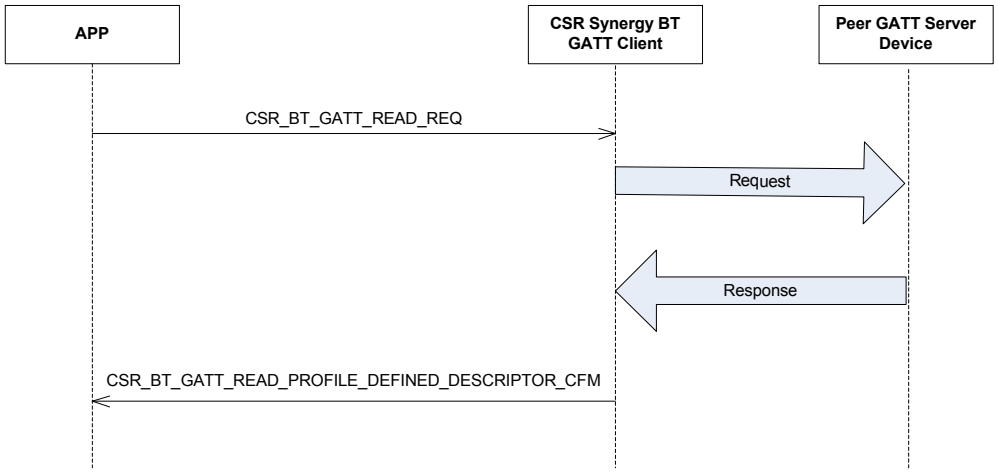


Figure 26: Read Profile Defined Descriptor declaration

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_READ_PROFILE_DEFINED_DESCRIPTOR_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier
CsrUint16	valueLength	Lentgh of the Profile defined Characteristic descriptor value in octets
CsrUint8	*value	Pointer to the Profile defined Characteristic descriptor value

Table 152: Members in a CSR_BT_GATT_READ_PROFILE_DEFINED_DESCRIPTOR_CFM primitive

3.13 Write Characteristic Value Procedures

A Client application can write a Characteristic Value to a GATT Server in four ways, if it knows the handle of the Characteristic Value. It can write it with and without getting a Response from the server, with Signed Write Without any Response from the server, and it can use Reliable Writes.

Common for all these procedures are that they will send a `CSR_BT_GATT_WRITE_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_WRITE_CFM`, as illustrated in Figure 27. The parameters for `CSR_BT_GATT_WRITE_CFM` are described in Table 153.

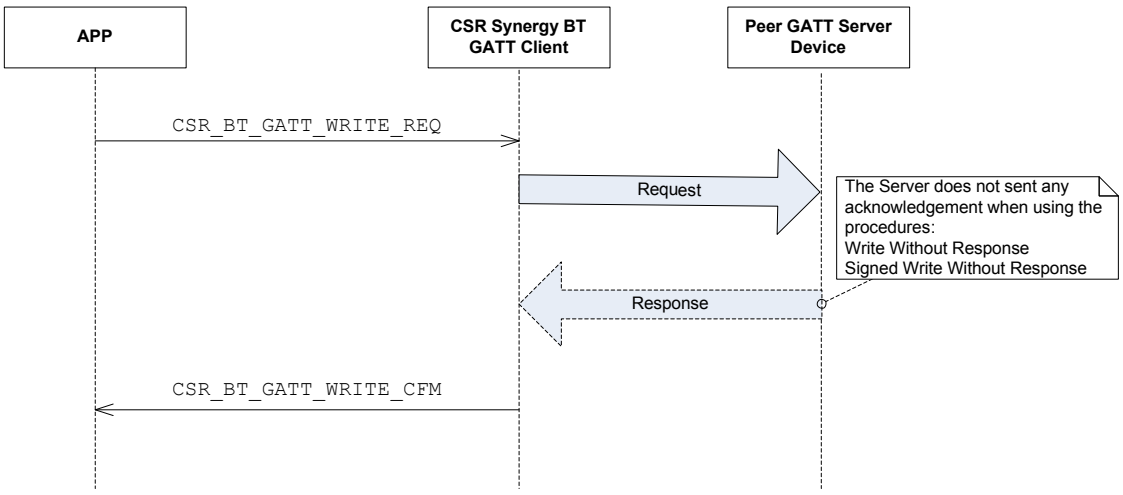


Figure 27: Write a Characteristic Value

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_WRITE_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 153: Members in a CSR_BT_GATT_WRITE_CFM primitive

3.13.1 Write Command

A Client application is able to write a Characteristic Value to a Peer GATT Server when the Client knows the handle of the Characteristic Value, by calling the function:

- *CsrBtGattWriteCmdReqSend*

This function shall only be used if the Client does not need any acknowledgement from the peer server, and if the peer server has enabled the 'Write Without Response' (0x04) bit, which is part of Characteristic Properties bit field. The Characteristic Properties bit field is described in Table 70 and is part of the Characteristic declaration, which can be found by using the functions described in section 3.9.

The arguments for *CsrBtGattWriteCmdReqSend* are described in Table 154, and as confirmation the application will receive a CSR_BT_GATT_WRITE_CFM, of which parameters are described in Table 153 .

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The Characteristic Value Handle
CsrUint16	valueLength	Length of the attribute value that must be written
CsrUint8	*value	An allocated pointer of the attribute value

Table 154: Arguments for `CsrBtGattWriteCmdReqSend` function

3.13.2 Write Signed Command

A Client application is able to write a Characteristic Value to a GATT Server when the Client knows the handle of the Characteristic Value and the physical connection is not encrypted, by calling the function:

- *CsrBtGattWriteSignedCmdReqSend*

This function shall only be used if:

- The Client does not need any acknowledgement from the peer server
- The Client has a trusted relationship with the server, e.g. the two devices are bonded.
- The physical connection is not encrypted,

- The peer server has enabled the 'Authenticated Signed Writes' (0x40) bit, which is part of Characteristic Properties bit field.

The Characteristic Properties bit field is described in Table 70 and is part of the Characteristic declaration, which can be found by using the functions described in section 3.9.

The arguments for *CsrBtGattWriteSignedCmdReqSend* are described in Table 155, and as confirmation the application will receive a CSR_BT_GATT_WRITE_CFM, of which parameters are described in Table 153 .

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The Characteristic Value Handle
CsrUint16	valueLength	Length of the attribute value that must be written
CsrUint8	*value	An allocated pointer of the attribute value

Table 155: Arguments for `CsrBtGattWriteSignedCmdReqSend` function

3.13.3 Write Request

A Client application is able to write a Characteristic Value on a GATT Server when the Client knows the handle of the Characteristic Value, by calling one of the following functions:

- *CsrBtGattWriteReqSend*
- *CsrBtGattWriteLocalReqSend*

The function *CsrBtGattWriteReqSend* writes a Characteristic Value to a peer server and shall only be used if the peer server has enabled the 'Write' (0x08) bit, which is part of Characteristic Properties bit field. The Characteristic Properties bit field is described in Table 70 and is part of the Characteristic declaration, which can be found by using the functions described in section 3.9.

The function *CsrBtGattWriteLocalReqSend* writes/updates a Characteristic Value to a local device. Please note if the Client application is writing to a local device, it does not need to be connected.

The arguments for the two functions are described in Table 156 and Table 157, and as confirmation the application will in both cases receive a CSR_BT_GATT_WRITE_CFM, of which parameters are described in Table 153 .

The Client application is permitted to cancel this write procedure early. This can be done by calling *CsrBtGattCancelReqSend*, of which parameters are described in Table 170. As confirmation, the application will still receive the CSR_BT_GATT_WRITE_CFM message.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The Characteristic Value Handle
CsrUint16	offset	The offset of the first octet to be written
CsrUint16	valueLength	Length of the attribute value that must be written
CsrUint8	*value	An allocated pointer of the attribute value

Table 156: Arguments for `CsrBtGattWriteReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The Characteristic Value Handle
CsrUint16	valueLength	Length of the attribute value that must be written
CsrUint8	*value	An allocated pointer of the attribute value

Table 157: Arguments for `CsrBtGattWriteLocalReqSend` function

3.13.4 Reliable Writes

A Client application is able to write a Characteristic Value or multiple Characteristic values with assurance to a GATT Server when the Client knows the handle(s) of the Characteristic Value(s), by calling the function:

- *CsrBtGattReliableWritesReqSend*

This function shall only be used if the peer server has enabled the 'Extended Properties' (0x80) bit, which is part of Characteristic Properties bit field, and it has enabled the 'Reliable Write' (0x0001) bit, which is part of Characteristic Extended Properties bit field.

The Characteristic Properties bit field is described in Table 70 and is part of the Characteristic declaration, which can be found by using the functions described in section 3.9, and the Extended Properties bit field is described in Table 73 and is part of the Extended Properties declaration, which can be read by using the functions described in section 3.12.1.

The arguments for *CsrBtGattReliableWritesReqSend* are described in Table 158, and as confirmation the application will receive a CSR_BT_GATT_WRITE_CFM, of which parameters are described in Table 153.

The Client application is permitted to cancel this write procedure early. This can be done by calling *CsrBtGattCancelReqSend*, of which parameters are described in Table 170. As confirmation, the application will still receive the CSR_BT_GATT_WRITE_CFM message.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrUint16	attrWritePairsCount	Number of Characteristic Values to be written
CsrBtGattAttrWritePairs	attrWritePairs	An allocated list of Characteristic Value Handles, offsets and Attribute Values

Table 158: Arguments for `CsrBtGattReliableWritesReqSend` function

3.14 Write Characteristic Descriptor Procedures

A Client application is able to write to four different Characteristic Descriptors on a GATT Server (Peer/Local) when the client knows the handle of these descriptors.

Common to all these procedures, except Write Client Configuration, are that they will send a `CSR_BT_GATT_WRITE_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_WRITE_CFM`, as illustrated on Figure 28. The parameters for `CSR_BT_GATT_WRITE_CFM` are described in Table 153.

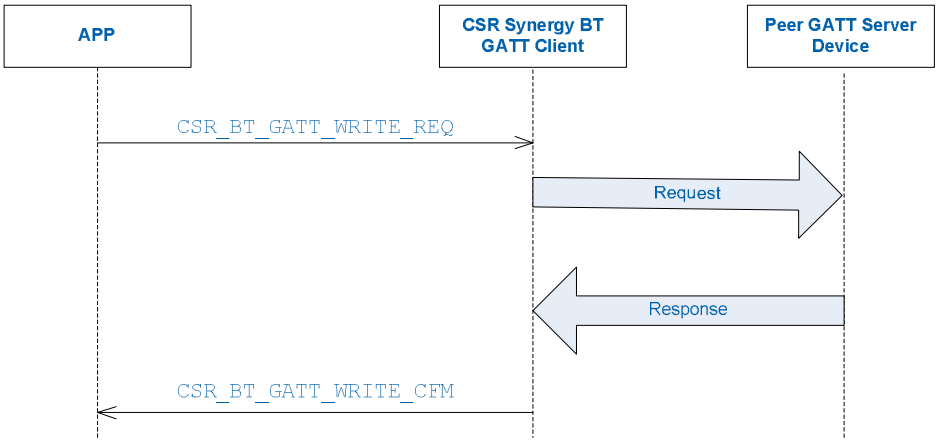


Figure 28: Write a Characteristic Descriptor Value

3.14.1 Write User Description

A Client application is able to write a Characteristic User Description value on a server, by calling one of the following functions:

- *CsrBtGattWriteUserDescriptionReqSend*
- *CsrBtGattWriteUserDescriptionLocalReqSend*

The function *CsrBtGattWriteUserDescriptionReqSend* writes the User Description value, which shall be a zero terminated UTF-8 string that defines a user textual description of the Characteristic Value, to a peer server and *CsrBtGattWriteUserDescriptionLocalReqSend* does the same just to the local device. The arguments for the two functions are described in Table 159 and Table 160.

Note the function *CsrBtGattWriteUserDescriptionReqSend* shall only be used if the peer server has enabled the 'Writeable Auxiliary' (0x002) bit, which is part of Characteristic Extended Properties bit field. The Extended Properties bit field is described in Table 73 and is part of the Extended Properties declaration, which can be read by using the functions described in section 3.12.1.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle of the Characteristic User Description declaration
CsrUtf8String	*utf8String	An allocated pointer, which contain a zero terminate Characteristic User Description UTF-8 String.

Table 159: Arguments for `CsrBtGattWriteUserDescriptionReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle of the Characteristic User Description declaration
CsrUtf8String	*utf8String	An allocated pointer, which contain a zero terminate Characteristic User Description UTF-8 String.

Table 160: Arguments for `CsrBtGattWriteUserDescriptionLocalReqSend` function

Common to these two functions are that they will send a `CSR_BT_GATT_WRITE_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_WRITE_CFM`. The parameters for

CSR_BT_GATT_WRITE_CFM are described in Table 153. Please note if the Client application is writing to a local device, it does not need to be connected.

The Client application is permitted to cancel this write procedure early. This can be done by calling *CsrBtGattCancelReqSend*, of which parameters are described in Table 170. As confirmation, the application will still receive the CSR_BT_GATT_WRITE_CFM message.

3.14.2 Write Server Configuration

A Client application is able to write a Server Characteristic Configuration value on a server, by calling one of the following functions:

- *CsrBtGattWriteServerConfigurationReqSend*
- *CsrBtGattWriteServerConfigurationLocalReqSend*

The function *CsrBtGattWriteServerConfigurationReqSend* writes the Server Characteristic Configuration value to a peer server and *CsrBtGattReadServerConfigurationLocalReqSend* does the same only to the local device.

The Server Characteristic Configuration value is a bit field that defines if the Characteristic Value shall be broadcasted or not. Changing the Server Characteristic Configuration value affects the configuration for all clients as the Server only has one instance of this descriptor which shall be shared by all clients. The arguments for the two functions are described in Table 161 and Table 162

The function *CsrBtGattWriteServerConfigurationReqSend* shall only be used if the peer server has enabled the 'Broadcast' (0x01) bit, which is part of Characteristic Properties bit field. The Characteristic Properties bit field is described in Table 70 and is part of the Characteristic declaration, which can be found by using the functions described in section 3.9.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtGattHandle	handle	The handle of the Server Characteristic Configuration declaration.
CsrBtGattSrvConfigBits	configBits	The Server Characteristic Configuration bits, which are described is Table 163, are defined in csr_bt_gatt_prim.h.

Table 161: Arguments for `CsrBtGattWriteServerConfigurationReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle of the Server Characteristic Configuration declaration.
CsrBtGattSrvConfigBits	configBits	The Server Characteristic Configuration bits, which are describe is Table 163, are defined in csr_bt_gatt_prim.h.

Table 162: Arguments for `CsrBtGattWriteServerConfigurationLocalReqSend` function

The Server Characteristic Configuration bit field value is described in Table 163.

Configuration	Value	Description
Disable	CSR_BT_GATT_SERVER_CHARAC_CONFIG_DISABLE (0x0000)	Disable Broadcast
Broadcast	CSR_BT_GATT_SERVER_CHARAC_CONFIG_BROADCATS (0x0001)	The Characteristic Value shall be broadcasted when the server is in the broadcast procedure if advertising data resources are available.
Reserved for Future Use	0xFFF2	Reserved for Future Use

Table 163: Server Characteristic Configuration bit field definition

Common to these two functions are that they will send a `CSR_BT_GATT_WRITE_REQ` primitive to GATT and as confirmation the application will receive a `CSR_BT_GATT_WRITE_CFM`. The parameters for `CSR_BT_GATT_WRITE_CFM` are described in Table 153. Please note if the Client application is writing to a local device, it does not need to be connected.

Note if the peer server is configured to enable broadcast the server will start to advertise the Characteristic Value. In order for the Client to receive these advertising messages it shall set the local devices into scan mode, by using one of the procedures described in section 3.2.2. The client may then start to receive `CSR_BT_GATT_REPORT_IND` messages from GATT, as illustrated in Figure 29. The parameters for `CSR_GATT_REPORT_IND` are described in Table 16. Similarly, if the peer server is configured to disable broadcast it is up to the Client application to stop scanning again. Scanning is stopped again using one of the procedures described in section 3.2.2.

The application can extract EIR type information from a CSR_BT_GATT_REPORT_IND message by using the function *`CsrBtGattUtilGetEirInfo`* which is defined in csr_bt_gatt_utils.h.

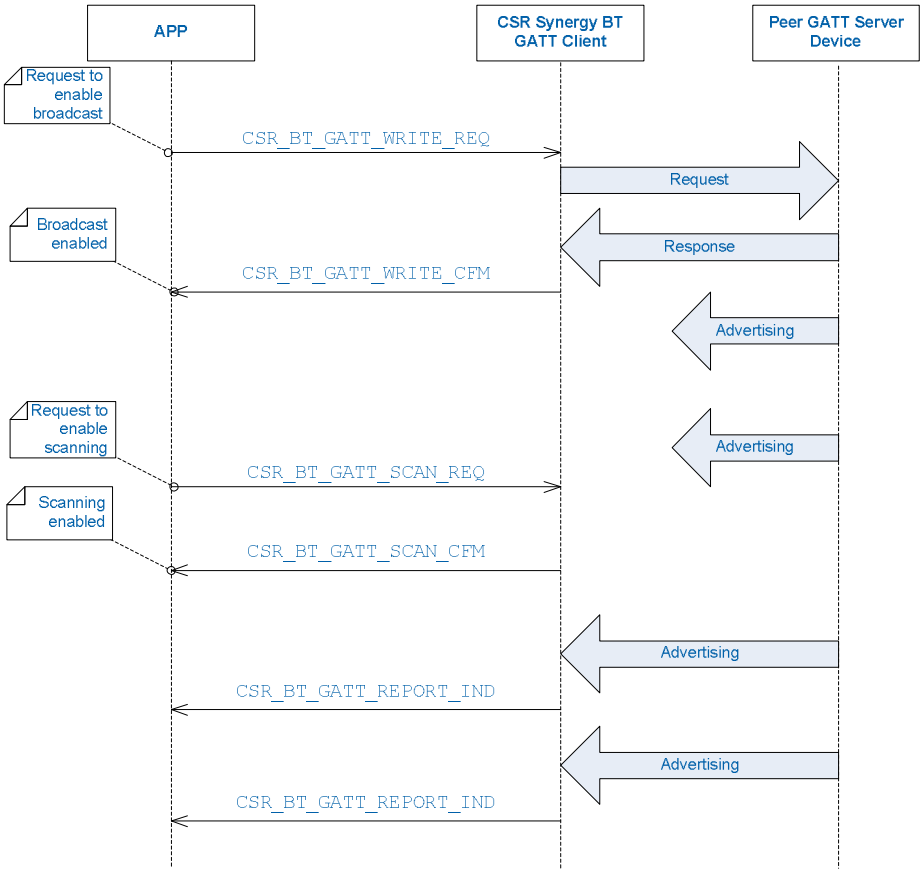


Figure 29: Configure Peer Server to enable broadcast of a Characteristic Value

3.14.3 Write Client Configuration

A Client application is able to write a Client Characteristic Configuration value on a peer server, by calling the function:

- *CsrBtGattWriteClientConfigurationReqSend*

This function writes the Client Characteristic Configuration value to a peer server, and the function arguments are described in Table 164.

The Client Characteristic Configuration value is a bit field. When a bit is set, the server shall either notify or indicate the change of the Characteristic Value to the client. When no bit is set it shall not be used. Changing the Client Characteristic Configuration value only affects the configuration of this specific client, as each client shall have its own instance of the server.

The function *CsrBtGattWriteServerConfigurationReqSend* shall only be used if the peer server has either enabled the 'Notify' (0x10) bit, the 'Indicate' (0x20) bit, or both, which are part of Characteristic Properties bit field. The Characteristic Properties bit field is described in Table 70 and is part of the Characteristic declaration, which can be found by using the functions described in section 3.9.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	The connection identifier
CarBtGattHandle	cliConfHdl	The handle of the Client Characteristic Configuration declaration
CsrBtGattHandle	valueHandle	The characteristic value handle. E.g. the attribute handle of the Characteristic Value declaration
CsrBtGattCliConfigBits	configBits	The Client Characteristic Configuration bits, which are described is Table 165, are defined in csr_bt_gatt_prim.h.

Table 164: Arguments for `CsrBtGattWriteClientConfigurationReqSend` function

The Client Characteristic Configuration bit field value is described in Table 165. Note the client is only allowed to set one bit.

Configuration	Value	Description
Disabled	CSR_BT_GATT_CLIENT_CHARAC_CONFIG_DEFAULT (0x000)	The characteristic Value shall not be notified or indicated
Notification	CSR_BT_GATT_CLIENT_CHARAC_CONFIG_NOTIFICATION (0x0001)	The Characteristic Value shall be notified
Indication	CSR_BT_GATT_CLIENT_CHARAC_CONFIG_INDICATION (0x0002)	The Characteristic Value shall be indicated
Reserved for Future Use	0xFFF4	Reserved for Future Use

Table 165: Client Characteristic Configuration bit field definition

CsrBtGattWriteClientConfigurationReqSend will send a CSR_BT_GATT_WRITE_CLIENT_CONFIGURATION_REQ primitive to GATT and as confirmation the application will receive a CSR_BT_GATT_WRITE_CFM. The parameters for CSR_BT_GATT_WRITE_CFM are described in Table 153. If the peer server is configured to notify or indicate the change of the Characteristic Value the Client will start to receive CSR_BT_GATT_NOTIFICATION_IND messages from GATT, as illustrated in Figure 30. The parameters for CSR_BT_GATT_NOTIFICATION_IND are described in Table 166.

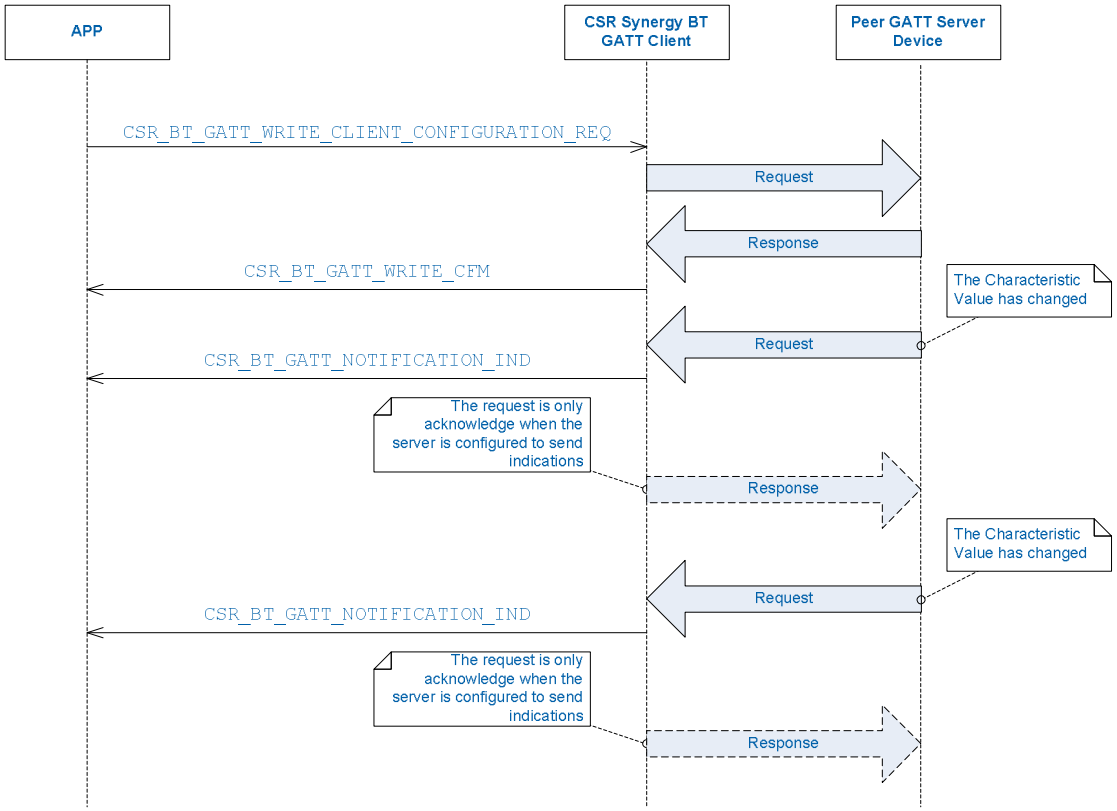


Figure 30: Configure Peer Server to notify or indicate the change of a Characteristic Value

Type	Member	Description
CsrBtGattPrim	Type	Signal identity – always set to CSR_BT_GATT_NOTIFICATION_IND
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	Connection identifier
CsrBtTypeAddr	Address	Peer address
CsrBtGattHandle	valueHandle	Attribute Handle of the Characteristic Value
CsrUint16	valueLength	Length of the Characteristic Value in octets
CsrUint8	*value	Pointer to the Characteristic Value

Table 166: Members in a CSR_BT_GATT_NOTIFICATION_IND primitive

The Client Characteristic Configuration value shall be persistent across connection for bonded devices. E.g. the Client is paired/bonded with a peer server and it has previously configured this server to notify or indicate the change of the Characteristic Value the client may start receiving CSR_BT_GATT_NOTIFICATION_IND messages from GATT as soon a physical connection between these two devices is established. For more information, refer to section 3.6.9.

Note that if the client application has configured the server to send a notification or indication, the client application shall allow re-establishment of the connection when it is disconnected. If the client is disconnected, but intends to become a Central in the connection it shall start one of the connection establishment procedures, described in section 3.3.1 or section 3.3.3. If the client is disconnected, but intends to become a Peripheral in the connection it shall be set connectable by using the procedures described in section 0 or section 3.3.4 .

The application is responsible for implementing the persistent storage of this feature. E.g. if the GATT module is reset/shutdown and the Client is paired/bonded with a peer server, which it has previously configured to notify or indicate the change of the Characteristic Value, the client application shall re-enable this feature locally by calling the function *CsrBtGattWriteClientConfigurationConnLessReqSend*. If the Application does not re-enable the feature after a reset/shutdown GATT does not know to which client application the incoming server events shall be sent to, and therefore it has to discharge the incoming event. The arguments for

CsrBtGattWriteClientConfigurationConnLessReqSend are described in Table 167, and it will send a CSR_BT_GATT_WRITE_CLIENT_CONFIGURATION_REQ primitive to GATT and as confirmation the application will receive a CSR_BT_GATT_WRITE_CFM. The parameters for CSR_BT_GATT_WRITE_CFM are described in Table 153.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtTypeAddr	address	The device address of the peer device
CsrBtGattHandle	valueHandle	The characteristic value handle. E.g. the attribute handle of the Characteristic Value declaration which were previously set by using the function CsrBtGattWriteClientConfigurationReqSend
CsrBtGattCliConfigBits	configBits	The Client Characteristic Configuration bit field value. Shall be set to the same value as the one that was previously set using the function CsrBtGattWriteClientConfiguationReqSend.

Table 167: Arguments for `CsrBtGattWriteClientConfigurationConnLessReqSend` function

3.14.4 Write Profile Defined Descriptor

A Client application is able to write to a Characteristic Descriptor that is defined by a higher layer Profile, by calling one of the following functions:

- *CsrBtGattWriteProfileDefinedDescriptorReqSend*
- *CsrBtGattReadProfileDefinedDescriptorLocalReqSend*

The function *CsrBtGattWriteProfileDefinedDescriptorReqSend* writes a Profile defined Characteristic descriptor Value on a peer server and *CsrBtGattReadProfileDefinedDescriptorLocalReqSend* does the same only on the local device. The arguments for the two functions are described in Table 168 and Table 169.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	The connection identifier
CsrBtGattHandle	handle	The handle of the Profile Defined Description declaration
CsrUint16	offset	The offset of the first octet to be written
CsrUint16	valueLength	Length of the attribute value that must be written
CsrUint8	*value	An allocated pointer of the attribute value

Table 168: Arguments for `CsrBtGattWriteProfileDefinedDescriptorReqSend` function

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattHandle	handle	The handle of the Profile Defined Description declaration
CsrUint16	valueLength	Length of the attribute value that must be written
CsrUint8	*value	An allocated pointer of the attribute value

Table 169: Arguments for `CsrBtGattWriteProfileDefinedDescriptorLocalReqSend` function

Common to these two functions are that they will send a CSR_BT_GATT_WRITE_REQ primitive to GATT and as confirmation the application will receive a CSR_BT_GATT_WRITE_CFM. The parameters for CSR_BT_GATT_WRITE_CFM are described in Table 153. Please note if the Client application is writing to a local device, it does not need to be connected.

The Client application is permitted to cancel this write procedure early. This can be done by calling *CsrBtGattCancelReqSend*, of which parameters are described in Table 170. As confirmation, the application will still receive the CSR_BT_GATT_WRITE_CFM message.

3.15 Cancel Procedure

A Client application may cancel an ongoing GATT procedure, by calling function:

- *CsrBtGattCancelReqSend*

of which arguments are described in Table 170. As confirmation the application will receive the confirm message of the GATT procedure being cancelled. Note the following procedures described in section 3.7, section 3.8, section 3.9, section 3.10, section 3.11, section 3.12, section 3.13 and section 3.14 may be cancelled.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	The connection identifier

Table 170: Arguments for `CsrBtGattCancelReqSend` function

3.16 Service Changed Indication

If the client application is bonded with a server it is able to read the database once and use the information across reconnections without reading the database again. In the case that the server changes its database the client application will receive a `CSR_BT_GATT_SERVICE_CHANGE_IND` message, indicating that it needs to rediscover the database information. The parameters for `CSR_BT_GATT_SERVICE_CHANGE_IND` are described in Table 171.

Note, if the Client is not bonded with a server it shall read the database at each reconnection.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_SERVICE_CHANGED_IND
CsrBtGattId	gattId	The application identifier
CsrBtTypeAddr	address	Peer address
CsrUint16	startHandle	Start of Affected Attribute Handle Range
CsrUint16	endHandle	End of Affected Attribute Handle Range

Table 171: Members in a CSR_BT_GATT_SERVICE_CHANGED_IND primitive

3.17 Subscribing Procedures

This procedure allows the Client application to subscribe/unsubscribe for indication or notification events from a server when a Client Characteristic Configuration declaration does not exist.

Note, whenever possible is it strongly recommended to use the procedure described in section 3.14.3, as the subscribing feature should only be used if a Characteristic definition on a peer server does not include a Client Characteristic Configuration descriptor. Further, the server sends notification or indication event every time the Characteristic Value within the Characteristic definition has been changed and the application needs these events.

A GATT Client is able to receive indication or notification events, see Table 166, from a server when a Client Characteristic Configuration declaration does not exist, by calling:

- *CsrBtGattSubscribeReqSend*

The arguments for *CsrBtGattSubscribeReqSend* are described in Table 172.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattAddress	address	The device address of the peer device
CsrBtGattHandle	handle	The handle of the Characteristic value

Table 172: Arguments for `CsrBtGattSubscribeReqSend` function

If the Client previously has called *CsrBtGattSubscribeReqSend* the client can request GATT to stop receiving indication or notification events, by calling:

- *CsrBtGattUnsubscribeReqSend*

The arguments for *CsrBtGattUnsubscribeReqSend* are described in Table 173.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtGattAddress	address	The device address of the peer device
CsrBtGattHandle	handle	The handle of the Characteristic value

Table 173: Arguments for `CsrBtGattUnsubscribeReqSend` function

Common to these two functions are that they will send a `CSR_BT_GATT_SUBSCRIPTION_REQ` primitive to GATT. When GATT has processed this request a `CSR_BT_GATT_SUBSCRIPTION_CFM` primitive is sent to the application, see Table 174.

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_SUBSCRIPTION_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 174: Members in a CSR_BT_GATT_SUBSCRIPTION_CFM primitive.

3.18 Set Event Mask Procedures

A GATT application can subscribe for extended information, by calling function:

- *CsrBtGattSetEventMaskReqSend*

This function, of which arguments are described in Table 175, will send a `CSR_BT_GATT_SET_EVENT_MASK_REQ` primitive to GATT. When GATT has processed this request a `CSR_BT_GATT_SET_EVENT_MASK_CFM` message is sent back to the application as confirmation, see Table 176.

<table>
<tr><th>Type</th><th>Argument</th><th>Description</th></tr>
<tr><td>CsrBtGattId</td><td>gattId</td><td>The application identifier</td></tr>
<tr><td>CsrBtGattEventMask</td><td>eventMask</td><td>Defines the event(s) to subscribe for. The following event mask values are defined in csr_bt_gatt_prim.h.
<table>
<tr><th>Value</th><th>Parameter description</th><th>Reference</th></tr>
<tr><td>CSR_BT_CM_EVENT_MASK_SUBSCRIBE_NONE</td><td>Stop subscribing for any extended information.</td><td>N/A</td></tr>
<tr><td>CSR_BT_GATT_EVENT_MASK_SUBSCRIBE_PHYSICAL_LINK_STATUS</td><td>Physical connection events</td><td>Table 177</td></tr>
<tr><td>CSR_BT_GATT_EVENT_MASK_SUBSCRIBE_WHITELIST_CHANGE</td><td>Whitelist change events</td><td>Table 178</td></tr>
<tr><td>CSR_BT_GATT_EVENT_MASK_SUBSCRIBE_PARAM_CONN_UPDATE_IND</td><td>Update Connection parameter events</td><td>Section 3.3.5</td></tr>
</table></td></tr>
</table>

Table 175: Arguments for `CsrBtGattSetEventMaskReqSend` function

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_SET_ EVENT_MASK_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 176: Members in a CSR_BT_GATT_SET_EVENT_MASK_CFM primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_PHYSICAL_LINK_STATUS_IND
CsrBtGattId	gattId	The application identifier
CsrBtTypeAddr	address	Peer address
CsrBtGattConnInfo	connInfo	Connection info flags (radio type, etc.), which are defined in csr_bt_gatt_prim.h.
CsrBool	status	TRUE = connected, FALSE = disconnected

Table 177: Members in a CSR_BT_GATT_PHYSICAL_LINK_STATUS_IND primitive

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_WHITELIST_CHANGE_IND
CsrBtGattId	gattId	The application identifier

Table 178: Members in a CSR_BT_GATT_WHITELIST_CHANGE_IND primitive

3.19 Initiating Authentication

A GATT application is able to start the LE authentication procedure, by calling function:

- *CsrBtGattSecurityReqSend*

of which arguments are described in Table 179. As confirmation, the application will receive a CSR_BT_GATT_SECURITY_CFM, of which parameters are described in Table 180.

Note this function shall only be use when running on a LE physical link.

Type	Argument	Description
CsrBtGattId	gattId	The application identifier
CsrBtConnId	btConnId	The connection identifier
CsrBtGattSecurityFlags	secRequirements	The security flags, which are define in csr_bt_gatt_prim.h. Valid values are: CSR_BT_GATT_SECURITY_FLAGS_DEFAULT : Default LE authentication requirement. CSR_BT_GATT_SECURITY_FLAGS_UNAUTHENTICATED Encrypt the LE link. CSR_BT_GATT_SECURITY_FLAGS_AUTHENTICATED Encrypt the LE link with MITM protection Note the Security Controller (SC) controls whether or not the bonding flag is set. The bonding flag is set if the the value of the key distribution parameters for Low Energy security is different from 0. The default value of the key distribution is defined by CSR_BT_SC_KEY_DIST_DEFAULT in csr_bt_usr_config_default.h, and may be change runtime by using CsrBtScLeKeyDistributionReqSend, see csr_bt_sc_lib.h

Table 179: Arguments for `CsrBtGattSecurityReqSend` function

Type	Member	Description
CsrBtGattPrim	type	Signal identity – always set to CSR_BT_GATT_SECURITY_CFM
CsrBtGattId	gattId	The application identifier
CsrBtResultCode	resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_GATT then the possible result codes can be found in csr_bt_gatt_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
CsrBtSupplier	resultsupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Connection identifier

Table 180: Members in a CSR_BT_GATT_SECURITY_CFM primitive

4 Document References

Document	Reference
Bluetooth® Core Specification Version 4.0 Volume 3 Part G www.bluetooth.org	[1]
Bluetooth® Assigned Numbers Specification: https://www.bluetooth.org/Technical/AssignedNumbers/home.htm	[2]
CSR Synergy Bluetooth, SD – Service Discovery API Description, api-0103-sd.pdf.	[3]
Bluetooth® Core Specification Version 4.0 Volume 3 Part C www.bluetooth.org	[4]

Terms and Definitions

GATT	Generic Attribute Profile
UUID	Universally Unique IDentifier
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
CSR	Cambridge Silicon Radio
PDU	Protocol Data Unit

Document History

Revision	Date	History
1	26 SEP 11	Ready for release 18.2.0

TradeMarks, Patents and Licences

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

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.

Life Support Policy and Use in Safety-critical Compliance

CSR's products are not authorised for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. CSR will not warrant the use of its devices in such applications.

Performance and Conformance

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