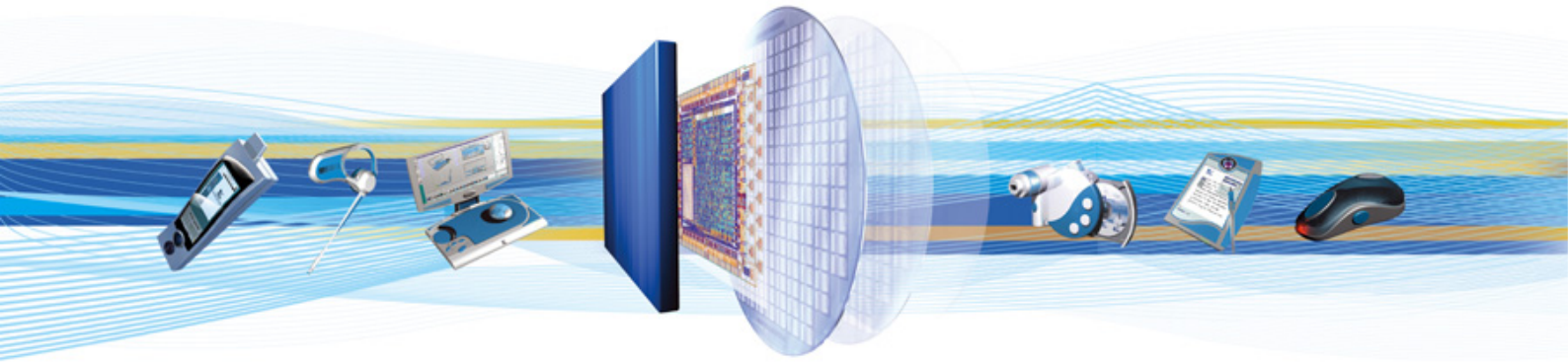
CSR Synergy Bluetooth 18.2.2

Proximity Server

API Description

January 2012



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 Bluetooth Low Energy(BLE) Proximity Service Server – as specified by the Bluetooth® Special Interest Group (SIG).

2 Description

2.1 Introduction

The BLE Proximity Server Service extends the GATT with the functionality needed to make a complete proximity server device. The server is implemented to give the implementer the easiest possible way to implement the proximity service without having to care about the full GATT API. This means that after activation, the server task will run in the background and keep serving incoming connection on BR/EDR and LE until it is deactivated.

The Proximity server makes it possible to implement the proximity server profile easily and provides a simple interface for the application to subscribe to relevant events. It handles both the setup of the database and the connection handling.

The following Low Energy services are supported by the database in the server:

- Link Loss Service
- Immediate Alert Service
- Tx Power Service
- Battery Service

The functionality of each of the services are defined in the Bluetooth Low Energy Proximity profile documentation as provided by the Bluetooth SIG.

2.2 Requirements

In order for the Proximity Server to work, the Synergy Bluetooth stack needs to be compiled with CSR_BT_LE_ENABLE=1 . Besides this, the GATT task needs to be included and enabled in the scheduler.

The Bluetooth chip needs to be compatible with the Synergy Bluetooth Low Energy enabled host stack.

2.3 Reference Model

The reference model for a proximity server is depicted in Figure 1 below. The server is placed between the GATT API and the application in order to provide a simplified API for the application. The Application does not have do have any communication directly with the GATT task if only the proximity server service is needed.

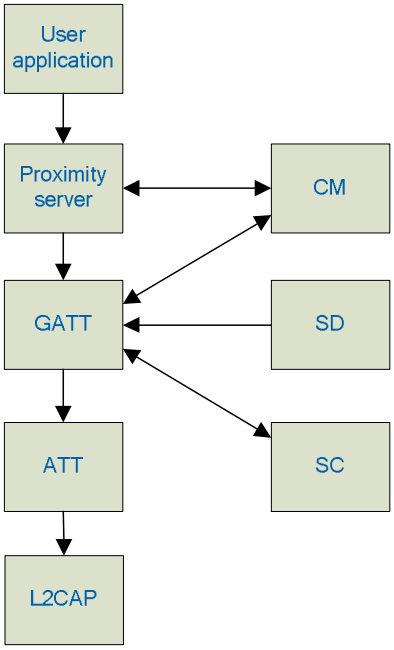


Figure 1: Proximity Server reference model

3 Interface Description

This chapter documents the public API of the Proximity Server service. The implementation follows the normal guidelines for Synergy Bluetooth task implementations.

3.1 Activate

In order to use the proximity server, it needs to be activated. Activating the server will make it start advertising on both the BR/EDR and LE radio and wait for incoming connections.

If a device connects to either of the radios, then the advertisement on the other radio will stop. When the device disconnects, the proximity server will start advertising on both radios again.

In order to provide the option to store Characteristic Client Configurations in a persistent store, the deactivate confirm signal returns a client configuration datablob. When you re-activate the server, it is possible to provide a pointer to this datablob and thereby restore the previous Characteristics Client Configuration state.

It is possible to provide an event mask to tell the server which type of event notifications the app would like to receive. See the csr_prox_srv_prim.h for definition of the event mask types.

To send the `CSR_BT_PROX_SRV_ACTIVATE_REQ` primitive, the `CsrBtProxSrvActivateReqSend()` function is used. The function takes the arguments described in Table 1.

Type	Argument	Description
CsrSchedQid	appHandle	Application handle for the app initializing the proximity server
CsrUint16	clientConfigSize	If a client config data blob is added, this is the size in bytes
CsrUint8	*clientConfig	Client config data blob returned in a previous deactivation
CsrBtProxSrvEventMask	eventMask	Mask indicating which events the app would like to subscribe to

Table 1: Arguments for `CsrBtProxSrvAvtivateReqSend` function

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

Type	Member	Description
CsrBtProxSrvPrim	type	Signal identity – always set to CSR_BT_PROX_SRV_ACTIVATE_CFM
CsrBtGattId	gattId	The ID the GATT profile uses to identify the proximity server
CsrUint16	dbStartHandle	The database start handle allocated for the proximity server
CsrUint16	dbEndHandle	The database end handle allocated for the proximity server
CsrBtResultCode	resultCode	Result code returned by the supplier in resultSupplier
CsrBtSupplier	resultSupplier	The result supplier – CSR_BT_SUPPLIER_PROX_SRV is success

Table 2: Members in a CSR_BT_PROX_SRV_ACTIVATE_CFM primitive

3.2 Deactivate

The server can be deactivated by the application at any time during use. When deactivated, the server will disconnect any existing connections and clean up state information.

In order to provide the option to store Characteristic Client Configurations in a persistent store, the deactivate confirm signal returns a client config datablob. When you re-activate the server, it is possible to provide a pointer to this datablob and thereby restore the previous Characteristics Client Configuration state.

To send the `CSR_BT_PROX_SRV_DEACTIVATE_REQ` primitive, the `CsrBtProxSrvDeactivateReqSend()` function is used.

When the deactivation request has been processed, a `CSR_BT_PROX_SRV_DEACTIVATE_CFM` primitive will be sent back to the application. The primitive members are described in Table 3.

Type	Member	Description
CsrBtProxSrvPrim	type	Signal identity – always set to CSR_BT_PROX_SRV_CFM
CsrUint16	clientConfigSize	Size of the client config data blob returned
CsrUint8	*clientConfig	Client config data blob for later re-addition in another activate
CsrBtResultCode	resultCode	Result code returned by the supplier in resultSupplier
CsrBtSupplier	resultSupplier	The result supplier – CSR_BT_SUPPLIER_PROX_SRV is success

Table 3: Members in a CSR_BT_PROX_SRV_DEACTIVE_CFM primitive

3.3 Connect and Disconnect

If a peer device connects to the server, a connect indication signal is sent to the application. This provides information about the connection, the peer device addres and type. No reply is needed for this signal.

The parameters for the `CSR_BT_PROX_SRV_CONNECT_IND` primitive are described in Table 4.

Type	Argument	Description
CsrBtProxSrvPrim	Type	Signal identity – always set to CSR_BT_PROX_SRV_CONNECT_IND
CsrBtTypedAddr	deviceAddr	Device address for peer device
CsrBtConnId	btConnId	ID identifying the Bluetooth connection in use
CsrBtResultCode	resultCode	Result code returned by the supplier in resultSupplier
CsrBtSupplier	resultSupplier	The result supplier – CSR_BT_SUPPLIER_PROX_SRV is success

Table 4: Members in a CSR_BT_PROX_SRV_CONNECT_IND primitive

If a peer device disconnects from the server, a disconnect indication signal is sent to the application. This provides information about the reason for the disconnect and who disconnected. No reply is needed for this signal.

The parameters for the `CSR_BT_PROX_SRV_DISCONNECT_IND` primitive are described in Table 5.

Type	Argument	Description
CsrBtProxSrvPrim	type	Signal identity – always set to CSR_BT_PROX_SRV_DISCONNECT_IND
CsrBtConnId	btConnId	Device address for peer device
CsrBtTypedAddr	deviceAddr	ID identifying the Bluetooth connection in use
CsrBtResultCode	reasonCode	Reason code returned by the supplier in reasonSupplier
CsrBtSupplier	reasonSupplier	The reason supplier – CSR_BT_SUPPLIER_PROX_SRV is success

Table 5: Members in a CSR_BT_PROX_SRV_DISCONNECT_IND primitive

3.4 Update TX Power

In order for the server to have the latest information about the Tx Power level, the application needs to send down this information whenever it thinks it needs updating. The interval for updating this is not set to anything specific as it is a task of the application to decide on this matter. If no Tx Power value is sent to the proximity server task, the default value of 0 is provided to peer devices requesting this value.

Note that Tx Power values are only relevant on BR/EDR connections as it is does not change on an LE connection.

To send the `CSR_BT_PROX_SRV_UPDATE_TX_POWER_REQ` primitive, the `CsrBtProxSrvUpdateTxPowerReqSend()` function is used. The function takes the arguments described in Table 6.

Type	Argument	Description
CsrUint16	txPower	The new TX power value

Table 6: Arguments for `CsrBtProxSrvUpdateTxPowerReqSend` function

When the update Tx power request has been processed, a `CSR_BT_PROX_SRV_UPDATE_TX_POWER_CFM` primitive will be sent back to the application. The primitive members are described in Table 7.

Type	Member	Description
CsrBtProxSrvPrim	type	Signal identity – always set to CSR_BT_PROX_SRV_UPDATE_TX_POWER_CFM
CsrBtResultCode	resultCode	Result code returned by the supplier in resultSupplier
CsrBtSupplier	resultSupplier	The result supplier – CSR_BT_SUPPLIER_PROX_SRV is success

Table 7: Members in a CSR_BT_PROX_SRV_UPDATE_TX_POWER_CFM primitive

3.5 Update Battery Level

In order for the server to have the latest information about the battery level and state, the application needs to send down this information whenever it thinks it needs updating. The interval for updating this is not set to anything specific as it is a task of the application to decide on this matter.

If no battery value is sent to the proximity server task, the default value of 0 is provided to peer devices requesting this value.

To send the `CSR_BT_PROX_SRV_UPDATE_BATT_LEVEL_REQ` primitive, the `CsrBtProxSrvUpdateBattLevelReqSend()` function is used. The function takes the arguments described in Table 8.

Type	Argument	Description
CsrUint16	battLevel	The new battery level value

Type	Argument	Description
CsrBtProxSrvBatteryMask	battMask	Bitmask identifying the state of the battery. Bit 0-1: battery present Bit 2-3: battery discharging Bit 4-5: battery charging Bit 6-7: battery state critical Possible values are defined in csr_bt_prox_srv_prim.h
CsrUint8	serviceRequired	Service required field values as defined in the file csr_bt_prox_srv_prim.h.

Table 8: Arguments for `CsrBtProxSrvUpdateBattLevelReqSend` function

When the Update Battery Level request has been processed, a `CSR_BT_PROX_SRV_UPDATE_BATT_LEVEL_CFM` primitive will be sent back to the application. The primitive members are described in Table 9.

Type	Member	Description
CsrBtProxSrvPrim	type	Signal identity – always set to CSR_BT_PROX_SRV_UPDATE_BATT_LEVEL_CFM
CsrBtResultCode	resultCode	Result code returned by the supplier in resultSupplier
CsrBtSupplier	resultSupplier	The result supplier – CSR_BT_SUPPLIER_PROX_SRV is success

Table 9: Members in a CSR_BT_PROX_SRV_UPDATE_BATT_LEVEL_CFM primitive

3.6 Various Event Indications

When the application is activated, it is possible to specify a bitmask telling which events it want to subscribe to. Whenever one of the events are triggered in the server, then the it will send off the subscribed event to the application.

The write event is triggered when the peer device tries to write to one of the writable handles in the database where IRQ is enabled (Tx Power Client Configuration, Immediate Alert Level, Link Loss Alert Level).

The Tx Power Changed Event is triggered when the peer device has subscribed to the TX Power Notifications (only relevant for BR/EDR connections) by writing to the Client Characteristic Configuration for the Tx Power Service. At this point a timer will update the Tx Power value by requesting it from the Connection Manager. If the value has changed since last interval, then the Tx Power Changed Event is sent to the application.

The alarm event is triggered if one of the following things happens:

- Link Loss Alert Level >0 and link is disconnected
- Immediate Alert level is updated to a value >0

The parameters for the `CSR_BT_PROX_SRV_WRITE_EVENT_IND` primitive are described in Table 10.

Type	Argument	Description
CsrBtProxSrvPrim	type	Signal identity – always set to CSR_BT_PROX_SRV_WRITE_EVENT_IND
CsrUint16	valueHandle	Handle number in the DB that the peer device is trying to write “value” to.
CsrUint16	valueSize	Size of the written data
CsrUint8	*value	The actual written data

Table 10: Members in a CSR_BT_PROX_SRV_WRITE_EVENT_IND primitive

The parameters for the `CSR_BT_PROX_SRV_TX_POWER_CHANGED_EVENT_IND` primitive are described in Table 11.

Type	Argument	Description
CsrBtProxSrvPrim	type	Signal identity – always set to CSR_BT_PROX_SRV_ TX_POWER_CHANGED _EVENT_IND
CsrUint16	txPower	

Table 11: Members in a CSR_BT_PROX_SRV_TX_POWER_CHANGED_EVENT_IND primitive

The parameters for the CSR_BT_PROX_SRV_ALERT_EVENT_IND primitive are described in Table 12.

Type	Argument	Description
CsrBtProxSrvPrim	type	Signal identity – always set to CSR_BT_PROX_SRV_ALERT _EVENT_IND
CsrBtProxSrvAlertType	alertType	
CsrBtProxSrvAlertLevel	alertLevel	

Table 12: Members in a CSR_BT_PROX_SRV_ALERT_EVENT_IND primitive

.

Terms and Definitions

BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
BLE	Bluetooth Low Energy
LE	Common name for the Low Energy Bluetooth® radio.
BR/EDR	Basic Rate / Enhanced Data Rate. Common name for the standard Bluetooth® radio technology.
CSR	Cambridge Silicon Radio
DUN	Dial Up Networking
FCS	Frame Check Sequence, 16bit CRC used in L2CAP
FTC	File Transfer Client
FTP	File Transfer Protocol
FTS	File Transfer Server
GOEP	Generic Object Exchange Protocol
HCI	Host Controller Interface
L2CAP	Logical Link Control and Adaption Protocol
MSC	Message Sequence Chart
OBEX	Object Exchange Protocol
UniFi™	Group term for CSR's range of chips designed to meet IEEE 802.11 standards

Document History

Revision	Date	History
1	26 SEP 11	Ready for release 18.2.0
2	23 JAN 2012	Ready for release 18.2.2

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