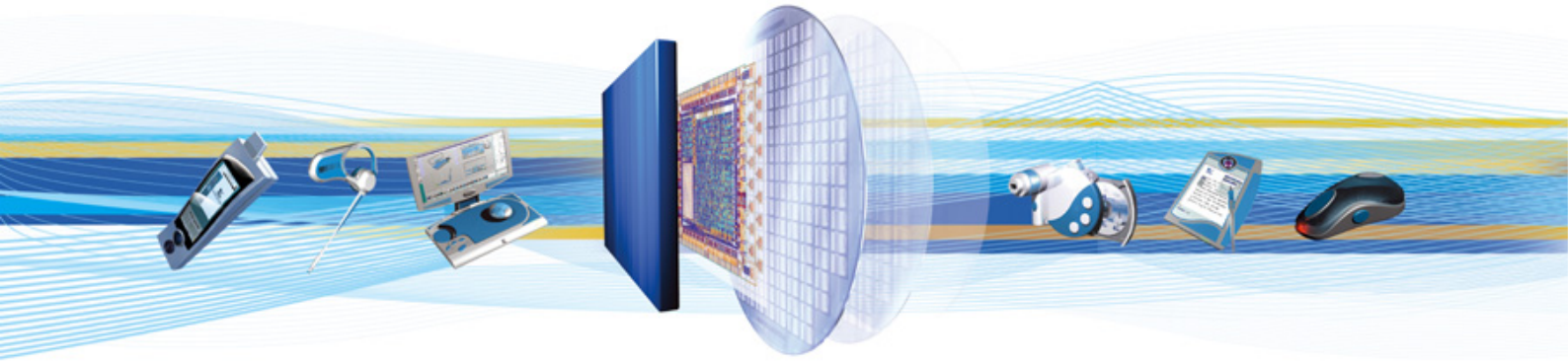
# CSR Synergy Bluetooth 18.2.0

# OBEX Synchronization Client

# 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 message interface provided by the OBEX Synchronization Client (SYNCC). The SYNCC conforms to the client side of the Synchronization Profile, ref. [SYNCC].

## 1.2 Assumptions

The following assumptions and preconditions are made in the following:

- There is a secure and reliable transport between the profile part, i.e. SYNCC and the application
- The SYNCC shall only handle one request at the time
- Bonding (pairing) is NOT handled by the SYNCC

# 2 Description

## 2.1 Introduction

The scenarios covered by this profile are the following:

- Usage of a Bluetooth® device e.g. a car-kit to synchronization of an object store (calendar/phonebook/note/messages) with that of another Bluetooth® device e.g. a mobile phone.
- A second usage is a Bluetooth® device to manipulate objects (calendar/phonebook/note/message entries) on another Bluetooth® device. This includes deleting objects.

The SYNCC provides the following services to the application:

- screening of devices
- connection handling
- OBEX protocol handling

The application is responsible for handling the requests and confirms from the SYNCC with correct data (object) as described in the IrOBEX specification. The SYNCC is not checking if the data is packed correctly with white spaces in the right places, for details see ref. [SYNC] and [OBEX].

## 2.2 Reference Model

The SYNCC interfaces to the Connection Manager (CM).

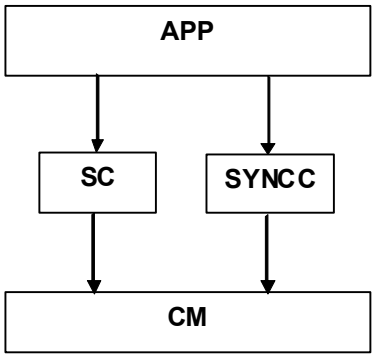



**Figure 1: Reference model**

## 2.3 Sequence Overview

If, in IDLE state, a connect request is received from the application, the SYNCC starts to connect to the specified device and the CONNECT state is entered, then the application receives a confirmation on the connect, the application can issue a request (get or put) to start the transfer for the object. The application can perform multiple gets/puts (one at the time) before disconnecting the connection. When the application disconnects the service, the SYNCC re-enters IDLE state.

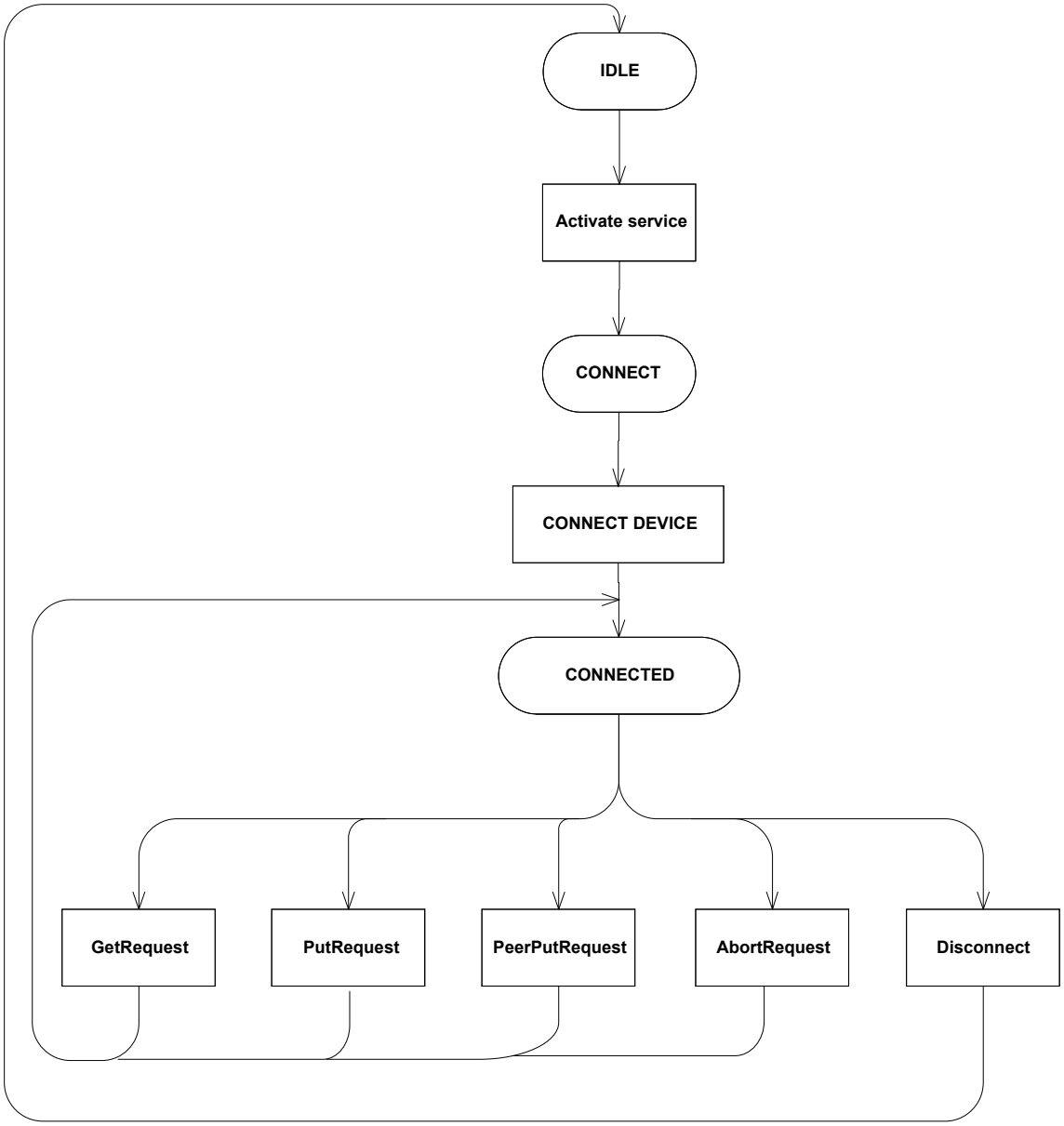



**Figure 2: SYNCC state diagram**

# 3 Interface Description

## 3.1 Connect

When the application wants to connect to a Synchronization Server it has to send a CSR_BT_SYNCC_CONNECT_REQ to the SYNCC. In this message the application has to specify which device to connect to. This message has a parameter called maxPacketSize, which indicates the maximum Obex packet size, which the application wants to receive from the SYNC server side. The value can be between 255 bytes to 64Kbytes – 1, see definition in ref. [OBEX]. If the packet size is large it is optimizing for quick object transfer, but the disadvantage will be use of corresponding larger memory blocks.

The SYNCC sends a CSR_BT_SYNCC_CONNECT_CFM message to the application, which has the status of the connection establishment - this is the parameter result code. For success in the request the code is CSR_BT_OBEX_SUCCESS_RESPONSE_CODE, any other response code indicates a failure in the connection.

Once the Obex connection is established, the SYNCC will, transparently for the application layer, make use of low power modes. This implies use of sniff if supported by the Synchronization Server side. Low power modes are enabled using a supervision timer. If no data is received within the specified time interval, the SYNCC manager will attempt a change to low power mode if possible. The value of the timer is determined by the CSR_BT_SYNCC_LP_SUPERVISION_TIMEOUT and is defined in the csr_bt_usr_config_default.h file.

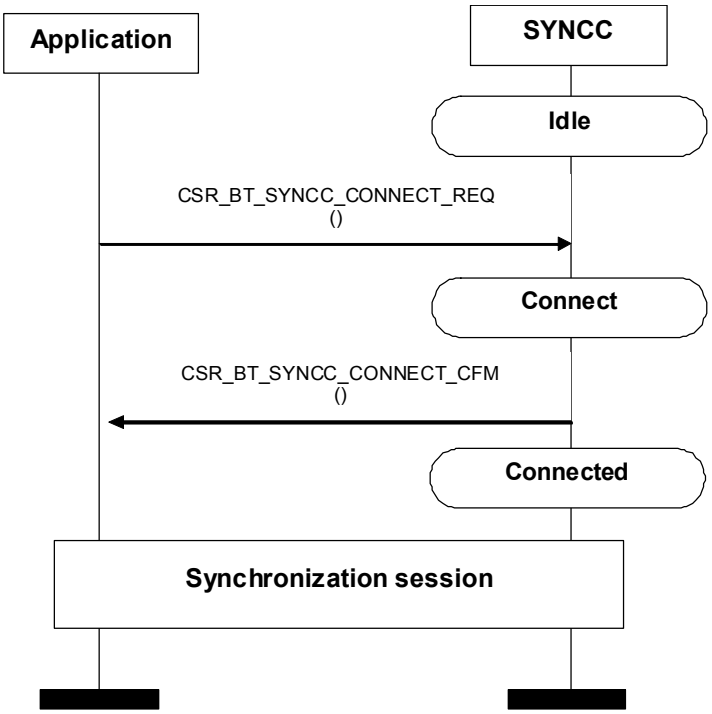



**Figure 3: Connection handling**

## 3.2 Cancel Connect

The application can cancel an outgoing connection request by sending a CSR_BT_SYNCC_CANCEL_CONNECT_REQ. If the outgoing connection can be cancelled the responses will be a CSR_BT_SYNCC_CONNECT_CFM with a response code different from CSR_BT_OBEX_SUCCESS_RESPONSE_CODE.

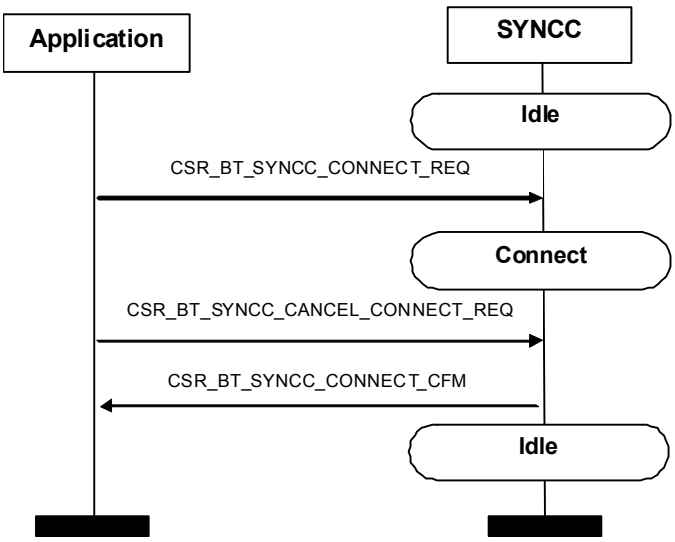



**Figure 4: Cancel Connect I**

Please note that if the application requests a CSR_BT_SYNCC_CANCEL_CONNECT_REQ while the SYNCC is sending a CSR_BT_SYNCC_CONNECT_CFM with the response code CSR_BT_OBEX_SUCCESS_RESPONSE_CODE to the application, then SYNCC will consider the CSR_BT_SYNCC_CANCEL_CONNECT_REQ as a CSR_BT_SYNCC_DISCONNECT_REQ and the application will receive a CSR_BT_SYNCC_DISCONNECT_IND.

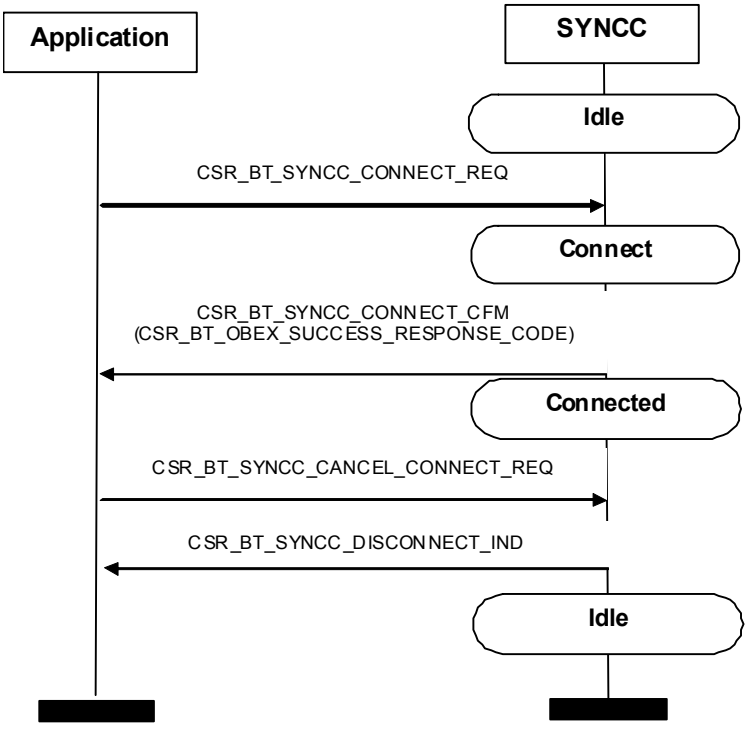



**Figure 5: Cancel Connect II**

## 3.3 Object Transfer

Objects are transmitted to the server by issuing a CSR_BT_SYNCC_ADD_OBJECT_REQ followed by a CSR_BT_SYNCC_ADD_OBJECT_RES or by issuing a CSR_BT_SYNCC_MODIFY_REQ followed by a CSR_BT_SYNCC_MODIFY_OBJECT_RES. The server side responds with the result of the operation in respectively a CSR_BT_SYNCC_ADD_OBJECT_CFM or a CSR_BT_SYNCC_MODIFY_OBJECT_CFM signal. In case the application wants to fragment the body due to memory considerations it can do so by sending a CSR_BT_SYNCC_ADD_OBJECT_RES/CSR_BT_SYNCC_MODIFY_OBJECT_RES with finalFlag set to FALSE. On the indication it can continue to send the next fragment. It can continue until it sets the finalFlag to TRUE.

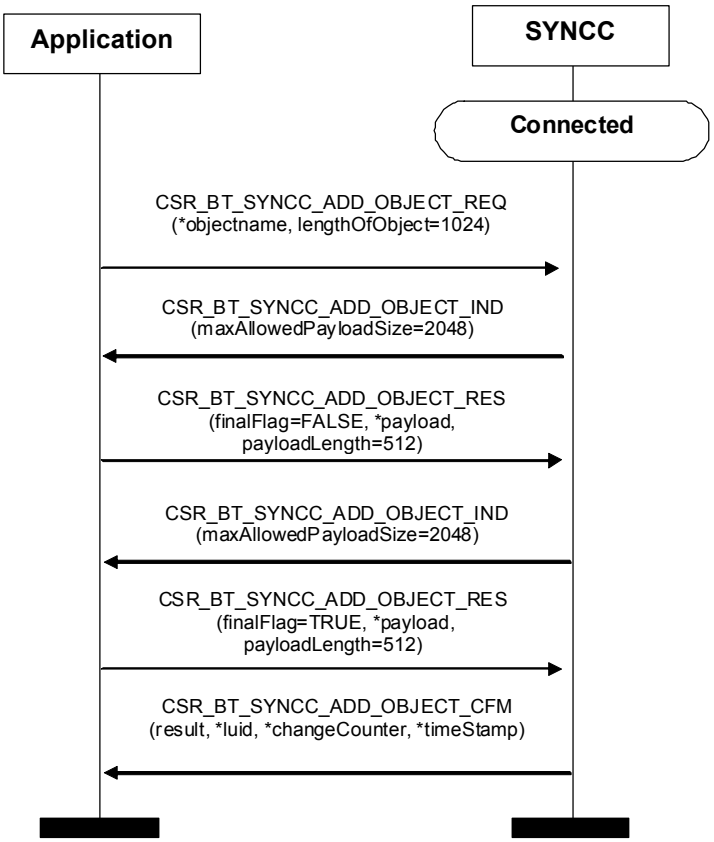



**Figure 6: Add object**

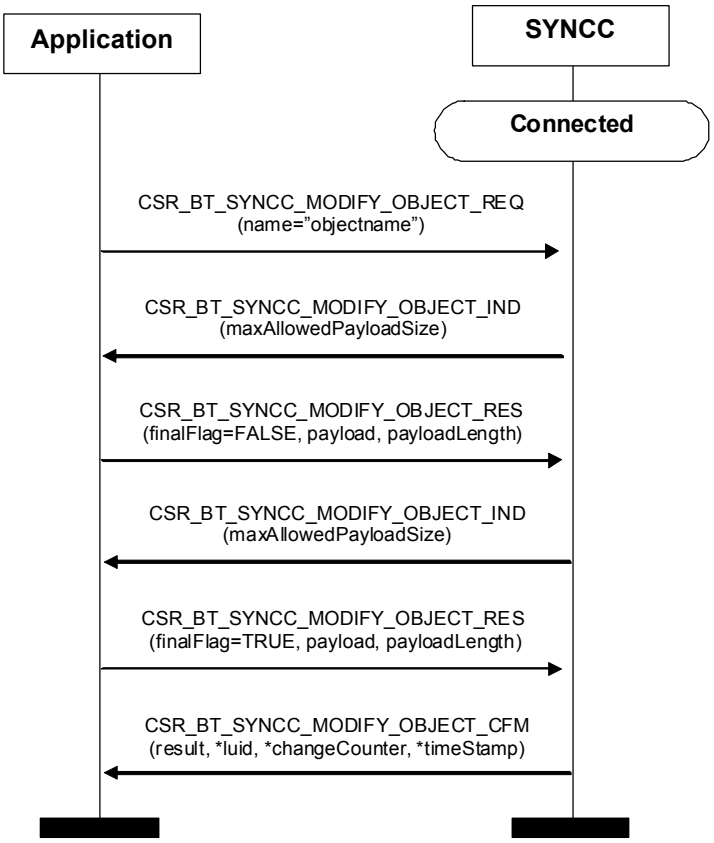



**Figure 7: Modify object**

Objects can be pulled from the server by issuing a CSR_BT_SYNCC_GET_OBJECT_REQ. The server responds with the result of the operation in a CSR_BT_SYNCC_GET_OBJECT_CFM signal. If the server responds with multiple fragments, the first will be received by the application as a CSR_BT_SYNCC_GET_OBJECT_IND and the application has to send a CSR_BT_SYNCC_GET_OBJECT_RES to get the next fragment. This is the pattern until the confirm is received.

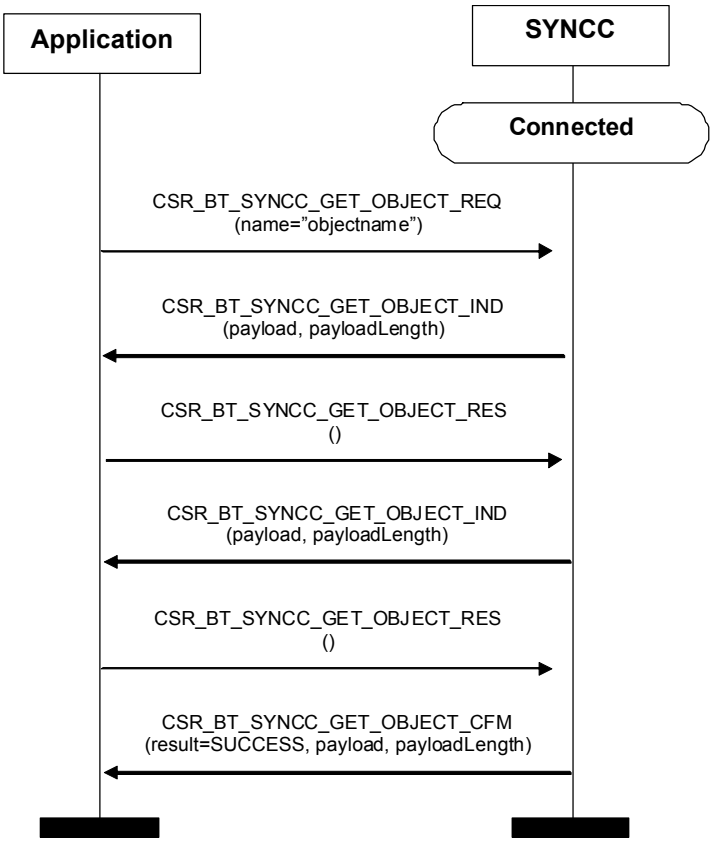



**Figure 8: Get object**

## 3.4 Objects Deletion

Deleting an object is done with the CSR_BT_SYNCC_DELETE_OBJECT_REQ message. The application can specify whether the delete operation must be a hard of soft delete. The result of the operation is carried in the confirm.

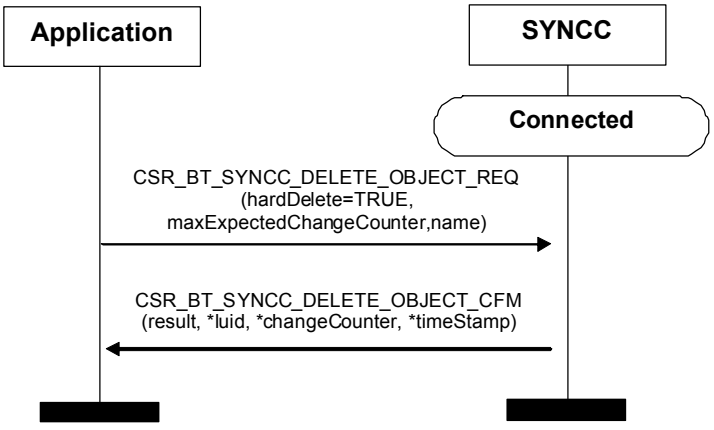



**Figure 9: Delete object**

## 3.5 Abort Operation

The abort request (CSR_BT_SYNCC_ABORT_REQ) is used when the application decides to terminate a multi-packet operation (such as PUT or GET) before it ends. The response (CSR_BT_SYNCC_ABORT_CFM) is received indicating that the abort request is a success. It is also indicating that the abort request is received and the SYNC server is now resynchronized with the client. If anything else is returned, the SYNCC will disconnect the link and send CSR_BT_SYNCC_DISCONNECT_IND to the application.

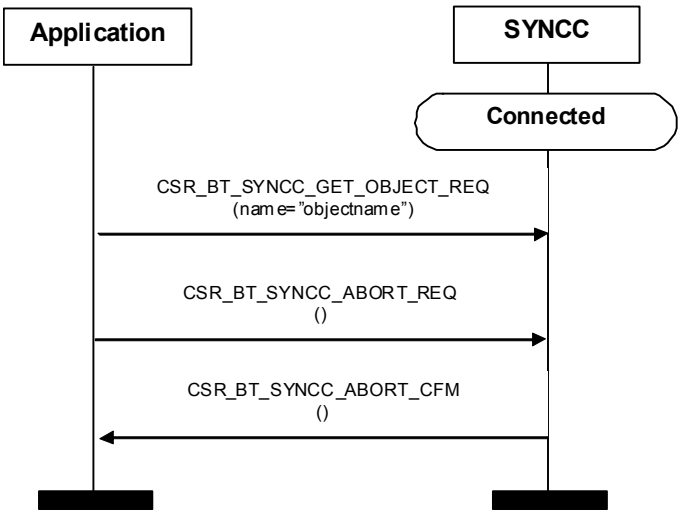



**Figure 10: Abort operation**

## 3.6 Obex Authentication

The application can authenticate the SYNC server by setting the authorize and password parameters in the CSR_BT_SYNCC_CONNECT_REQ. The SYNCC will then authenticate the SYNC server under the obex connection establishment. A SYNC server can authenticate the SYNCC on every operation individually. For each application request it sends it can receive a CSR_BT_SYNCC_AUTHENTICATE_IND instead of the corresponding confirm message. If the application receives a CSR_BT_SYNCC_AUTHENTICATE_IND it must response with a CSR_BT_SYNCC_AUTHENTICATE_RES signal using the password or pin number that the SYNC server requires. An example of the authenticate sequence is illustrated below.

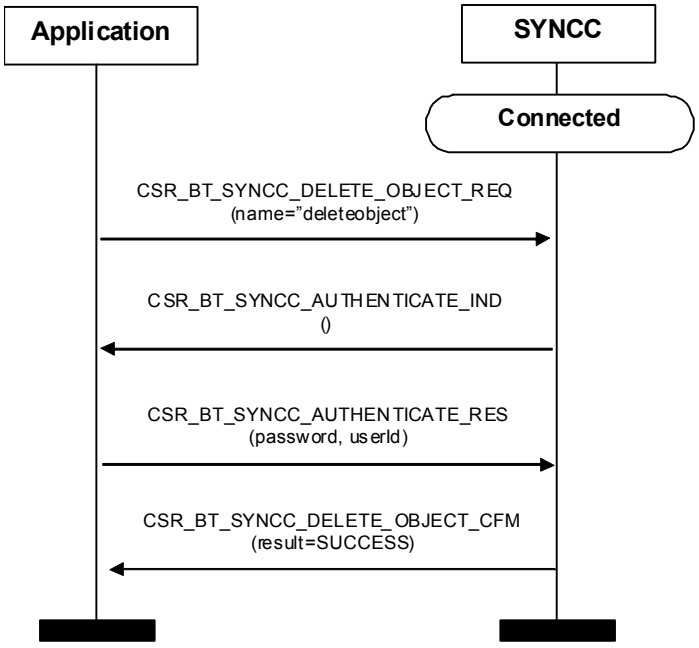



**Figure 11: Authenticate delete object**

## 3.7 Sync Command

The application has to send a CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND_REQ to allow peer SYNC servers to connect in and issue the Sync Command. When the received in the profile it will enter page scan mode to allow peers to connect and issue the Sync Command (which will be received in a CSR_BT_SYNCC_SYNC_COMMAND_IND). The application has to send a CSR_BT_SYNCC_SYNC_COMMAND_RES with its response to the SYNC server In case the finalFlag is not set in the indication it means that the whole object wasn't included in this indication and the application can then retrieve the next part by sending a response=CSR_BT_OBEX_CONTINUE_RESPONSE_CODE. If the finalFlag is set the application should response with a CSR_BT_OBEX_SUCCESS_RESPONSE_CODE. When a SYNC server has disconnected again the profile will automatically prepare itself to receive new incoming connections from the same or other SYNC servers. If the application wants the profile to exit page scan and drop a connection to SYNC server it can do so by sending a CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_REQ. The response of this operation will be carried in a CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_CFM. An example of an activation and a following incoming Sync Command is illustrated below.

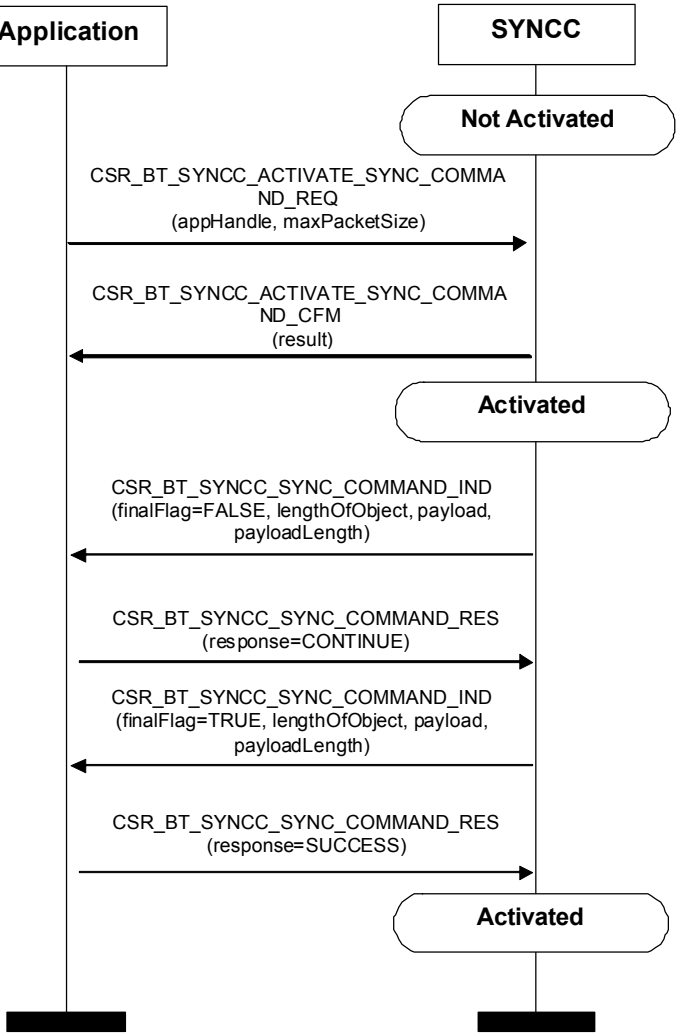



**Figure 12: Activation and reception of Sync Command**

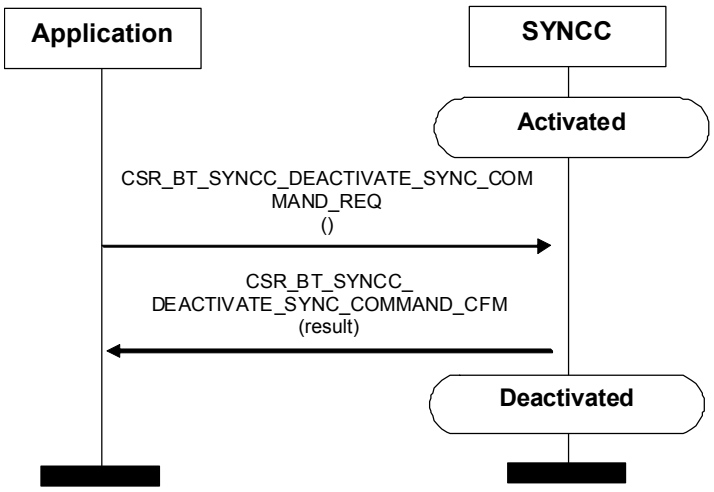



**Figure 13: Deactivation of Sync Command**

## 3.8 Disconnect

Sending a CSR_BT_SYNCC_DISCONNECT_REQ to the SYNCC disconnects the current connection (if any). The disconnect may take some time and is confirmed with a CSR_BT_SYNCC_DISCONNECT_IND signal.

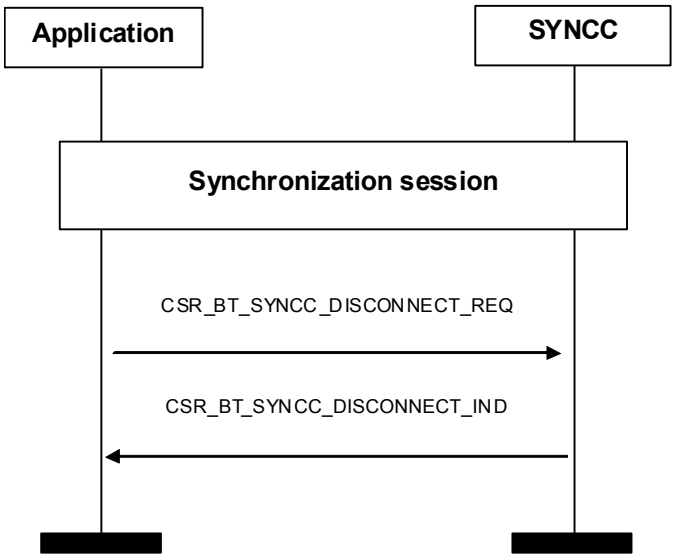



**Figure 14: Normal disconnect**

In case the peer side prematurely disconnects, the SYNCC sends a CSR_BT_SYNCC_DISCONNECT_IND to the application and enters IDLE state.

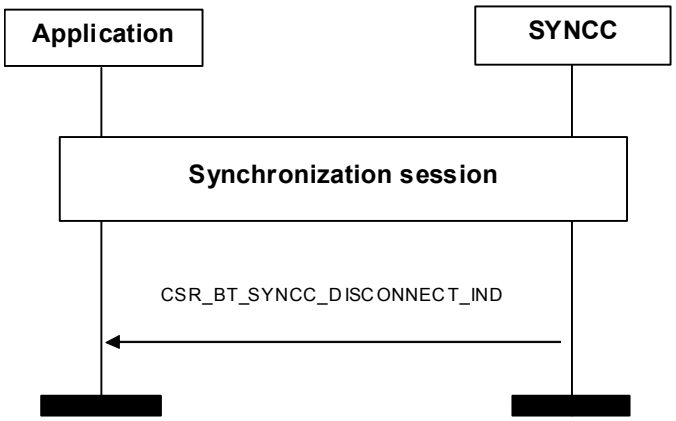



**Figure 15: Abnormal disconnect**

## 3.9 Payload Encapsulated Data

### 3.9.1 Using Offsets

As many OBEX messages contain multiple parameters with variable length, some of the parameters are based on *offsets* instead of standard pointers to the data. Signals with offset-based data can easily be recognized as they have both a *payload* and a *payloadLength* parameter. The *payload* contains the actual data, on which the offset is based. For example, a typical signal may contain the following:

```
CsrCommonPrim  type;
CsrUint8       result;
CsrUint16  ucs2nameOffset;
CsrUint16  bodyOffset;
CsrUint16  bodyLength;
CsrUint16  payloadLength;
CsrUint8       *payload;
```

In this example, two offset parameters can be found, namely *ucs2nameOffset* and *bodyOffset*. To obtain the actual data, the offset value is added to the *payload* pointer, which yields a pointer to the data, i.e.:

```
CsrUint8  *ucs2name;
ucs2name = (CsrUint8*)(primitive->payload + primitive->ucs2nameOffset);
```

As can be seen, the offset contains the number of bytes within the *payload* where the information begins. Similarly, the body data can be retrieved using the following:

```
CsrUint8 *body;
body = (CsrUint8*)(primitive->payload + primitive->bodyOffset);
```

And to illustrate the usage of the *length* parameter, which is also a common parameter, to copy the body one would typically use:

```
CsrMemCpy( copyOfBody, body, primitive->bodyLength );
```

Offset parameters will always have an “Offset” suffix on the name, and offsets are *always* relative to the “payload” parameter.

If the `bodyOffset` or the `bodyLength` is 0 (zero) it means that the signal does not contain any body. The same holds when the `payloadLength` is 0 (zero), which means that there is not payload.

### 3.9.2 Payload Memory

When the application receives a signal which has a *payload* parameter, the application must always free the payload pointer to avoid memory leaks, for example

```
CsrPfree(primitive->payload);
CsrPfree(primitive);
```

will free both the payload data and the message itself. Note that when the payload has been freed, offsets can not be used anymore, as the actual data is contained within the payload.

Signals that do not use the *payload* parameter must still have each of their pointer-based parameters freed.

# 4 OBEX Synchronization Client Primitives

This section gives an overview of the primitives and parameters in the interface. Detailed information can be found in the corresponding csr_bt_syncc_prim.h file.

## 4.1 List of All Primitives

| Primitives: | Reference: |
|---|---|
| CSR_BT_SYNCC_CONNECT_REQ | See section 4.2 |
| CSR_BT_SYNCC_CONNECT_CFM | See section 4.2 |
| CSR_BT_SYNCC_CANCEL_CONNECT_REQ | See section 4.3 |
| CSR_BT_SYNCC_DISCONNECT_REQ | See section 4.4 |
| CSR_BT_SYNCC_DISCONNECT_IND | See section 4.4 |
| CSR_BT_SYNCC_AUTHENTICATE_IND | See section 4.5 |
| CSR_BT_SYNCC_AUTHENTICATE_RES | See section 4.5 |
| CSR_BT_SYNCC_GET_OBJECT_REQ | See section 4.6 |
| CSR_BT_SYNCC_GET_OBJECT_IND | See section 4.6 |
| CSR_BT_SYNCC_GET_OBJECT_RES | See section 4.6 |
| CSR_BT_SYNCC_GET_OBJECT_CFM | See section 4.6 |
| CSR_BT_SYNCC_ADD_OBJECT_REQ | See section 4.7 |
| CSR_BT_SYNCC_ADD_OBJECT_IND | See section 4.7 |
| CSR_BT_SYNCC_ADD_OBJECT_RES | See section 4.7 |
| CSR_BT_SYNCC_ADD_OBJECT_CFM | See section 4.7 |
| CSR_BT_SYNCC_MODIFY_OBJECT_REQ | See section 4.8 |
| CSR_BT_SYNCC_MODIFY_OBJECT_IND | See section 4.8 |
| CSR_BT_SYNCC_MODIFY_OBJECT_RES | See section 4.8 |
| CSR_BT_SYNCC_MODIFY_OBJECT_CFM | See section 4.8 |
| CSR_BT_SYNCC_DELETE_OBJECT_REQ | See section 4.9 |
| CSR_BT_SYNCC_DELETE_OBJECT_CFM | See section 4.9 |
| CSR_BT_SYNCC_ABORT_REQ | See section 4.10 |
| CSR_BT_SYNCC_ABORT_CFM | See section 4.10 |
| CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND_REQ | See section 4.11 |
| CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND_CFM | See section 4.11 |
| CSR_BT_SYNCC_SYNC_COMMAND_IND | See section 4.12 |
| CSR_BT_SYNCC_SYNC_COMMAND_RES | See section 4.12 |
| CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_REQ | See section 4.13 |
| CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_CFM | See section 4.13 |
| CSR_BT_SYNCC_SYNC_COMMAND_ABORT_IND | See section 4.14 |
| CSR_BT_SYNCC_SECURITY_OUT_REQ | See section 4.15 |
| CSR_BT_SYNCC_SECURITY_OUT_CFM | See section 4.15 |
| CSR_BT_SYNCC_SECURITY_IN_REQ | See section 4.16 |
| CSR_BT_SYNCC_SECURITY_IN_CFM | See section 4.16 |

**Table 1: List of all primitives**

## 4.2 CSR_BT_SYNCC_CONNECT

| Primitives \ Parameters | type | appHandle | maxPacketSize | deviceAddr | authorize | supportedStores | obexPeerMaxPacketSize | resultCode | resultSupplier | realmlength | *realm | passwordLength | *password | *userId | length | count | btConnId | windowSize | srmEnable |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CSR_BT_SYNCC_CONNECT_REQ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ |
| CSR_BT_SYNCC_CONNECT_CFM | ✓ | | | | | ✓ | ✓ | ✓ | ✓ | | | | | | | | ✓ | | |

**Table 2: CSR_BT_SYNCC_CONNECT Primitives**

**Description**

To start an OBEX synchronization session against the SYNC server, the application sends a CSR_BT_SYNCC_CONNECT_REQ with the wanted max packet size allowed to send from the server. The server responds with a CSR_BT_SYNCC_CONNECT_CFM. In case the result in the confirmation is CSR_BT_OBEX_SUCCESS_RESULT_CODE the connection is established. Any other value indicates a failure in the connection.

The connect messages between the OBEX Synchronization client and Server is guarded by a timer, thus if for some reason the server do not reply to the OBEX connect request within a fixed time interval the Bluetooth connection is disconnected direct. The timeout functionality is per default set to five seconds, or twenty seconds if the client request OBEX authentication. The timeout value can be disabled, or change by changing CSR_BT_OBEX_CONNECT_TIMEOUT or CSR_BT_OBEX_CONNECT_WITH_AUTH_TIMEOUT, which is define in csr_bt_user_config.default.h. Note if the value of CSR_BT_OBEX_CONNECT_TIMEOUT or CSR_BT_OBEX_CONNECT_WITH_AUTH_TIMEOUT is change, it will influence all OBEX profiles.

The function:

```
CsrBtSynccConnectReqSend (CsrSchedQid          appHandle,
                          CsrUint16            maxPacketSize,
                          CsrBtDeviceAddr      deviceAddr,
                          CsrBool              authorize,
                          CsrUint16            realmLength,
                          CsrUint8             *realm,
                          CsrUint16            passwordLength,
                          CsrUint8             *password,
                          CsrCharString        *userId,
                          CsrUint32            length,
                          CsrUint32            count,
                          CsrUint16            windowSize,
                          CsrBool              srmEnable);
```

defined in csr_bt_syncc_lib.h, builds and sends the CSR_BT_SYNCC_CONNECT_REQ primitive to the SYNCC profile..

**Parameters**

| | |
|---|---|
| type | Signal identity, CSR_BT_SYNCC_CONNECT_REQ/CFM. |
| appHandle | The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle. |

| | |
|---|---|
| maxPacketSize | The maximum obex packet size allowed sending to the client application. |
| deviceAddr | The Bluetooth address of the device that has initiated the OBEX authentication procedure |
| authorize | Is to control the OBEX authentication on connection to a SYNC server. If TRUE the SYNCC will initiate the authentication against the server. |
| supportedStores | Bit pattern of supported stores in the server |
| obexPeerMaxPacketSize | Indicates the maximum size OBEX packet that is allowed to send to the server |
| resultCode | The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors. |
| resultSupplier | This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h |
| realmLength | Number of bytes in realm of type CsrUint16 |
| *realm | A displayable string indicating for the user which userid and/or password to use. The first byte of the string is the character set of the string. The table below shows the different values for character set. |

| Char set Code | Meaning |
|---|---|
| 0 | ASCII |
| 1 | ISO-8859-1 |
| 2 | ISO-8859-2 |
| 3 | ISO-8859-3 |
| 4 | ISO-8859-4 |
| 5 | ISO-8859-5 |
| 6 | ISO-8859-6 |
| 7 | ISO-8859-7 |
| 8 | ISO-8859-8 |
| 9 | ISO-8859-9 |
| 0xFF = 255 | UNICODE |

| | |
|---|---|
| passwordLength | The length of the response password. |
| *password | Pointer to data containing the challenge password of the OBEX authentication. |
| *userId | Pointer to a Zero terminated string (ASCII) containing the userId for the authentication. |
| length | Length is use to express the approximate total length of the bodies of all the objects in the transaction. If set to 0 this header will not be include. |

| | |
|---|---|
| count | Count is use to indicate the number of objects that will be sent during this connection. If set to 0 this header will not be include |
| btConnId | Identifier used when moving the connection to another AMP controller, i.e. when calling the `CsrBtAmpmMoveReqSend`-function. |
| windowSize | Controls how many packets the OBEX profile (and lower protocol layers) are allowed to cache on the data receive side. A value of zero (0) will cause the system to auto-detect this value. |
| srmEnable | Enable local support for Single Response Mode. |

## 4.3 CSR_BT_SYNCC_CANCEL_CONNECT

| Primitives \ Parameters | type |
|---|---|
| CSR_BT_SYNCC_CANCEL_CONNECT_REQ | ✓ |

**Table 3: CSR_BT_SYNCC_CANCEL_CONNECT Primitives**

**Description**

The CSR_BT_SYNCC_CANCEL_CONNECT_REQ can be used the cancel an ongoing connect procedure. If the SYNCC succeeds in cancelling the ongoing connection attempt the application will receive a CSR_BT_SYNCC_CONNECT_CFM with a response code different from CSR_BT_OBEX_SUCCESS_RESPONSE_CODE.

**Parameters**

type Signal identity, CSR_BT_SYNCC_CANCEL_CONNECT_REQ

## 4.4 CSR_BT_SYNCC_DISCONNECT

| Parameters / Primitives | type | normalDisconnect | reasonCode | reasonSupplier |
|---|---|---|---|---|
| CSR_BT_SYNCC_DISCONNECT_REQ | ✓ | ✓ | | |
| CSR_BT_SYNCC_DISCONNECT_IND | ✓ | | ✓ | ✓ |

**Table 4: CSR_BT_SYNCC_DISCONNECT Primitives**

**Description**

To disconnect a connection to a server (if any) send a CSR_BT_SYNCC_DISCONNECT_REQ to the SYNCC. When disconnected, the SYNCC will respond with a CSR_BT_SYNCC_DISCONNECT_IND. If the link is dropped in the middle of a session, the apps will receive a CSR_BT_SYNCC_DISCONNECT_IND indicating that the OBEX syncronization session is finished, and is ready for a new one.

The disconnect messages between the OBEX Synchronization client and Server is guarded by a timer, thus if for some reason the server do not reply to the OBEX disconnect request within a fixed time interval the Bluetooth connection is disconnected direct. The timeout functionality is per default set to five seconds. The timeout value can be disable, or change by changing CSR_BT_OBEX_DISCONNECT_TIMEOUT, which is define in csr_bt_user_config.default.h. Note if the value of CSR_BT_OBEX_DISCONNECT_TIMEOUT is change, it will influence all OBEX profiles.

**Parameters**

| | |
|---|---|
| type | Signal identity, CSR_BT_SYNCC_DISCONNECT_REQ/IND. |
| normalDisconnect | FALSE defines an Abnormal disconnect sequence where the Bluetooth connection is released directly. TRUE defines a normal disconnect sequence where the OBEX connection is released before the Bluetooth connection. |
| reasonCode | The reason code of the operation. Possible values depend on the value of reasonSupplier. If e.g. the reasonSupplier == CSR_BT_SUPPLIER_CM then the possible reason codes can be found in csr_bt_cm_prim.h. If the reasonSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors. |
| reasonSupplier | This parameter specifies the supplier of the reason given in reasonCode. Possible values can be found in csr_bt_result.h |

## 4.5 CSR_BT_SYNCC_AUTHENTICATE

| Parameters / Primitives | type | deviceAddr | options | realmLength | * realm | *password | passwordLength | *userId |
|---|---|---|---|---|---|---|---|---|
| CSR_BT_SYNCC_AUTHENTICATE_IND | ✓ | ✓ | ✓ | ✓ | ✓ | | | |
| CSR_BT_SYNCC_AUTHENTICATE_RES | ✓ | | | | | ✓ | ✓ | ✓ |

**Table 5: CSR_BT_SYNCC_AUTHENTICATE Primitives**

**Description**

The indication and response signal is used when the SYNC server wants to OBEX authenticate the SYNC client. The application has to response with the password or pin number in the password and userId for the server to identify the proper password.

**Parameters**

type — Signal identity, CSR_BT_SYNCC_AUTHENTICATE_IND/RES.

deviceAddr — The Bluetooth address of the device that has initiated the OBEX authentication procedure

options — Challenge information of type CsrUint8.
If bit 0 is set it means that the application must response with a user Id in a CSR_BT_SYNCS_AUTHENTICATE_RES message. If bit 0 is not set the application can just set the userId to NULL.

Bit 1 indicates the access mode being offered by the sender
If bit 1 is set the access mode is read only. If bit 1 is not set the sender gives full access, e.g. both read and write.

Bit 2 - 7 is reserved.

realmLength — Number of bytes in realm of type CsrUint16

* realm — A displayable string indicating for the user which userid and/or password to use. The first byte of the string is the character set of the string. The table below shows the different values for character set.

Note that this pointer must be CsrPfree by the application, and that this pointer can be NULL because the realm field is optional to set by the peer device.

| Char set Code | Meaning |
|---|---|
| 0 | ASCII |
| 1 | ISO-8859-1 |
| 2 | ISO-8859-2 |
| 3 | ISO-8859-3 |

| | |
|---|---|
| 4 | ISO-8859-4 |
| 5 | ISO-8859-5 |
| 6 | ISO-8859-6 |
| 7 | ISO-8859-7 |
| 8 | ISO-8859-8 |
| 9 | ISO-8859-9 |
| 0xFF = 255 | UNICODE |

| | |
|---|---|
| *password | Containing the response password of the OBEX authentication. This is a pointer which shall be allocated by the application. |
| passwordLength | The length of the response password. |
| *userId | Zero terminated string (ASCII) containing the userId for the authentication. This is a pointer which shall be allocated by the application. |

## 4.6 CSR_BT_SYNCC_GET_OBJECT

| Primitives \ Parameters | type | *ucs2name | responseCode | bodyOffset | bodyLength | *payload | payloadLength | smpOn |
|---|---|---|---|---|---|---|---|---|
| CSR_BT_SYNCC_GET_OBJECT_REQ | ✓ | ✓ | | | | | | ✓ |
| CSR_BT_SYNCC_GET_OBJECT_RES | ✓ | | | | | | | ✓ |
| CSR_BT_SYNCC_GET_OBJECT_IND | ✓ | | | ✓ | ✓ | ✓ | ✓ | |
| CSR_BT_SYNCC_GET_OBJECT_CFM | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | |

**Table 6: CSR_BT_SYNCC_GET_OBJECT Primitives**

**Description**

To retrieve an object from the SYNC server the apps sends the CSR_BT_SYNCC_GET_OBJECT_REQ to the SYNCC and the name parameter specifies which object the SYNC client wants. If the response is too large to fit into one packet the client first receives a CSR_BT_SYNCC_GET_OBJECT_IND which it has to response with a CSR_BT_SYNCC_GET_OBJECT_RES and this sequence can be repeated several times until the SYNC server signals that this is the final part of the object, in which case the application will receive the last part of the object in the confirm signal (CSR_BT_SYNCC_GET_OBJECT_CFM). In case the result is different from success (when the result is different from CSR_BT_OBEX_SUCCESS_RESPONSE_CODE), the other parameters are invalid and not used.

**Parameters**

| | |
|---|---|
| type | Signal identity, CSR_BT_SYNCC_GET_OBJECT_REQ/CFM. |
| *ucs2name | A null terminated 16 bit Unicode text string (UCS2) containing the (file) name of the object.<br><br>The function “CsrUtf82Ucs2String” can be used for converting a null terminated UTF8 text string into a null terminated UCS2 text string. |
| responseCode | The response code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors.<br><br>The responseCodes are defined in (csr_bt_obex.h) with the following type CsrBtObexResponseCode and can also be found in IrDA Object Exchange Protocol. |
| bodyOffset | Offset of object relative to payload. |
| bodyLength | The length of the body (object). |
| *payload | OBEX payload data. Offsets are relative to this pointer. |
| payloadLength | Number of bytes in payload. |
| smpOn | Reserved for future use. Set to FALSE. |

## 4.7 CSR_BT_SYNCC_ADD_OBJECT

| Primitives \ Parameters | type | *objectName | lengthOfObject | maxAllowedPayloadSize | finalFlag | *payload | payloadLength | responseCode | *luid | *changeCounter | *timeStamp |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CSR_BT_SYNCC_ADD_OBJECT_REQ | ✓ | ✓ | ✓ | | | | | | | | |
| CSR_BT_SYNCC_ADD_OBJECT_IND | ✓ | | | ✓ | | | | | | | |
| CSR_BT_SYNCC_ADD_OBJECT_RES | ✓ | | | | ✓ | ✓ | ✓ | | | | |
| CSR_BT_SYNCC_ADD_OBJECT_CFM | ✓ | | | | | | | ✓ | ✓ | ✓ | ✓ |

**Table 7: CSR_BT_SYNCC_ADD_OBJECT Primitives**

**Description**

To add an object to the Synchronization Server, send the CSR_BT_SYNCC_ADD_OBJECT_REQ to the SYNCC. The SYNCC then sends an indication with the maximum size of the object to send in the response (CSR_BT_SYNCC_ADD_OBJECT_RES). If the body needs to be sent as multiple fragments, the app can send a new CSR_BT_SYNCC_ADD_OBJECT_RES after receiving a CSR_BT_SYNCC_ADD_OBJECT_IND message. The finalFlag indicates the last part of the body (object). In case the result in the confirm is different from success, the other parameters are invalid and not used and the apps can stop sending more of this object.

**Parameters**

| | |
|---|---|
| type | Signal identities, CSR_BT_SYNCC_ADD_OBJECT_REQ/IND and CSR_BT_SYNCC_ADD_OBJECT_RES/CFM. |
| *objectName | Unicode string containing the full path and name of the object to send |
| lengthOfObject | The total length of the object to send.<br><br>Note that if the total length of the object is known in advance, this parameter should be set, as it allows the receiver to quickly terminate transfers requiring too mush space, and also makes progress reporting easier. In the case that the total length of the object is unknown, this parameter can be set to 0. |
| maxAllowedPayloadSize | The maximum allowed payload size to include in the CsrBtSyncAddObjectRes |
| finalFlag | Indicates that the payload (object) fits the whole/last part of the object. |
| *payload | Payload to send |
| payloadLength | Length of the payload to send |
| responseCode | The response code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as |

| | |
|---|---|
| | errors.<br><br>The responseCodes are defined in (csr_bt_obex.h) with the following type CsrBtObexResponseCode and can also be found in IrDA Object Exchange Protocol. |
| *luid | Luid returned by server if add was level 4 operation |
| *changeCounter | New change counter returned by server if it support change counter sync anchors |
| *timeStamp | New timestamp returned by server if it support timestamps as sync anchors |

## 4.8 CSR_BT_SYNCC_MODIFY_OBJECT

| Primitives \ Parameters | type | *maxExpectedChangeCounter | *objectName | lengthOfObject | maxAllowedPayloadSize | finalFlag | *payload | payloadLength | responseCode | *luid | *changeCounter | *timeStamp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CSR_BT_SYNCC_MODIFY_OBJECT_REQ | ✓ | ✓ | ✓ | ✓ | | | | | | | | |
| CSR_BT_SYNCC_MODIFY_OBJECT_IND | ✓ | | | | ✓ | | | | | | | |
| CSR_BT_SYNCC_MODIFY_OBJECT_RES | ✓ | | | | | ✓ | ✓ | ✓ | | | | |
| CSR_BT_SYNCC_MODIFY_OBJECT_CFM | ✓ | | | | | | | | ✓ | ✓ | ✓ | ✓ |

**Table 8: CSR_BT_SYNCC_MODIFY_OBJECT Primitives**

**Description**

To push a modified object to the Synchronization Server, send the CSR_BT_SYNCC_MODIFY_OBJECT_REQ to the SYNCC. The SYNCC then sends an indication with the maximum size of the object to send in the response (CSR_BT_SYNCC_MODIFY_OBJECT_RES). If the body needs to be sent as multiple fragments, the app can send a new CSR_BT_SYNCC_MODIFY_OBJECT_RES after receiving a CSR_BT_SYNCC_MODIFY_OBJECT_IND message. The finalFlag indicates the last part of the body (object). In case the result in the confirm is different from success, the other parameters are invalid and not used and the apps can stop sending more of this object.

**Parameters**

| | |
|---|---|
| type | Signal identities, CSR_BT_SYNCC_MODIFY_OBJECT_REQ/CFM and CSR_BT_SYNCC_MODIFY_OBJECT_/REQ/CFM. |
| *maxExpectedChangeCounter | Zero-terminated ASCII number string specifying the maximum expected change counter value e.g. "5678", NB: if NULL the change counter value is not sent |
| *objectName | Unicode string containing the full path and name of the object to send |
| lengthOfObject | The total length of the object to send.<br><br>Note that if the total length of the object is known in advance, this parameter should be set, as it allows the receiver to quickly terminate transfers requiring too mush space, and also makes progress reporting easier. In the case that the total length of the object is unknown, this parameter can be set to 0. |
| maxAllowedPayloadSize | The maximum allowed payload size to include in the CsrBtSyncModifyObjectRes |
| finalFlag | Indicates that the body (object) fits the whole object or that it is the last part. |
| *payload | Payload to send |
| payloadLength | Length of the payload to send |
| responseCode | The response code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should |

| | |
|---|---|
| | consider them as errors.<br><br>The responseCodes are defined in (csr_bt_obex.h) with the following type CsrBtObexResponseCode and can also be found in IrDA Object Exchange Protocol. |
| *luid | Luid returned by server if add was level 4 operation |
| *changeCounter | New change counter returned by server if it support change counter sync anchors |
| *timeStamp | New timestamp returned by server if it support timestamps as sync anchors |

## 4.9 CSR_BT_SYNCC_DELETE_OBJECT

| Parameters / Primitives | type | hardDelete | *maxExpectedChangeCounter | *objectName | responseCode | *luid | *changeCounter | *timeStamp |
|---|---|---|---|---|---|---|---|---|
| CSR_BT_SYNCC_DELETE_OBJECT_REQ | ✓ | ✓ | ✓ | ✓ | | | | |
| CSR_BT_SYNCC_DELETE_OBJECT_CFM | ✓ | | | ✓ | ✓ | ✓ | ✓ | ✓ |

**Table 9: CSR_BT_SYNCC_DELETE_OBJECT Primitives**

**Description**

This signal CSR_BT_SYNCC_DELETE_OBJECT_REQ is used for deleting objects on the SYNC server. The result of the delete operation is given to the apps with the confirm signal CSR_BT_SYNCC_DELETE_OBJECT_CFM. The result can contain error codes corresponding to the reason for failure or if the server does not permit this operation from the SYNCC. The apps can also receive a CSR_BT_SYNCC_AUTHENTICATE_IND before getting the confirm, this takes place if the other side (SYNC server) wants to authenticate the client for this operation.

**Parameters**

| | |
|---|---|
| Type | Signal identity, CSR_BT_SYNCC_DELETE_OBJECT_REQ/CFM. |
| HardDelete | If TRUE a hard delete of the object is requested, if FALSE a soft delete is requested |
| *maxExpectedChangeCounter | Zero-terminated ASCII number string specifying the maximum expected change counter value e.g. "5678", NB: if NULL the change counter value is not sent |
| *objectName | Unicode string containing the full path and name of the object to delete |
| responseCode | The response code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors.<br><br>The responseCodes are defined in (csr_bt_obex.h) with the following type CsrBtObexResponseCode and can also be found in IrDA Object Exchange Protocol. |
| *luid | Luid returned by server if delete was level 4 operation |
| *changeCounter | New change counter returned by server if it support change counter sync anchors |
| *timeStamp | New timestamp returned by server if it support timestamps as sync anchors |

## 4.10 CSR_BT_SYNCC_ABORT

| Primitives \ Parameters | type |
|---|---|
| CSR_BT_SYNCC_ABORT_REQ | ✓ |
| CSR_BT_SYNCC_ABORT_CFM | ✓ |

**Table 10: CSR_BT_SYNCC_ABORT Primitives**

**Description**

The CSR_BT_SYNCC_ABORT_REQ is used when the apps decides to terminate a multi-packet operation (such as GET/PUT) before it normally ends. The CSR_BT_OPC_SYNCC_CFM indicates that the SYNC server has received the abort response and the SYNC server is now resynchronized with the client. If the server does not respond the Abort Request or it response with a response code different from CSR_BT_OBEX_SUCCESS_RESPONSE_CODE, the profile will disconnect the Bluetooth connection and send a CSR_BT_DISCONNECT_IND to the application.

**Parameters**

type Signal identity, CSR_BT_SYNCC_ABORT_REQ/CFM.

## 4.11 CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND

| Parameters / Primitives | type | appHandle | maxPacketSize | resultCode | resultSupplier |
|---|---|---|---|---|---|
| CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND_REQ | ✓ | ✓ | ✓ | | |
| CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND_CFM | ✓ | | | ✓ | ✓ |

**Table 11: CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND Primitives**

**Description**

The application has to send a CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND_REQ to allow peer SYNC servers to connect in and issue the Sync Command. When the received in the profile it will make sure that the device enters page scan mode to allow peers to connect and issue the Sync Command. If a server has made a connection and disconnected again the profile will automatically make sure to become connectable again, hence it is only necessary to send this signal once.

**Parameters**

| | |
|---|---|
| Type | Signal identity, CSR_BT_SYNCC_ACTIVATE_SYNC_COMMAND_REQ/CFM. |
| appHandle | The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle. |
| maxPacketSize | The maximum obex packet size allowed sending to the client application. |
| resultCode | The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors. |
| resultSupplier | This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h |

## 4.12 CSR_BT_SYNCC_SYNC_COMMAND

| Parameters / Primitives | type | finalFlag | lengthOfObject | ucs2nameOffset | bodyOffset | bodyLength | *payload | payloadLength | responseCode | smpOn |
|---|---|---|---|---|---|---|---|---|---|---|
| CSR_BT_SYNCC_SYNC_COMMAND_IND | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| CSR_BT_SYNCC_SYNC_COMMAND_RES | ✓ | | | | | | | | ✓ | ✓ |

**Table 12: CSR_BT_SYNCC_SYNC_COMMAND Primitives**

**Description**

This signal CSR_BT_SYNCC_SYNC_COMMAND_IND is received when a peer SYNC server has connected and sent a Sync Command.

**Parameters**

| | |
|---|---|
| type | Signal identity, CSR_BT_SYNCC_SYNC_COMMAND_IND/RES. |
| finalFlag | If TRUE this is the complete obex packet from the peer, if FALSE the rest of the packet can be retrieved by sending a OBEX_CONTINUE_RESPONSE_CODE in CsrBtSynccSyncCommandRes |
| lengthOfObject | Lenght of the object sent from the peer. NB: If this carries the value 0 it means that the peer has not included it in the PUT request |
| ucs2nameOffset | Payload relative offset to where the zero-terminated unicode object name starts. NB: Only valid if != 0 |
| bodyOffset | Payload relative offset to where the body part starts. NB: Only valid if bodyLength |
| bodyLength | Length of the object body carried with this payload |
| *payload | Pointer to the complete OBEX payload received from the server |
| payloadLength | Total length of the payload |
| responseCode | The response code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors.<br><br>The responseCodes are defined in (csr_bt_obex.h) with the following type CsrBtObexResponseCode and can also be found in IrDA Object Exchange Protocol. |
| smpOn | Reserved for future use. Set to FALSE. |

## 4.13 CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND

| Primitives \ Parameters | type | resultCode | resultSupplier |
|---|---|---|---|
| CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_REQ | ✓ | | |
| CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_CFM | ✓ | ✓ | ✓ |

**Table 13: CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND Primitives**

**Description**

If the application wants the profile to exit page scan and drop a connection to SYNC server (if any) it can do so by sending a CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_REQ. The response of this operation will be carried in a CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_CFM.

**Parameters**

| | |
|---|---|
| type | Signal identity,<br>CSR_BT_SYNCC_DEACTIVATE_SYNC_COMMAND_REQ/CFM. |
| resultCode | The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors. |
| resultSupplier | This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h |

## 4.14 CSR_BT_SYNCC_SYNC_COMMAND_ABORT_IND

| Parameters / Primitives | type | descriptionOffset | descriptionLength | *payload | payloadLength |
|---|---|---|---|---|---|
| CSR_BT_SYNCC_SYNC_COMMAND_ABORT_IND | ✓ | ✓ | ✓ | ✓ | ✓ |

**Table 14: CSR_BT_SYNCC_SYNC_COMMAND_ABORT_IND Primitives**

**Description**

This signal CSR_BT_SYNCC_SYNC_COMMAND_ABORT_IND is received in case the SYNC server for any reasons chooses to abort an ongoing Sync Command.

**Parameters**

| | |
|---|---|
| type | Signal identity, CSR_BT_SYNCC_SYNC_COMMAND_ABORT_IND |
| descriptionOffset | Payload relative offset to where the description part starts. NB: Only valid if descriptionLength |
| descriptionLength | Length of the description carried with this payload |
| *payload | Pointer to the complete OBEX payload received from the server |
| payloadLength | Total length of the payload |

## 4.15 CSR_BT_SYNCC_SECURITY_OUT

| Parameters / Primitives | type | appHandle | secLevel | resultCode | resultSupplier |
|---|---|---|---|---|---|
| CSR_BT_SYNCC_SECURITY_OUT_REQ | ✓ | ✓ | ✓ | | |
| CSR_BT_SYNCC_SECURITY_OUT_CFM | ✓ | | | ✓ | ✓ |

**Table 15: CSR_BT_SYNCC_SECURITY_OUT Primitives**

**Description**

Applications that wish to change the enforcement to a specific profile security level, i.e. authentication, encryption and/or authorisation, can use this API to set up the security level for *new* connections. Note that this API is for the local device only and can be used from within any state.

The *CSR_BT_SECURITY_OUT_REQ* signal sets up the security level for new outgoing connections. Already established and pending connections are not altered. Note that *authorisation* should not be used for outgoing connections as that may be confusing for the user – there is really no point in requesting an outgoing connection and afterwards having to authorise as they are both locally-only decided procedures.

Note, that any attempts to set security to a less secure level than the mandatory security level will be rejected. See csr_bt_profiles.h for mandatory security settings. The default settings used by CSR Synergy Bluetooth are set to require authentication and encryption.

Note that if MITM protection is requested and the remote device does not have the required IO capabilities, pairing/bonding will fail and connections to the remote device *cannot* be made. See [SC] for further details.

**Parameters**

type — Signal identity CSR_BT_SYNCC_SECURITY_OUT_REQ/CFM.

appHandle — Application handle to which the confirm message is sent.

secLevel — The application must specify one of the following values:

- CSR_BT_SEC_DEFAULT : Use default security settings
- CSR_BT_SEC_MANDATORY: Use mandatory security settings
- CSR_BT_SEC_SPECIFY: Specify new security settings

If CSR_BT_SEC_SPECIFY is set the following values can be OR'ed additionally:

- CSR_BT_SEC_AUTHORISATION: Require authorisation
- CSR_BT_SEC_ AUTHENTICATION: Require authentication
- CSR_BT_SEC_SEC_ENCRYPTION: Require encryption (implies authentication)
- CSR_BT_SEC_MITM: Require MITM protection (implies encryption)

| | |
|---|---|
| resultCode | The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors. |
| resultSupplier | This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h |

## 4.16 CSR_BT_SYNCC_SECURITY_IN

| Primitives \ Parameters | type | appHandle | secLevel | resultCode | resultSupplier |
|---|---|---|---|---|---|
| CSR_BT_SYNCC_SECURITY_IN_REQ | ✓ | ✓ | ✓ | | |
| CSR_BT_SYNCC_SECURITY_IN_CFM | ✓ | | | ✓ | |

**Table 16: CSR_BT_SYNCC_SECURITY_IN Primitives**

**Description**

Applications that wish to change the enforcement to a specific profile security level, i.e. authentication, encryption and/or authorisation, can use this API to set up the security level for *new* connections. Note that this API is for the local device only and can be used from within any state.

The *CSR_BT_SECURITY_IN_REQ* signal sets up the security level for new outgoing connections. Already established and pending connections are not altered. Note that *authorisation* should not be used for outgoing connections as that may be confusing for the user – there is really no point in requesting an outgoing connection and afterwards having to authorise as they are both locally-only decided procedures.

Note, that any attempts to set security to a less secure level than the mandatory security level will be rejected. See csr_bt_profiles.h for mandatory security settings. The default settings used by CSR Synergy Bluetooth are set to require authentication and encryption.

Note that if MITM protection is requested and the remote device does not have the required IO capabilities, pairing/bonding will fail and connections to the remote device *cannot* be made. See [SC] for further details.

**Parameters**

type — Signal identity CSR_BT_SYNCC_SECURITY_IN_REQ/CFM.

appHandle — Application handle to which the confirm message is sent.

secLevel — The application must specify one of the following values:

- CSR_BT_SEC_DEFAULT : Use default security settings
- CSR_BT_SEC_MANDATORY: Use mandatory security settings
- CSR_BT_SEC_SPECIFY: Specify new security settings

If CSR_BT_SEC_SPECIFY is set the following values can be OR'ed additionally:

- CSR_BT_SEC_AUTHORISATION: Require authorisation
- CSR_BT_SEC_ AUTHENTICATION: Require authentication
- CSR_BT_SEC_SEC_ENCRYPTION: Require encryption (implies authentication)
- CSR_BT_SEC_MITM: Require MITM protection (implies encryption)

| | |
|---|---|
| resultCode | The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. If the resultSupplier == CSR_BT_SUPPLIER_OBEX then the possible result codes can be found in csr_bt_obex.h. All values which are currently not specified in the respective prim.h files or csr_bt_obex.h are regarded as reserved and the application should consider them as errors. |
| resultSupplier | This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h |

# 5 Document References

| Document | Reference |
|---|---|
| IrDA Object Exchange Protocol - IrOBEX<br>Version 1.2<br>18 March 1999 | [OBEX] |
| Specifications for Ir Mobile Communications (IrMC)<br>Version 1.1<br>01 March 1999 | [IRMC] |
| CSR Synergy Bluetooth, SC – Security Controller API Description, Document no. api-0102-sc | [SC] |

# 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 |
| CSR | Cambridge Silicon Radio |
| SYNCC | OBEX Synchronization Client |
| SDS | Service Discovery Server |
| SIG | Special Interest Group |
| SYNCC | Obex Synchronization Client |
| 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 |

# 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.