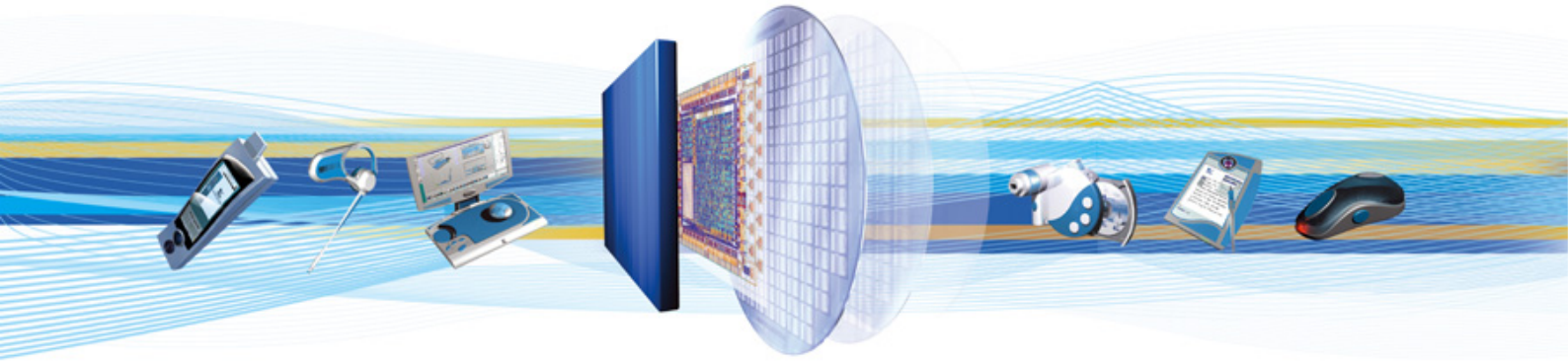
CSR Synergy Bluetooth 18.2.0

SPP – Serial Port 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 message interface provided by the Serial Port Profile, henceforward called SPP. The SPP requirements are specified in [SPP].

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. SPP and the application

2 Description

2.1 Introduction

The SPP manager supplies the interface for applications that should provide the functionality and conforms to the Serial Port Profile. The SPP is implemented as specified in the Serial Port Profile (K-5) [SPP]. The interface is in this document defined to be event based both in upwards and downwards direction, but can be mapped to a function base interface.

The SPP profile layer provides functionality for:

- Establishing and maintaining a channel between the SPP profile layer and a peer device, which conforms to the terminal part of the Serial Port Profile in part K-5 of [SPP]
- Sending and receiving data and control signals, between terminal and application

2.2 Reference Model

The application must interface to the SPP, the Security Controller, and the Connection Manager. The Connection Manager has an interface e.g. discovery functionality, and the Security Controller has an interface e.g. bonding functionality.

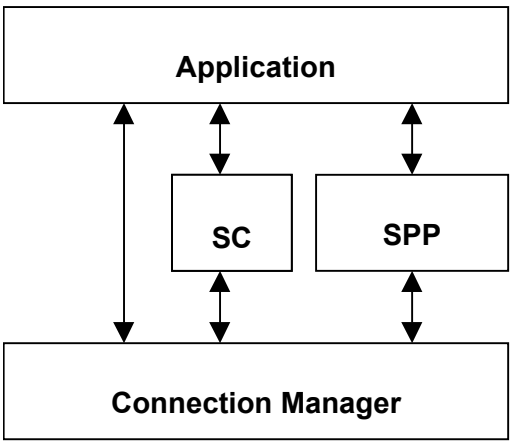


Figure 1: Reference model

The SC is optional and only necessary if Security is enabled. There is no profile requirement to use Security in the SPP. Security can be defined in the csr_bt_profiles.h file.

2.3 Communication Flow Architecture

The SPP profile implementation optionally supports divided communication channels (or paths): A Bluetooth® management signal flow and a data management flow. The communication flow architecture is depicted in Figure 2.

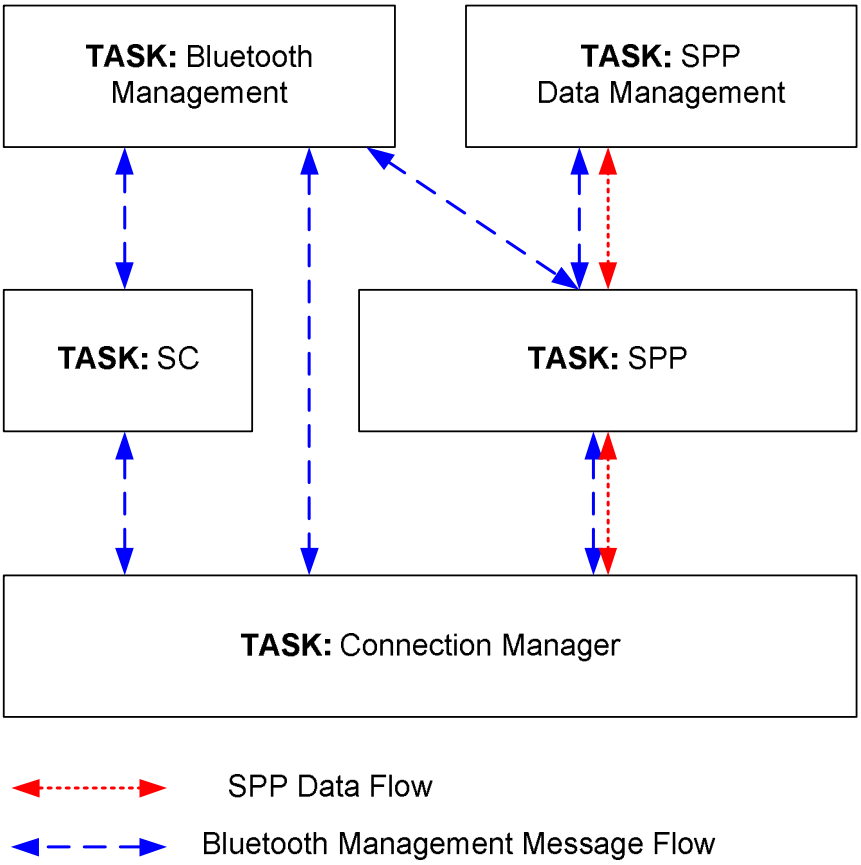


Figure 2: Communication Flow Architecture

The blue arrows depict the communication flow necessary for managing the Bluetooth® connection, whereas the red arrows depict the communication flow to be considered as SPP data.

This architecture implies that two application tasks must be registered in SPP when using its functionality[1]. The idea of supporting a Bluetooth® management flow and a SPP data flow enables more flexibility in the implementation of the application layer above the SPP profile.

Registration of the *Bluetooth Management* task is done when requesting a connection establishment (described in 3.3). The *Bluetooth Management* task will receive all messages (both management and data path messages) until a *SPP Data Management* task is registered. When the *SPP Data Management* task is registered, all messages related to the SPP data will be forwarded to the SPP *Data Management* task instead of the *Bluetooth Management* task.

An example of an application layer utilising the more flexible structure could be a separate application for controlling establishment of the Bluetooth® connection. Another separate application could be a device driver

[1] It can be the same task that is registered for the Bluetooth® connection management task and the DUN-DT data task.

appearing as a serial port device in the Operating System (OS) handling the internet connection establishment and forwarding of the received data to the IP stack of the OS.

In the interface and primitive descriptions it will be described explicitly whether the primitives are used for the Bluetooth® Management message flow or the *SPP Data Management* message flow.

2.4 Sequence Overview

The SPP starts up being in IDLE state. A connection may be initiated from either side; this is covered in the connection establishment section described in 3.2 and 3.4. Upon connection completion the SPP enters CONNECTED state where data and control signals may be exchanged between the two parties. The SPP remains in this state until the connection is released. When released it re-enters IDLE state.

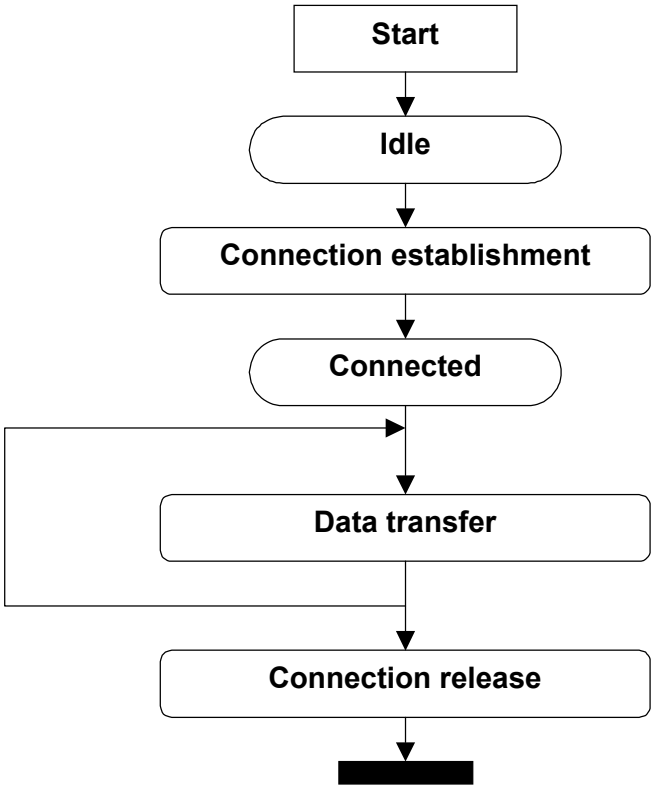


Figure 3: Sequence overview

3 Interface Description

3.1 Relations between Application and SPP Profile

The number of SPP profiles that can be run simultaneously is limited by the number of available RFCOMM channels. As this is limited to 60 (30 in each direction) it will not be possible to run more than 30 SPP's at the same time. This number may be lower if other profiles that also need RFCOMM channels are present.

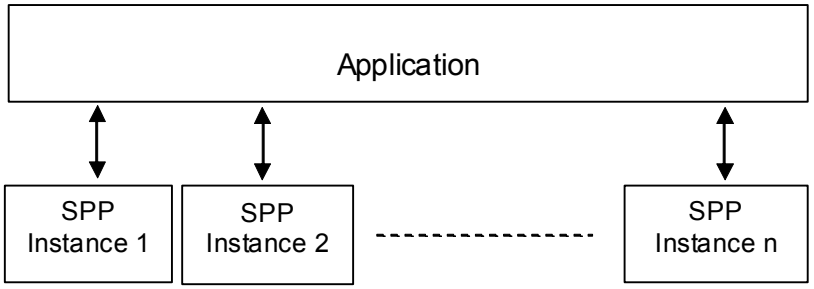


Figure 4: Application interfaces to the SPP profile

Each instance of the SPP profile has its own queue id, e.g. if the application must send a message to SPP instance 2, it must be put onto the message queue of SPP instance 2. The message queues of the different instances may be retrieved by means of the `CSR_BT_SPP_GET_INSTANCES_QID_REQ/CFM` messages.

SPP instance 1 is special because it is the SPP-manager where all other SPP instances register themselves.

This means that in order to get the register SPP-instances the `CSR_BT_SPP_GET_INSTANCES_QID_REQ` is to be sent to SPP-instance 1 which has queue ID = CSR_BT_SPP_IFACEQUEUE.

csr_bt_spp_lib.h defines a SPP access library, which provides functions for building and sending downstream SPP primitives. These functions also provide a parameter (queue id) which is the queue ID of the SPP instance that the message is to be sent to.

3.2 Getting the registered Instances

In order to get the registered SPP-instances the application sends the CSR_BT_SPP_GET_INSTANCES_QID_REQ to SPP-instance 1 (which has message queue CSR_BT_SPP_IFACEQUEUE). This returns the registered instances in a CSR_BT_SPP_GET_INSTANCES_QID_CFM. **Note:** Where signals include a queueId parameter, expect them to return one of the queue IDs given by this operation. Likewise, the queueId in downstream signals should be set to the queue ID corresponding to the instance in use.

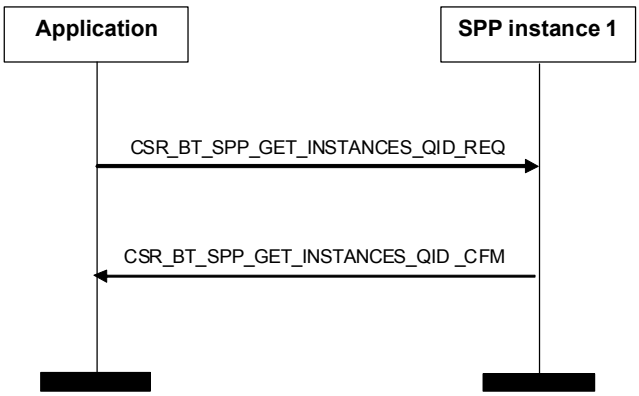


Figure 5: Retrieving registered SPP-instance

3.3 Accept Connection Establishment

In order to accept a connection initiating from a remote device, the SPP profile manager and the local device must be set in a mode where incoming connections are accepted. This is accomplished by issuing the CSR_BT_SPP_ACTIVATE_REQ from the application towards the SPP profile manager. After sending the CSR_BT_SPP_ACTIVATE_REQ the local device is connectable and the application receives a CSR_BT_SPP_ACTIVATE_CFM signal. The SPP manager stays connectable until a remote device has connected or the time expires. The CSR_BT_SPP_CONNECT_IND is an indication that the connection is established. The CSR_BT_SPP_STATUS_IND message with the connect parameter is sent to the data flow if applicable.

Please note that whether or not the Bluetooth device will be discoverable, i.e. can be found by other Bluetooth devices, it must be controlled by the application. For more information, please refer to [CM]. After initialization of CSR Synergy Bluetooth the Bluetooth® device is set up to be discoverable.

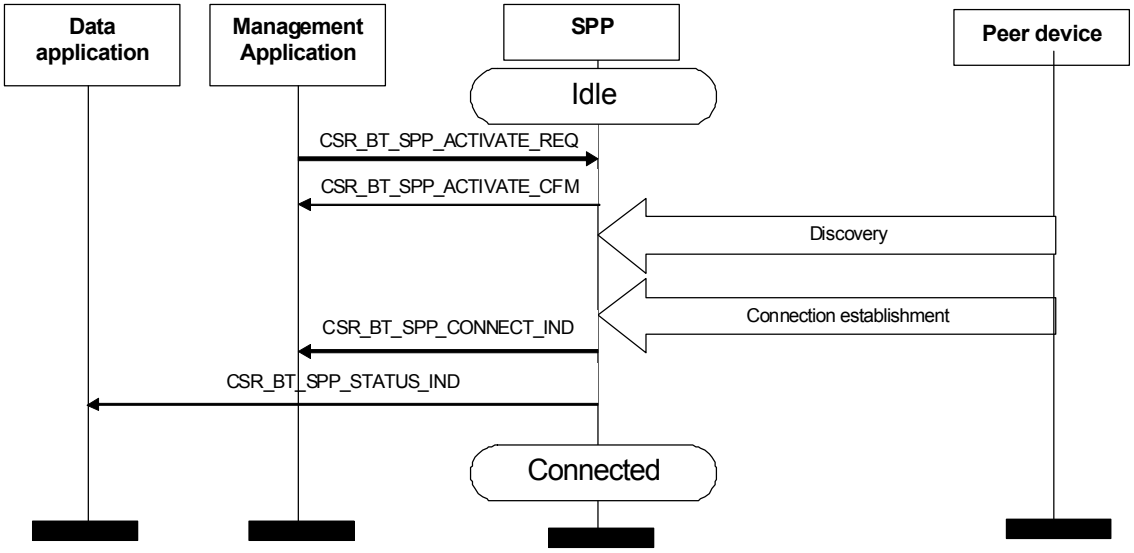


Figure 6: Accept connection establishment sequence

The SPP Class Of Device (COD) is defined as a compiler switch and is set in init. The COD is defined in csr_bt_spp_main.h.

If the application decides that the SPP service should no longer be connectable, the application may send a CSR_BT_SPP_DEACTIVATE_REQ and get a CSR_BT_SPP_DEACTIVATE_CFM back when deactivated. Please note that this is a local procedure that does not invoke any remote side functions.

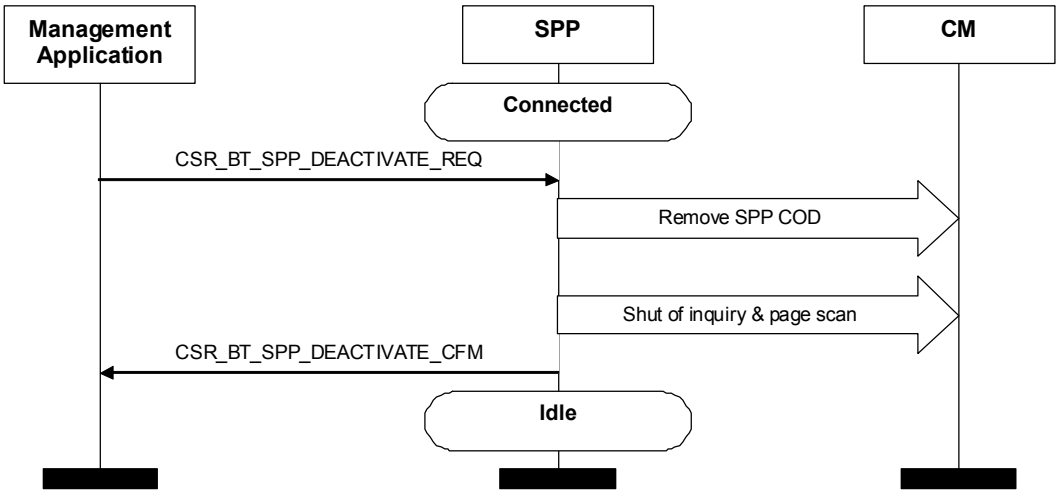


Figure 7: Deactivate accept connection establishment sequence

Note that if the connection establishment fails or the service is deactivated, the application must send a CSR_BT_SPP_ACTIVATE_REQ in order to accept incoming connection again.

3.4 Initiate Connection Establishment

It is possible to initiate a connection to a remote data terminal. Issuing a CSR_BT_SPP_CONNECT_REQ from the application will accomplish this. Doing connect the SPP profile starts service discovery to retrieve the remote data terminal service name(s), indicated with CSR_BT_SPP_SERVICE_NAME_IND. The application must then select the service name it wants to initiate a connection to by sending a CSR_BT_SPP_SERVICE_NAME_RES. Completion of the connection is indicated with a CSR_BT_SPP_CONNECT_IND to the management flow and a CSR_BT_SPP_STATUS_IND to the data flow. If remote port negotiation is required this must be indicated in the connect request signal, see 4.4 for further details about the parameters.

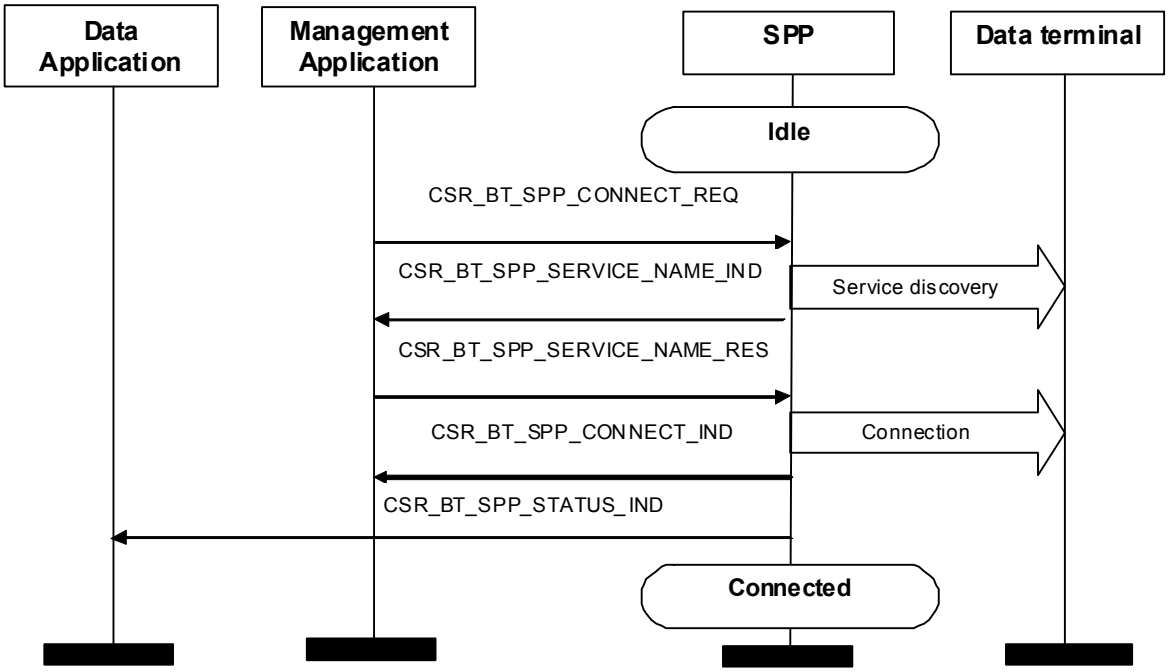


Figure 8: Locally initiated connection establishment sequence

Before sending CSR_BT_SPP_CONNECT_REQ the Bluetooth® device address of the device must be known. If not it can be found by using the discovery functionality.

If remote port negotiation is requested the connect indication includes the parameters/response from the remote device.

Note: that the application is not allowed to send a data packet before it receives a CSR_BT_SPP_CONNECT_IND with CSR_BT_SUCCESS.

3.5 Modem Control Signal

The local modem line status can be transferred to the remote side.

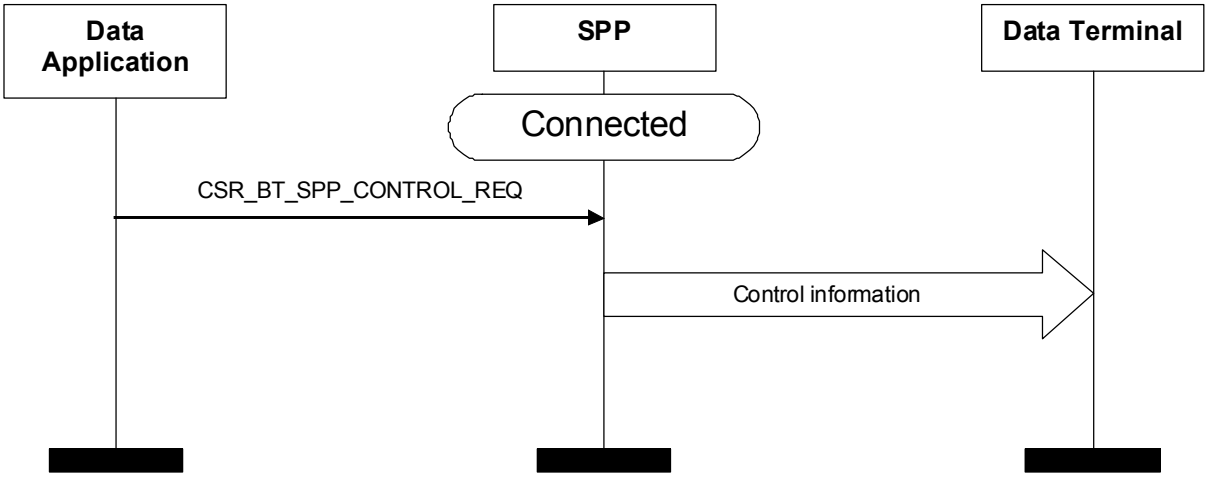


Figure 9: Set the modem status

The remote side modem line status is indicated to the application layer through a CSR_BT_SPP_CONTROL_IND signal.

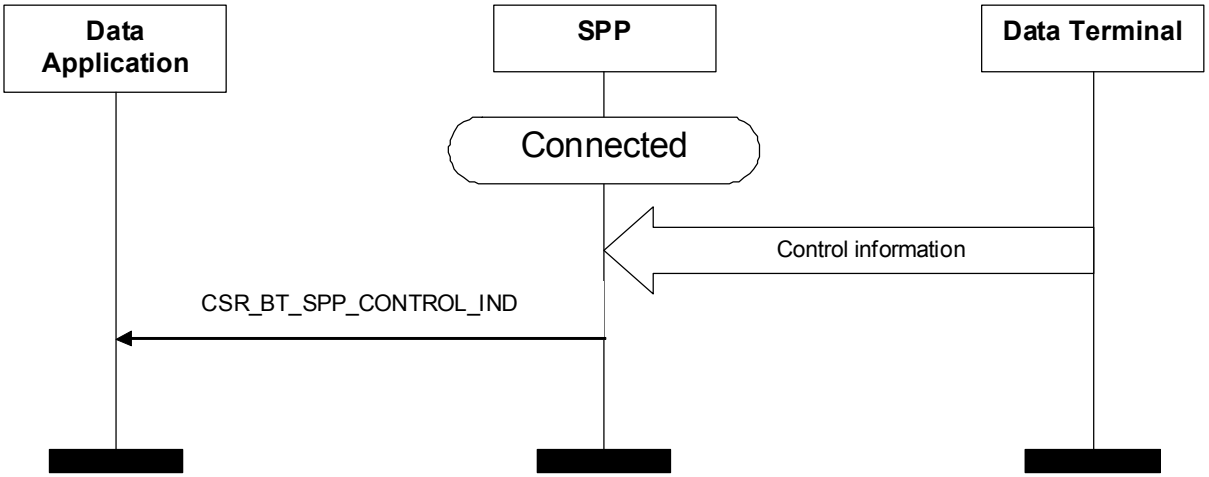


Figure 10: Read the modem status

Please note that after a CSR_BT_SPP_CONNECT_IND is received with CSR_BT_SUCCESS the application is required to send information on all changes in RS232 control signals with the modem status command, e.g. CSR_BT_SPP_CONTROL_REQ.

3.6 Remote Port Negotiation

If the remote side requests remote port negotiation this is indicated to the application layer in a CSR_BT_SPP_PORTNEG_IND signal. This signal can be received at any time, even before a connection request is confirmed.

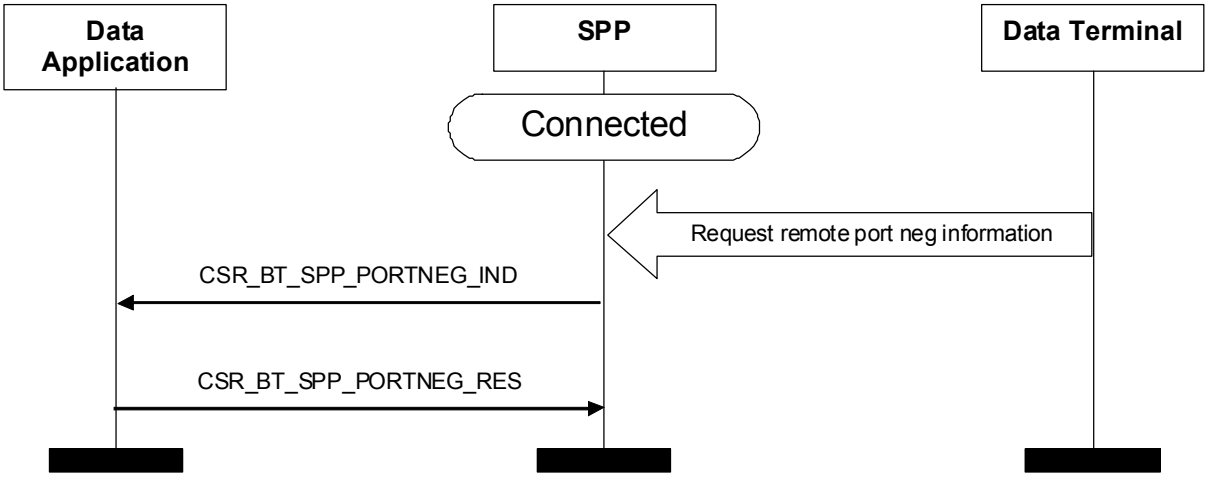


Figure 11: Remote port negotiation request

The remote side can either indicate its own current values or request values from the local side.

3.7 Data Transfer

3.7.1 Upstream Data

Data can be received from the peer Bluetooth® device and is forwarded to the application.

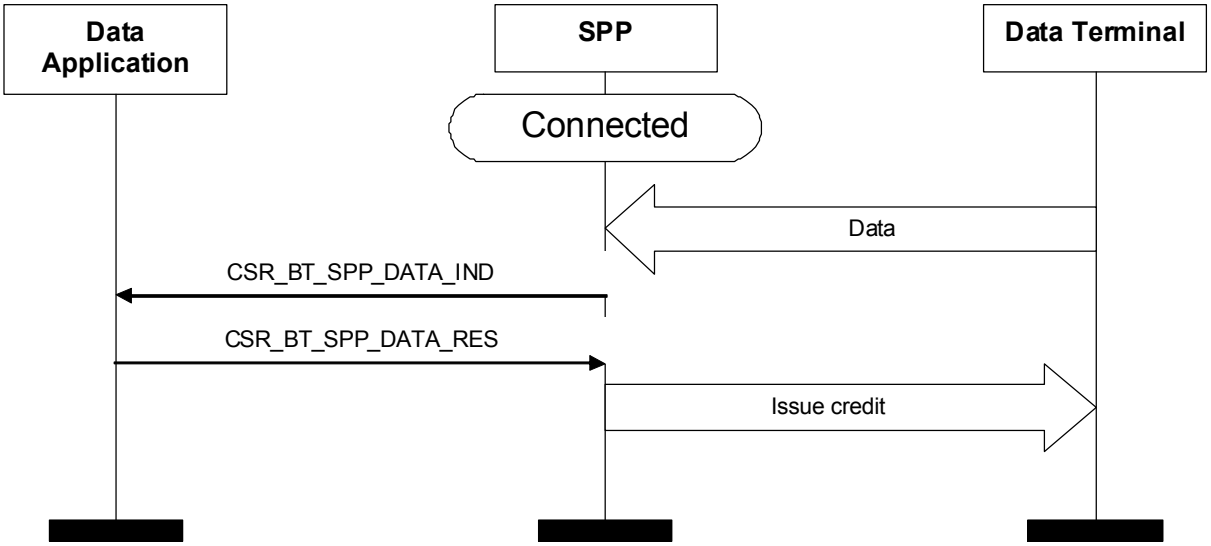


Figure 12: Upstream data transfer

The CSR_BT_SPP_DATA_RES is used for informing the SPP that data is handled and that the received data frame can be released. As long as the CSR_BT_SPP_DATA_RES has not been sent to SPP, the application layer has access to the data in the received frame. The SPP manager will ensure that a credit information package is issued to the terminal side when CSR_BT_SPP_DATA_RES is received.

Note that the application is responsible of releasing the payload received in a CSR_BT_SPP_DATA_IND signal.

3.7.2 Downstream Data

Data can be received from the application layer and forwarded to the remote device.

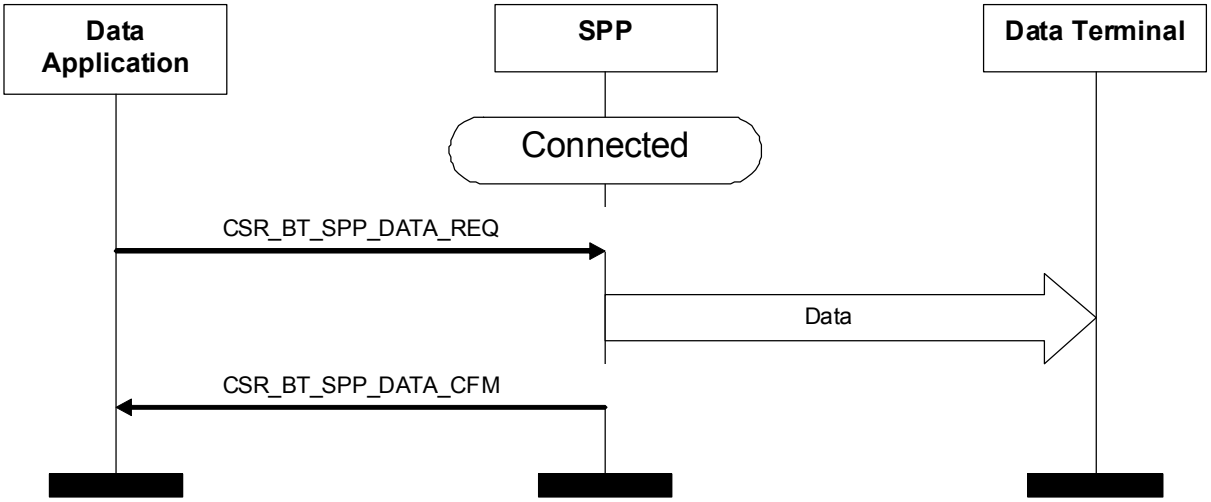


Figure 13: Downstream data

The SPP will forward the received data to the lower layers of the Bluetooth® stack for transmission to the data terminal side. Once the data is sent to the lower layers a CSR_BT_SPP_DATA_CFM is issued to the application layer, which is now allowed to send a new data packet. Note that the CSR_BT_SPP_DATA_CFM not necessarily implies that the data has been received on the data terminal side.

3.8 Link Mode

3.8.1 Link Mode Request

If a connection is established the CSR_BT_SPP_MODE_CHANGE_REQ message can be used for requesting a change of the link mode, e.g. Active (no power saving mode), Sniff, or Park mode.

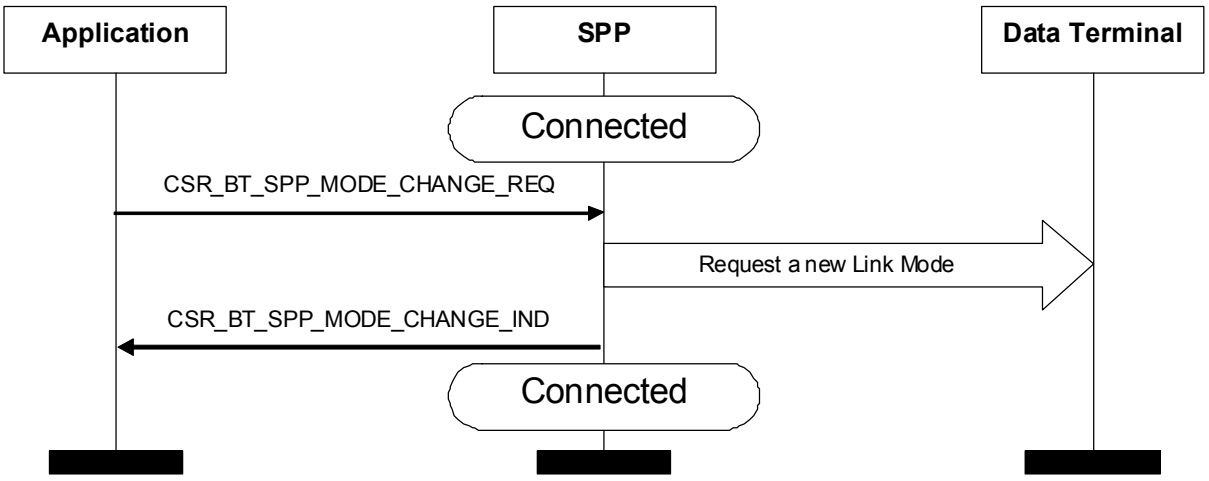


Figure 14: Link Mode Request sequence

3.8.2 Link Mode by Peer Side

If a connection is established and the peer device changes the link mode the application will receive a CSR_BT_SPP_MODE_CHANGE_IND.

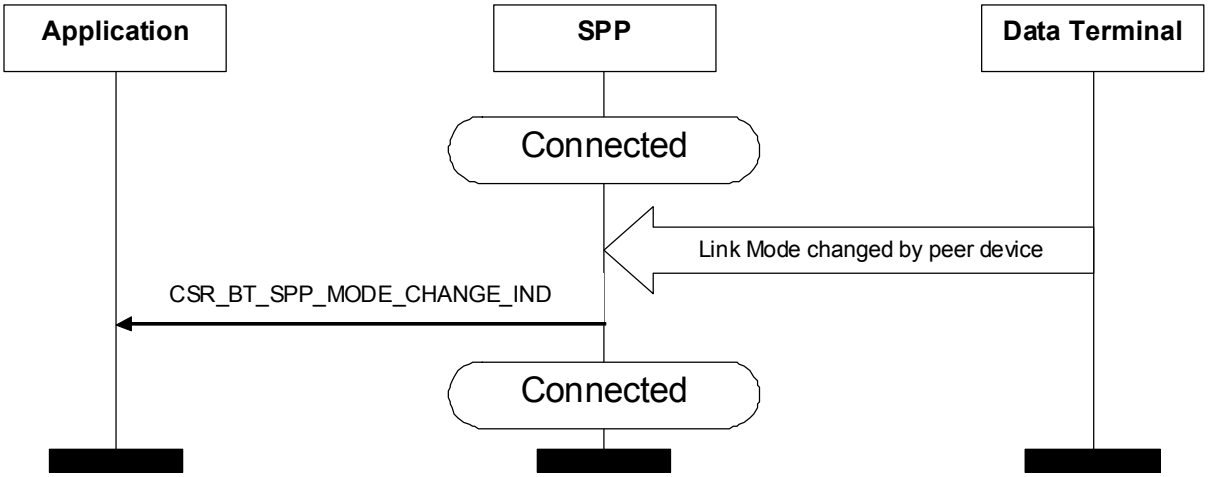


Figure 15: Peer side Link Mode sequence

3.9 Connection Release

3.9.1 Connection Release

Once a connection is established, the application may release it. A connection is released by sending a CSR_BT_SPP_DISCONNECT_REQ to the SPP manager. The CSR_BT_SPP_STATUS_IND will have the connect flag set to false, so the data flow can detect when the connection was released.

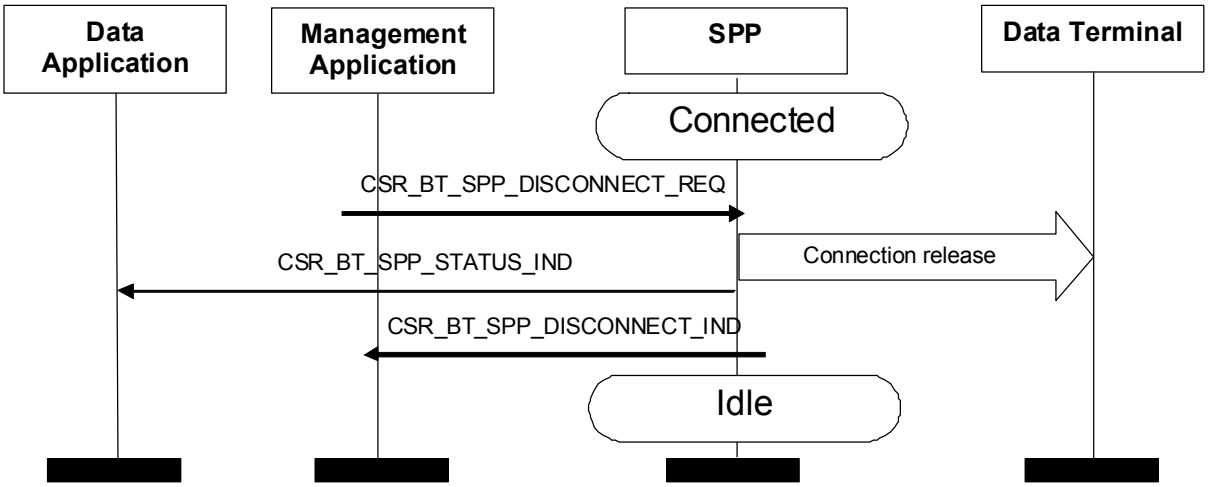


Figure 16: Connection release sequence

3.9.2 Peer Side Connection Release

The connected terminal at the remote end may at any time release the connection. Further, it is possible that the physical link is closed due to an abnormal situation, e.g. radio interference or the devices getting out of coverage from each other.

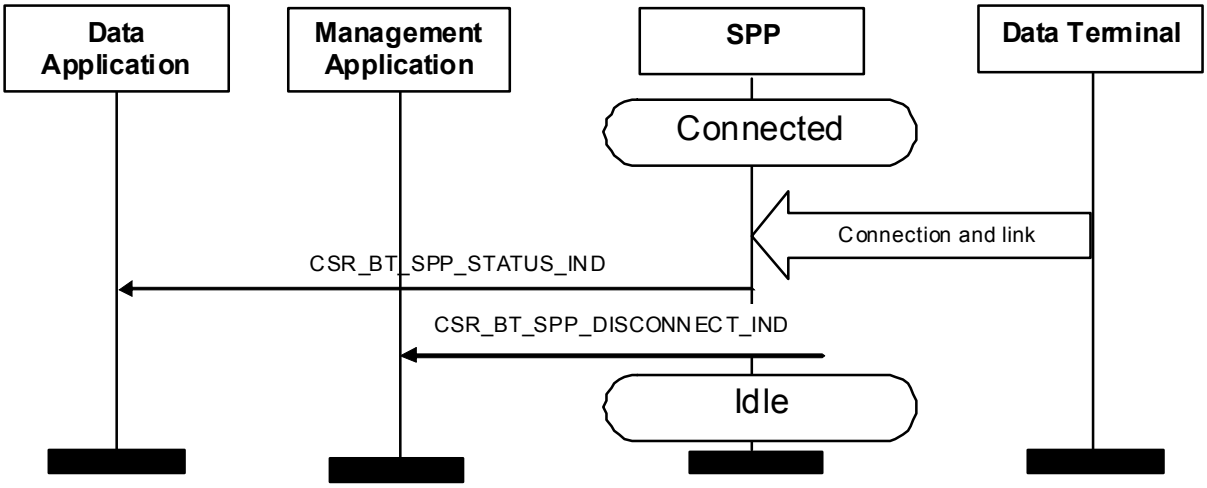


Figure 17: Peer side connection release sequence

The SPP manager will handle the disconnect signal in a similar manner no matter the reason for the release. A new CSR_BT_SPP_ACTIVATE_REQ may be issued to make the service available again.

4 SPP Primitives

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

4.1 List of All Primitives

Primitives	Reference
CSR_BT_SPP_ACTIVATE_REQ	See section 4.2
CSR_BT_SPP_ACTIVATE_CFM	See section 4.2
CSR_BT_SPP_DEACTIVATE_REQ	See section 4.3
CSR_BT_SPP_DEACTIVATE_CFM	See section 4.3
CSR_BT_SPP_CONNECT_REQ	See section 4.4
CSR_BT_SPP_CONNECT_IND	See section 4.4
SSP_SERVICE_NAME_IND	See section 4.5
CSR_BT_SPP_SERVICE_NAME_RES	See section 4.5
CSR_BT_SPP_DATA_REQ	See section 4.6
CSR_BT_SPP_DATA_CFM	See section 4.6
CSR_BT_SPP_DATA_IND	See section 4.6
CSR_BT_SPP_DATA_RES	See section 4.6
CSR_BT_SPP_MODE_CHANGE_REQ	See section 4.7
CSR_BT_SPP_MODE_CHANGE_IND	See section 4.7
CSR_BT_SPP_CONTROL_REQ	See section 4.8
CSR_BT_SPP_CONTROL_IND	See section 4.8
CSR_BT_SPP_PORTEG_IND	See section 4.9
CSR_BT_SPP_PORTEG_RES	See section 4.9
CSR_BT_SPP_PORTNEG_REQ	See section 4.9
CSR_BT_SPP_PORTNEG_CFM	See section 4.9
CSR_BT_SPP_DISCONNECT_REQ	See section 4.10
CSR_BT_SPP_DISCONNECT_IND	See section 4.10
CSR_BT_SPP_GET_INSTANCES_QID_REQ	See section 4.11
CSR_BT_SPP_GET_INSTANCES_QID_CFM	See section 4.11
CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE_REQ	See section 4.12
CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE_CFM	See section 4.12
CSR_BT_SPP_DATA_PATH_STATUS_REQ	See section 4.13
CSR_BT_SPP_DATA_PATH_STATUS_IND	See section 4.13
CSR_BT_SPP_STATUS_IND	See section 4.14
CSR_BT_SPP_CANCEL_CONNECT_REQ	See section 4.15
CSR_BT_SPP_SECURITY_IN_REQ	See section 4.16
CSR_BT_SPP_SECURITY_IN_CFM	See section 4.16
CSR_BT_SPP_SECURITY_OUT_REQ	See section 4.16
CSR_BT_SPP_SECURITY_OUT_CFM	See section 4.16

Table 1: List of all primitives

4.2 CSR_BT_SPP_ACTIVATE

Parameters / Primitives	type	phandle	timeout	role	serviceName	queueId	serverChannel
CSR_BT_SPP_ACTIVATE_REQ	✓	✓	✓	✓	✓		
CSR_BT_SPP_ACTIVATE_CFM	✓					✓	✓

Table 2: CSR_BT_SPP_ACTIVATE Primitive

Description

This signal is used for activating a service and making it connectable. The process includes:

- Registration of the SPP service in the service discovery database
- Enabling page scan

The service availability can be time limited or may run forever.

Please note that the CSR_BT_SPP_ACTIVATE_REQ may fail in which case the application will *not* receive a CSR_BT_SPP_ACTIVATE_CFM. However, a CSR_BT_SPP_CONNECT_IND signal (see section 4.4) will be sent to notify that the request has failed.

Parameters

type	Signal identity, CSR_BT_SPP_ACTIVATE_REQ/CFM.
phandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to phandle.
timeout	The time, in number of seconds, where the service should be available for remote connections. The maximum Timeout time allowed is defined by the CSR_BT_MAX_TIME in the file “csr_bt_profiles.h”. A timeout value of 0 (zero) or the define CSR_BT_INFINITE_TIME in the file “csr_bt_profiles.h” indicates an infinity timeout time. When a connection is made, the service available stops.
role	Decides if SPP role is DTC or DCE.
serviceName	Name of the service the application is looking for (ignored in phase I, see csr_bt_spp_sef.c). This field will be CsrPfreed by the spp profile to avoid memory leaks.
queueId	Identifier of the SPP instance which has been activated.
serverChannel	The local server number, which is a reference ID used by the Connection Manager.

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_ACTIVATE_REQ signal:

```
void CsrBtSppActivateReqSend(CsrSchedQid sppQueueId, CsrSchedQid appHandle,
cplTimer_t timeout, CsrUint8 role, CsrCharString *serviceName);
```

The `sppQueueId` parameter is used for indicating the destination SPP instance of the signal. The remaining parameters are as described in the table above.

4.3 CSR_BT_SPP_DEACTIVATE

Parameters / Primitives	type	phamdle	resultCode	resultSupplier
CSR_BT_SPP_DEACTIVATE_REQ	✓	✓		
CSR_BT_SPP_DEACTIVATE_CFM	✓	✓	✓	✓

Table 3: CSR_BT_SPP_DEACTIVATE Primitives

Description

Deactivate a service that has been activated previously. The service will no longer be connectable

Parameters

type	Signal identity, CSR_BT_SPP_DEACTIVATE_REQ/CFM.
phandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to phandle.
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. All values which are currently not specified in the respective prim.h file 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

Library Function

A library function is provided by CSR_BT_SPP_lib.h, and should be used for building and sending the CSR_BT_SPP_DEACTIVATE_REQ signal:

```
void CsrBtSppDeactivateReqSend(CsrSchedQid sppQueueId, CsrSchedQid appHandle);
```

The `sppQueueId` parameter is used for indicating the destination SPP instance of the signal. The appHandle parameter is used for indicating which application should receive the confirm signal.

4.4 CSR_BT_SPP_CONNECT

Primitives \ Parameters	type	phandle	resultCode	resultSupplier	deviceAddr	role	validPortPar	requestPortPar	portPar	serviceName	queueId	profileMaxFrameSize	btConnId
CSR_BT_SPP_CONNECT_REQ	✓	✓			✓	✓	✓	✓	✓	✓			
CSR_BT_SPP_CONNECT_IND	✓		✓	✓	✓		✓		✓		✓	✓	✓

Table 4: CSR_BT_SPP_CONNECT Primitives

Description

A CSR_BT_SPP_CONNECT_REQ will initiate a connection towards a device specified by the Bluetooth® device address. The SPP will send a CSR_BT_SPP_CONNECT_IND back to the initiator with the result of the connection attempt. If the connection is not established within the time frame given, the connection attempt will be aborted and indicated in CSR_BT_SPP_CONNECT_IND.

If the SPP service has been activated with CSR_BT_SPP_ACTIVATE_REQ and a remote data terminal initiates a connection, a CSR_BT_SPP_CONNECT_IND will be submitted to the application layer as well. If the SPP is active (a CSR_BT_SPP_ACTIVATE_REQ has been submitted) the scanning procedure will be aborted.

Parameters

type	Signal identity, CSR_BT_SPP_CONNECT_REQ/IND.
phandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to phandle.
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. All values which are currently not specified in the respective prim.h file 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
deviceAddr	The Bluetooth® device address to which a connection must be/or is established.
role	Decides if SPP role is DTC or DCE.
validPortPar	TRUE if remote port negotiation is to be initiated during connection establishment. If set to FALSE the values of requestPortPar and portPar are insignificant.
requestPortPar	TRUE if this is a request to retrieve the peer side port parameters. FALSE if own side parameters are transmitted.
portPar	Port parameters as defined in the `RFC_PORTNEG_VALUES_T` structure. The portPar is a structure defined as `RFC_PORTNEG_VALUES_T`, see below.
queueId	Identifier of the SPP instance in use.
serverChannel	The local server number, which is a reference ID used by the Connection Manager.

profileMaxFrameSize — The maximum payload length allowed in up– and downstream data requests. Please note that it is possible to receive one octet (byte) more than indicated in the profileMaxFrameSize parameter.

```
typedef struct
{
   CsrUint8     baud_rate;      /* port speed indicator - see #defines above */
   CsrUint8     data_bits;      /* DATA_BITS_5, _6, _7 or _8 - see above */
   CsrUint8     stop_bits;      /* STOP_BITS_ONE or _ONE_AND_A_HALF - see above */
   CsrUint8     parity;         /* PARITY_OFF or PARITY_ON */
   CsrUint8     parity_type;    /* PARITY_TYPE_ODD, _EVEN, _MARK or _SPACE */
   CsrUint8     flow_ctrl_mask; /* 6 bits - use FLC_ #defines above (see 07.10) */
   CsrUint8     xon;            /* xon character  (default DC1 0x11) */
   CsrUint8     xoff;           /* xoff character (default DC3 0x13) */
   CsrUint16    parameter_mask; /* 16 bits (top two reserved) see PM_ #defines */
} RFC_PORTNEG_VALUES_T;
```

Baud_rate — Takes the form `RFC_xxxx_BAUD`, where `xx` is the port speed in bits per second.

Encoded as:

0x00	=	2400
0x01	=	4800
0x02	=	7200
0x03	=	9600
0x04	=	19200
0x05	=	38400
0x06	=	57600
0x07	=	115200
0x08	=	230400
0xFF	=	RFC_UNKNOWN_BAUD

dataBits — Number of data bits encoded as an unsigned integer. Valid values are 5, 6, 7 and 8.

Encoded as:

0x00	=	5 bit
0x02	=	6 bit
0x01	=	7 bit
0x03	=	8 bit

stopBits — Encoded as:

0x00	=	1 stop bit
0x01	=	1.5 stop bits

Parity — Encoded as:

0x00	=	PARITY_OFF
0x01	=	PARITY_ON

parityType — Encoded as:

0x00	odd parity
0x02	even parity
0x01	mark parity
0x03	space parit

flowCtrlMask — Encoded as:

Bit 0	XON / XOFF, input
Bit 1	XON / XOFF, output
Bit 2	RTR input

Bit 3 RTR output
Bit 4 RTC input
Bit 5 RTC output

Xon — Xon character (default DC1, 0x11)

Xoff — Xoff character (default DC3, 0x13)

parameterMask — `parameter_mask` is used for indicating the parameters in the Remote Port Negotiation command, which are negotiated. The command, `parameter_mask` is interpreted as follows:

0 no change
1 change

For a response the following values apply:

0 not accepted proposal
1 accepted proposal - the new values are used

The bit mask is shown as:
Bit0 bit rate
Bit1 data bits
Bit2 stop bits
Bit3 Parity
Bit4 parity type
Bit5 XON character
Bit6 XOFF character
Bit7 Reserved
Bit8 XON / XOFF, input
Bit9 XON / XOFF, output
Bit10 RTR input
Bit11 RTR output
Bit12 RTC input
Bit13 RTC output
Bit14 Reserved, set 0 by sender
Bit15 Reserved, set 0 by sender

btConnId — Identifier used when moving the connection to another AMP controller, i.e. when calling the `CsrBtAmpmMoveReqSend`-function.

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_CONNECT_REQ signal:

```
void CsrBtSppConnectReqSend(CsrSchedQid sppQueueId, CsrSchedQid appHandle,
                                deviceAddr_t deviceAddr,
                                CsrBool requestPortPar,
                                RFC_PORTNEG_VALUES_T *portPar,
                                CsrUint8 role );
```

The `sppQueueId` parameter is used for indicating the destination SPP instance of the signal. The remaining parameters are as described in the table above.

4.5 CSR_BT_SPP_SERVICE_NAME

Primitives \ Parameters	type	queueId	* serviceNameList	serviceNameListSize	serviceHandle	accept
CSR_BT_SPP_SERVICE_NAME_IND	✓	✓	✓	✓		
CSR_BT_SPP_SERVICE_NAME_RES	✓				✓	✓

Table 5: CSR_BT_SPP_SERVICE_NAME Primitives

Description

When connecting, it retrieves the remote device SDP database(s) and sends the service name to the application. The application must select a service name it wants to connect to by sending a CSR_BT_SPP_SERVICE_NAME_RES.

Parameters

type	Signal identity, CSR_BT_SPP_SERVICE_NAME_IND/RES.
queueId	Identifier of the SPP instance in use.
*serviceNameList	A list of Service Names and service record handles, retrieve from the service record. In order to prevent a memory leak this parameter must be CsrPfreed by the application
serviceNameListSize	Number of Service Names returned
serviceHandle	The 32-bit service record handle, which is attached to the Service name the application will connect to.
accept	TRUE will start initializing a connection, FALSE will cancel connect procedure

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_SERVICE_NAME_RES signal:

```
void CsrBtSppServiceNameResSend(CsrSchedQid queueId, CsrBool connect,
                                uuid32 theServiceHandle);
```

The `queueId` parameter is used for indicating the destination SPP instance of the signal. The value of the `connect` Boolean in the library function is written to the signal's `accept` parameter. The remaining parameters are as described in the table above.

4.6 CSR_BT_SPP_DATA

Primitives \ Parameters	type	queueId	serverChannel	payloadLength	*payload
CSR_BT_SPP_DATA_REQ	✓		✓	✓	✓
CSR_BT_SPP_DATA_CFM	✓	✓	✓		
CSR_BT_SPP_DATA_IND	✓	✓	✓	✓	✓
CSR_BT_SPP_DATA_RES	✓		✓		

Table 6: CSR_BT_SPP_DATA Primitives

Description

Send and receive data.

Parameters

type	Signal identity, CSR_BT_SPP_DATA_REQ/CFM/IND/RES.
queueId	Identifier of the SPP instance in use.
serverChannel	The local server number, which is a reference ID used by the Connection Manager.
payloadLength	Length of data in number of octets.
*payload	Pointer reference to the actual payload.

Library Function

Library functions are provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_DATA_REQ and CSR_BT_SPP_DATA_RES signals:

```
void CsrBtSppDataReqSend( CsrSchedQid   queueId,
                          CsrUint8      theServerChannel,
                          CsrUint16     thePayloadLength,
                          CsrUint8      *thePayload  );

void CsrBtSppDataResSend( CsrSchedQid   queueId,
                          CsrUint8      theServerChannel);
```

The queue`Id` parameters are used for indicating which SPP instance the downstream signal is meant for, and which instance the upstream signal came from. The remaining parameters are as described in the table above.

4.7 CSR_BT_SPP_MODE_CHANGE

Parameters / Primitives	type	queueId	serverChannel	mode	resultCode	resultSupplier
CSR_BT_SPP_MODE_CHANGE_REQ	✓		✓	✓		
CSR_BT_SPP_MODE_CHANGE_IND	✓	✓	✓	✓	✓	✓

Table 7: CSR_BT_SPP_MODE_CHANGE Primitives

Description

If a connection is made the CSR_BT_SPP_MODE_CHANGE_REQ message can be used for requesting a change of the link mode, e.g. Active (no power saving mode), Sniff, or Park mode. Please notice that when a connection is made it is registered to run in ACTIVE mode per default.

The application can request the link to enter ACTIVE, SNIFF, or PARK mode by sending a CSR_BT_SPP_MODE_CHANGE_REQ.

The CSR_BT_SPP_MODE_CHANGE_IND indicates either that the remote device has changed the current link mode or the request has been accepted by the CM.

In the case where more CSR Synergy Bluetooth profiles are connected to the same peer device the Connection Manager (CM) will coordinate this with the following priorities: 1) ACTIVE, 2) SNIFF, 3) PARK.

An example of this could be: if two SPP instances A and B are connected to the same peer device (using the same ACL link) and device A requests to enter SNIFF mode while B is in ACTIVE mode, then the CM will save this request in its database, and return a CSR_BT_SPP_MODE_CHANGE_IND with the result CSR_BT_SUCCESS but the link will stay in ACTIVE mode. If device B releases its connection or its requests SNIFF or PARK mode, then the CM will try to set the link in SNIFF mode.

In general each profile has a data based entry where it saves the mode change request from the profile. The mode request stays in the database until the profile requests another mode or the connection is released.

Please notice that the CM will per default accept a link mode change initiated by the peer device (if ACTIVE, SNIFF or PARK mode is requested) but the requested mode from the local profile will still be registered in the CSR Synergy Bluetooth database. E.g. if an application has requested SNIFF mode and the peer device requests the link in SNIFF or PARK mode this will always be accepted per default.0

An example of this could be: if a local profile requests SNIFF mode and this request is succeeded and if the peer device then requests a mode different from the SNIFF mode then CSR Synergy Bluetooth will per default accept this request even if it has a lower priority (in this case PARK mode), however the original request (SNIFF mode) will still be in the database. I.e. if the profile has requested ACTIVE mode and the peer device sets it in SNIFF mode then the profile must request ACTIVE mode in order to get the link in ACTIVE mode again.

Please note that it is possible to define which link mode the CM will accept by sending a CSR_BT_CM_WRITE_LINK_POLICY_REQ, for more information refers to the [CM].

Please notice that if the mode of an ACL link is changed, either by a local profile or a peer device, then all the instances using this ACL link will receive a CSR_BT_SPP_MODE_CHANGE_IND with the new mode the link is in.

Finally if the link is in PARK mode then it is up to the profile to request ACTIVE or SNIFF mode before sending data.

Parameters

type	Signal identity, CSR_BT_SPP_MODE_CHANGE_REQ/IND.
queueId	Handle of SPP instance which has just changed connection status.
serverChannel	The local server number, which is a reference ID used by the Connection Manager.
mode	Sets/indicates the mode of the link ACTIVE_MODE (0x0000) Sets/indicates that the link is in Active mode SNIFF_MODE (0x0002) Sets/indicates that the link is in Sniff mode PARK_MODE (0x0003) Sets/indicates that the link is in Park mode
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. All values which are currently not specified in the respective prim.h file 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

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_MODE_CHANGE_REQ signal:

```
void CsrBtSppModeChangeReqSend(  CsrSchedQid   queueId,
                                 CsrUint8      theServerChannel,
                                 CsrUint8      theMode);
```

The queue`Id` parameter is used for indicating the destination SPP instance of the signal. The remaining parameters are as described in the table above.

4.8 CSR_BT_SPP_CONTROL

Primitives \ Parameters	type	queueId	serverChannel	modemStatus	break_signal
CSR_BT_SPP_CONTROL_REQ	✓		✓	✓	✓
CSR_BT_SPP_CONTROL_IND	✓	✓	✓	✓	✓

Table 8: CSR_BT_SPP_CONTROL Primitives

Description

Send and receive modem status information.

Parameters

type — Signal identity, CSR_BT_SPP_CONTROL_REQ/IND.

queueId — Identifier of the SPP instance in use.

serverChannel — The local server number, which is a reference ID used by the Connection Manager.

modemStatus — The modem status contains the following bit from the RS-232 interface:

Bit 0 CTS (Clear to Send)
Bit 1 RTS (Request to Send)
Bit 2 DSR (Data Set Ready)
Bit 3 DTR (Data terminal Ready)
Bit 4 RI (Ring Indicator)
Bit 5 DCD (Data Carrier Detect)

There is a mask code for this bit in the csr_bt_profiles.h. The mask code must be used.

break_signal — The `break_signal` is encoded as follows:

Bit 0 Not used.
Bit 1 0: No break signal encoded. 1: Break signal encoded.
Bit 2 Not used.
Bit 3 Not used.
Bit 4-7 Duration of break signal in 200mS increments.

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_CONTROL_REQ signal:

```
void CsrBtSppControlReqSend(  CsrSchedQid  queueId,
                              CsrUint8     theServerChannel,
                              CsrUint8     theModemStatus,
                              CsrUint8     theBreakSignal);
```

The `queueId` parameter is used for indicating the destination SPP instance of the signal. The remaining parameters are as described in the table above.

4.9 CSR_BT_SPP_PORTNEG

Primitives \ Parameters	type	serverChannel	queueId	portPar	request	resultCode	resultSupplier
CSR_BT_SPP_PORTNEG_IND	✓	✓	✓	✓	✓		
CSR_BT_SPP_PORTNEG_RES	✓	✓		✓			
CSR_BT_SPP_PORTNEG_REQ	✓	✓	✓	✓			
CSR_BT_SPP_PORTNEG_CFM	✓	✓	✓	✓		✓	✓

Table 9: CSR_BT_SPP_PORTNEG Primitives

Description

Send and receive port set-up information. If the remote device changes the port settings or requests the current settings, the SPP will send a CSR_BT_SPP_PORTNEG_IND to the application. The application shall then answer with a CSR_BT_SPP_PORTNEG_RES. If the application wants to change the port settings after a connection has been established, it shall use the CSR_BT_SPP_PORTNEG_REQ. Once the operation is performed it will receive a CSR_BT_SPP_PORTNEG_CFM with the outcome.

Parameters

type	Signal identity, CSR_BT_SPP_PORTNEG_IND/RES/REQ/CFM.
serverChannel	The local server number, which is a reference ID used by the Connection Manager.
queueId	Identifier of the SPP instance in use.
portPar	The portPar is a structure defined as `RFC_PORTNEG_VALUES_T`. The `RFC_PORTNEG_VALUES_T` structure, shown below, is included into the PORTNEG primitive. In the library function call the `RFC_PORTNEG_VALUES_T` structure should be called as a pointer and the library function will copy the data into the PORTNEG primitive.
request	TRUE: If the request is TRUE, the remote device requests the local `RFC_PORTNEG_VALUES_T` settings. The receiver must respond with current `RFC_PORTNEG_VALUES_T` settings. FALSE: The remote device must confirm the settings or propose new ones. Setting new suggested values includes setting also the proper parameter mask.
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. All values which are currently not specified in the respective prim.h file 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

```
typedef struct
{
   CsrUint8     baud_rate;      /* port speed indicator - see #defines above */
   CsrUint8     data_bits;      /* DATA_BITS_5, _6, _7 or _8 - see above */
   CsrUint8     stop_bits;      /* STOP_BITS_ONE or _ONE_AND_A_HALF - see above */
   CsrUint8     parity;         /* PARITY_OFF or PARITY_ON */
   CsrUint8     parity_type;    /* PARITY_TYPE_ODD, _EVEN, _MARK or _SPACE */
   CsrUint8     flow_ctrl_mask; /* 6 bits - use FLC_ #defines above (see 07.10) */
   CsrUint8     xon;            /* xon character  (default DC1 0x11) */
   CsrUint8     xoff;           /* xoff character (default DC3 0x13) */
   CsrUint16    parameter_mask; /* 16 bits (top two reserved) see PM_ #defines */
} RFC_PORTNEG_VALUES_T;
```

Baud_rate

Takes the form `RFC_xxxx_BAUD`, where `xx` is the port speed in bits per second.

Encoded as:

0x00 = 2400
0x01 = 4800
0x02 = 7200
0x03 = 9600
0x04 = 19200
0x05 = 38400
0x06 = 57600
0x07 = 115200
0x08 = 230400
0xFF = RFC_UNKNOWN_BAUD

data_bits	Number of data bits encoded as an unsigned integer. Valid values are 5, 6, 7 and 8. Encoded as: 0x00 = 5 bit 0x02 = 6 bit 0x01 = 7 bit 0x03 = 8 bit
stop_bits	Encoded as: 0x00 = 1 stop bit 0x01 = 1.5 stop bits
Parity	Encoded as: 0x00 = PARITY_OFF 0x01 = PARITY_ON
parity_type	Encoded as: 0x00 odd parity 0x02 even parity 0x01 mark parity 0x03 space parity
flow_ctrl_mask	Encoded as: Bit 0 XON / XOFF, input Bit 1 XON / XOFF, output Bit 2 RTR input Bit 3 RTR output Bit 4 RTC input Bit 5 RTC output
xon	Xon character (default DC1, 0x11)

xoff	Xoff character (default DC3, 0x13)
parameter_mask	`parameter_mask` is used for indicating which parameters in the Remote Port Negotiation command is negotiate able. For a command, `parameter_mask` is interpreted as follows: 0 no change 1 change For a response, `parameter_mask` is interpreted as follows: 0 not accepted / not supported 1 accepted proposal - the new values are used The bit mask is shown as: Bit0 bit rate Bit1 data bits Bit2 stop bits Bit3 Parity Bit4 parity type Bit5 XON character Bit6 XOFF character Bit7 Reserved Bit8 XON / XOFF, input Bit9 XON / XOFF, output Bit10 RTR input Bit11 RTR output Bit12 RTC input Bit13 RTC output Bit14 Reserved, set 0 by sender Bit15 Reserved, set 0 by sender

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_PORTNEG_RES signal:

```
void CsrBtSppPortnegResSend(  CsrSchedQid queueId, CsrUint8 serverChannel,
                              RFC_PORTNEG_VALUES_T *portPar );
```

The queue`Id` parameter is used for indicating the destination SPP instance of the signal. The remaining parameters are as described in the table above.

4.10 CSR_BT_SPP_DISCONNECT

Primitives \ Parameters	type	queueId	serverChannel	deviceAddr	reasonCode	reasonSupplier	localTerminated
CSR_BT_SPP_DISCONNECT_REQ	✓		✓				
CSR_BT_SPP_DISCONNECT_IND	✓	✓	✓	✓	✓	✓	✓

Table 10: CSR_BT_SPP_DISCONNECT Primitives

Description

Release a connection between the local and remote device. Depending on the number of connections and other active services, the link may or may not be released.

Parameters

type	Signal identity, CSR_BT_SPP_DISCONNECT_REQ/IND.
queueId	Identifier of the SPP instance in use.
serverChannel	The local server number, which is a reference ID used by the Connection Manager.
deviceAddr	The Bluetooth® device address to which a connection must be/or is established.
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. 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.
reasonSupplier	This parameter specifies the supplier of the reason given in reasonCode. Possible values can be found in csr_bt_result.h
localTerminated	TRUE if termination of connection happened on request from the local host; FALSE otherwise.

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_DISCONNECT_REQ signal:

```
void CsrBtSppDisconnectReqSend(  CsrSchedQid queueId,
                           CsrUint8 theServerChannel );
```

The `queueId` parameter is used for indicating the destination SPP instance of the signal. The remaining parameters are as described in the table above.

4.11 CSR_BT_SPP_GET_INSTANCES_QID

Primitives \ Parameters	type	phandle	phandleListSize	*phandleList
CSR_BT_SPP_GET_INSTANCES_QID_REQ	✓	✓		
CSR_BT_SPP_GET_INSTANCES_QID_CFM	✓		✓	✓

Table 11: CSR_BT_SPP_GET_INSTANCES_QID Primitives

Description

Retrieves list of registered SPP-instances from the SPP-manager.

Parameters

type	Signal identity, CSR_BT_SPP_GET_INSTANCES_QID_REQ/CFM.
phandle	Identifies the queue that the CSR_BT_SPP_GET_INSTANCES_QID_CFM is to be returned to.
phandlesListSize	Number of phandles that is returned in the CSR_BT_SPP_GET_INSTANCES_QID_CFM.
*phandlesList	Pointer to array of phandles for registered SPP instances. These values must be used for distinguishing the registered SPP instances when sending signals to, or receiving signals from them. They are carried in the *queueId* parameter of the signals and library functions.

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_GET_INSTANCES_QID_REQ signal:

```
void CsrBtSppGetInstancesQIDReqSend(CsrSchedQid appHandle);
```

The parameters are as described in the table above.

4.12 CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE

Primitives \ Parameters	type	dataAppHandle	resultCode	resultSupplier
CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE_REQ	✓	✓		
CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE_CFM	✓		✓	✓

Table 12: CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE Primitives

Description

Register special application handle to which all future data-related messages will be sent to instead of the standard application handle. Data related messages are defined as: CSR_BT_SPP_DATA, CSR_BT_SPP_CONTROL, CSR_BT_SPP_PORTNEG.

Parameters

type	Signal identity, CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE_REQ/CFM.
dataAppHandle	Application handle to receive future data messages.
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. All values which are currently not specified in the respective prim.h file 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

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE_REQ signal:

```
void CsrBtSppRegisterDataPathHandleReqSend(    CsrSchedQid sppQueueId,
                                               CsrSchedQid dataHandle);
```

The `sppQueueId` parameter is used for indicating the destination SPP instance of the signal. The remaining parameters are as described in the table above.

4.13 CSR_BT_SPP_DATA_PATH_STATUS

Primitives \ Parameters	type	queueId	status
CSR_BT_SPP_DATA_PATH_STATUS_REQ	✓		✓
CSR_BT_SPP_DATA_PATH_STATUS_IND	✓	✓	✓

Table 13: CSR_BT_SPP_DATA_PATH_STATUS Primitives

Description

When a data path has been registered using the CSR_BT_SPP_REGISTER_DATA_PATH_HANDLE signals (see section 4.12), the data application may need to notify the regular application of changes with regard to data availability. These availability states are “open”, “closed” and “lost”.

The data application request the change to be sent (the REQ), and the SPP then forwards the signal to the regular application (the IND).

Parameters

type	Signal identity, CSR_BT_SPP_DATA_PATH_STATUS_REQ/IND.
queueId	Handle of SPP instance for which the data path is changing status.
status	Data path status code. See csr_bt_profiles.h.

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_DATA_PATH_STATUS_REQ signal:

```
void CsrBtSppDataPathStatusReqSend(CsrSchedQid queueId, CsrUint8 status);
```

The `sppQueueId` parameter is used for indicating the destination SPP instance of the signal. The remaining parameters are as described in the table above.

4.14 CSR_BT_SPP_STATUS

Primitives \ Parameters	type	queueId	serverChannel	deviceAddr	connect	maxMsgSize
CSR_BT_SPP_STATUS_IND	✓	✓	✓	✓	✓	✓

Table 14: CSR_BT_SPP_STATUS Primitive

Description

When connections are established or released, a status indication is sent to the data path, such that a data application may prepare to receive data.

Note that the CSR_BT_SPP_STATUS_IND signal is always sent *after* a CSR_BT_SPP_CONNECT signal and always *before* a CSR_BT_SPP_DISCONNECT signal.

Also note that the CSR_BT_SPP_STATUS_IND signal will be sent only to the data path flow. If a data path handle is not registered, or the data and control handle are equal, the signal is not sent.

Parameters

type	Signal identity, CSR_BT_SPP_STATUS_IND.
queueId	Handle of SPP instance which has just changed connection status.
serverChannel	The local server number, which is a reference ID used by the Connection Manager.
deviceAddr	Bluetooth address of remote device which has been connected (zero otherwise)
connect	True if connection established, false if released.
maxMsgSize	Maximum frame size in a CSR_BT_SPP_DATA_REQ/IND message.

4.15 CSR_BT_SPP_CANCEL_CONNECT

Primitives \ Parameters	type
CSR_BT_SPP _CANCEL_CONNECT_REQ	✓

Table 15: CSR_BT_SPP_CANCEL_CONNECT Primitive

Description

The CSR_BT_SPP_CANCEL_CONNECT_REQ primitive is used for cancelling an ongoing connection establishment. It is used for cancelling the connection attempt right after the CSR_BT_SPP_CONNECT_REQ.

When the connection has been cancelled the SPP profile manager will send a CSR_BT_SPP_CONNECT_IND with resultcode: CSR_BT_CANCELLED_CONNECT_ATTEMPT. If a CSR_BT_SPP_CONNECT_IND has been sent before the connection could be cancelled, the CSR_BT_SPP_CANCEL_CONNECT_REQ will cause the SPP to disconnect the connection to the device and sent a CSR_BT_SPP_DISCONNECT_IND with resultcode CSR_BT_CANCELLED_CONNECT_ATTEMPT.

The response to the CSR_BT_SPP_CANCEL_CONNECT_REQ will be sent to the app handle that initiated the connect attempt.

Parameters

type Signal identity, CSR_BT_SPP_CANCEL_CONNECT.

Library Function

A library function is provided by csr_bt_spp_lib.h, and should be used for building and sending the CSR_BT_SPP_CANCEL_CONNECT_REQ signal:

```
void CsrBtSppCancelConnectReqSend(CsrSchedQid  queueId);
```

The `queueId` parameter is used for indicating the destination SPP instance of the signal.

4.16 CSR_BT_SPP_SECURITY_IN / OUT

Parameters / Primitives	type	appHandle	secLevel	resultCode	resultSupplier
CSR_BT_SPP_SECURITY_IN_REQ	✓	✓	✓		
CSR_BT_SPP_SECURITY_IN_CFM	✓			✓	✓
CSR_BT_SPP_SECURITY_OUT_REQ	✓	✓	✓		
CSR_BT_SPP_SECURITY_OUT_CFM	✓			✓	✓

Table 16: CSR_BT_SPP_SECURITY_IN and CSR_BT_SPP_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_IN_REQ* signal sets up the security level for new incoming connections. Already established or pending connections are not altered.

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_SPP_SECURITY_IN/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. All values which are currently not specified in the respective prim.h file 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	
Bluetooth® Core Specification v.1.1, v.1.2 and v.2.0, section N/A	[BT]
CSR Synergy Bluetooth, SC – Security Controller API Description, section N/A	[SC]
The Bluetooth® Specification, the Serial Port Profile, K:5	[SPP]
CSR Synergy Bluetooth. CM – Connection Manager API Description, doc. no. api-0101-cm, section N/A	[CM]

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
DTE	Data Terminal Equipment
DCE	Data Communication Equipment
SPP	Serial Port Profile
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.