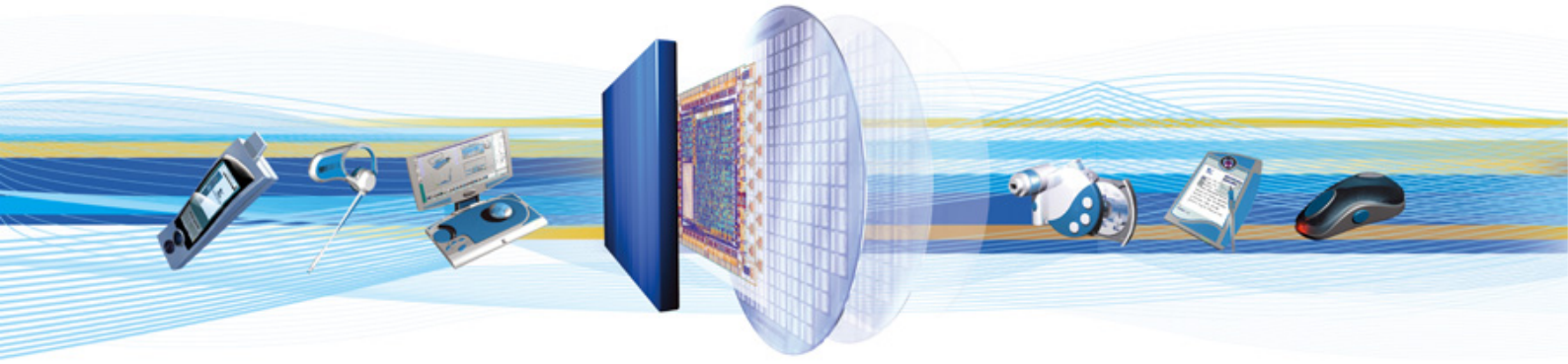
CSR Synergy Bluetooth 18.2.0

DUNC – Dial-Up Networking Data Terminal

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 Dial-up Networking Profile (DUN) Data Terminal (DT) side (DUN-DT). The DUN-DT conforms to the DT side of the general DUN specification given in [DUNSPEC].

1.2 Assumptions

The following assumptions and preconditions are made in the following:

- The communication transport between the DUN-DT profile and the application is believed to be reliable
- Bonding (pairing) is not handled by the DUN-DT profile

It is assumed that the reader has basic knowledge about Bluetooth® and the Dial-up Networking Profile [DUNSPEC]. Furthermore, it is assumed that the reader has basic understanding of the syntax and interpretation of Message Sequence Charts (MSC) and state diagrams, as these diagrams will be used throughout the description of the DUN-DT interface.

2 Description

This section gives an overview of the functionality and architecture of the DUN-DT.

2.1 Introduction

The DUN profile provides the functionality necessary to implement the Internet Bridge usage model using Bluetooth®. The following two scenarios most commonly cover the Internet Bridge usage model:

- Usage of a cellular phone or modem by a computer as a wireless modem for connecting to a dial-up internet access server, or using other dial-up services
- Usage of a cellular phone or modem by a computer to receive data calls

A typical example of the usage of the DUN profile is a laptop connecting to the Internet wireless through a cellular phone using Bluetooth®. In the given example the laptop will have the Dial-up Networking Profile Data Terminal role. Most typical devices acting as DUN-DTs are laptops and desktop PCs.

The DUN-DT profile supplies functionality for:

- Connection Management (establishment and termination)
- Data Transferral (i.e. AT commands and data)
- Control of the emulated serial port
- Division of the message flow into a Bluetooth® management path and a DUN data path
- Prepared for multiple instances of the profile
- A simple application control of the low power mode

2.2 Reference Model

The DUN-DT interfaces with the Connection Manager (CM), which handles the connection established to the DUN Gateway, as depicted in Figure 1.

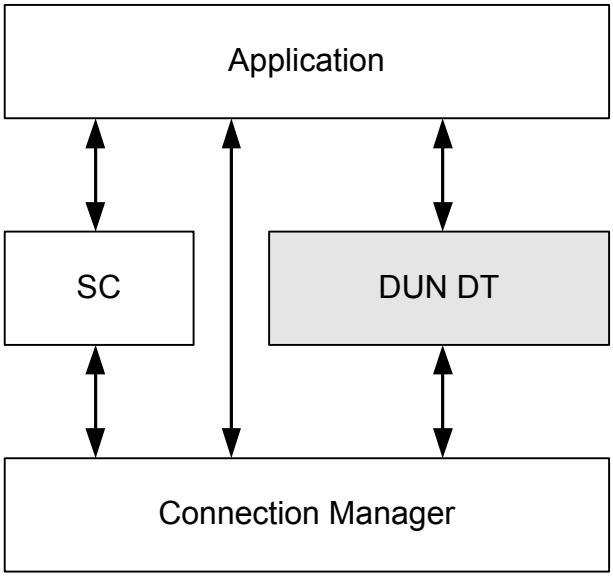


Figure 1: Reference model

The application that wants to utilise the DUN-DT functionality provided by CSR Synergy Bluetooth interfaces the DUN-DT profile in order to establish a connection to the DUN Gateway with the purpose of gaining dialup network access. Furthermore, the application must interface the Security Controller (SC) and the CM directly, because no functionality for pairing and discovery is implemented in DUN-DT. The SC is utilized during pairing with the DUN-GW Bluetooth® device, whereas the CM is interfaced when making a discovery for the DUN-GW Bluetooth® device. The API of the CM and SC is described in [CM] and [SC], respectively.

2.3 Communication Flow Architecture

The DUN-DT profile implementation divides the communication flow into two parts; a Bluetooth® management flow and a data management flow. This communication flow architecture is shown in Figure 2.

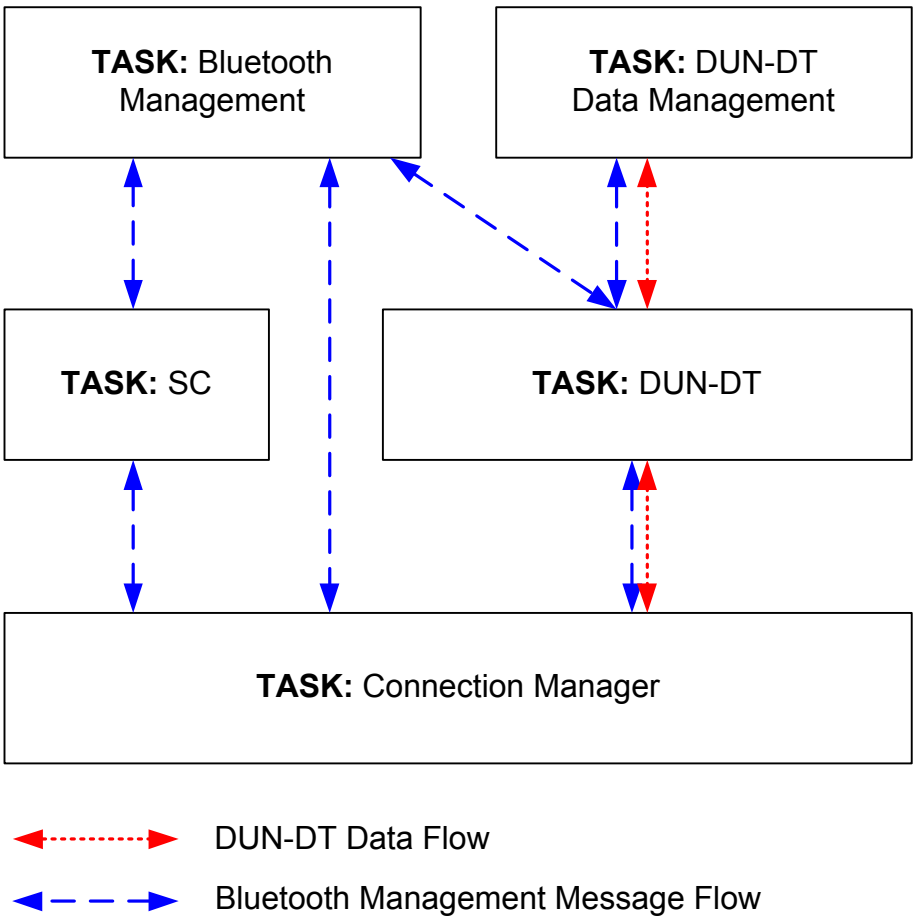


Figure 2: Communication Flow Architecture

The blue arrows depict the communication flow needed for managing the Bluetooth® connection, whereas the red arrows depict the communication flow to be considered as DUN-DT data. This architecture implies that two application tasks must be registered in DUN-DT when using its functionality[1]. The idea of supporting a Bluetooth® management flow and a DUN-DT data flow enables more flexibility in the implementation of the application layer above the DUN-DT profile.

Registration of the *Bluetooth Management* task is done when requesting a connection establishment (described in 3.1). The *Bluetooth Management* task will receive all messages (both management and data path messages), except the CSR_BT_DUNC_STATUS_IND (see 4.10 for description), until a *DUN-DT Data Management* task is registered (described in 3.3). When the *DUN-DT Data Management* task is registered, all messages related to

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

the DUN-DT data will be forwarded to the *DUN-DT 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 appearing as a serial port device in the Operating System 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 *DUN-DT Data Management* message flow.

2.4 State Sequence Overview

A High Level Message Sequence Chart (HMSC) showing the basic functionality of the DUN-DT profile is depicted in Figure 3.

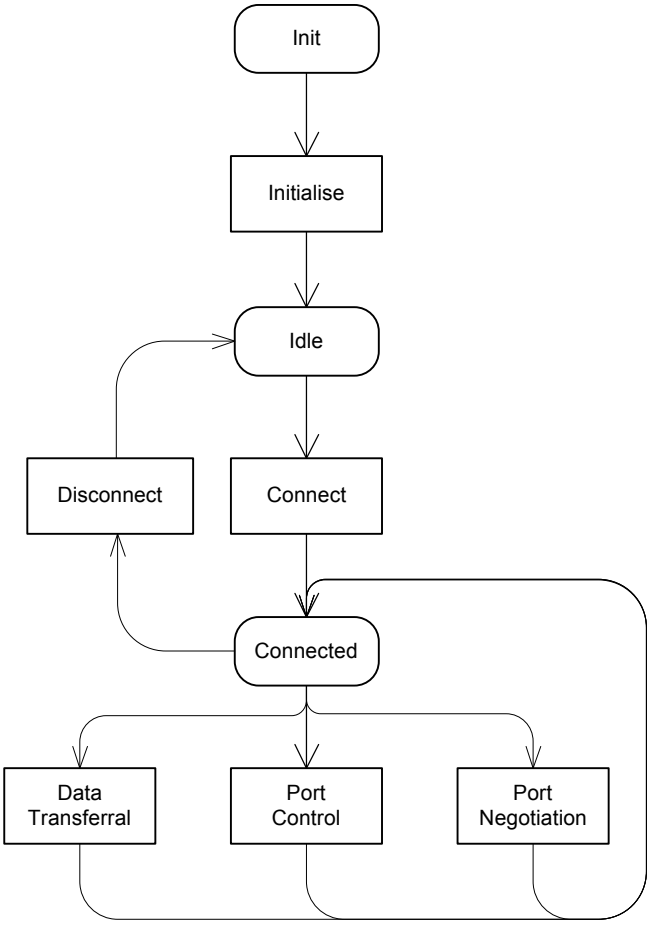


Figure 3: HMSC showing the states of the DUN-DT and the actions available in each state

During start up of CSR Synergy Bluetooth, the DUN-DT profile will be initialised and go to *Idle* state. In this state, the profile is ready to be used and receives messages from the application layer. If the application layer requests a connection, the profile will go into a *Connect* phase. When connection establishment is completed, the profile enters *Connected* state. In this state data can be transferred (AT commands, PPP data, etc.) and the virtual serial port can be controlled. Finally, the application can issue a request for disconnection of the Bluetooth® link, which will cause the profile to enter *Idle* state again awaiting another connect request.

3 Interface Description

In this section, a series of MSCs will be presented to explain the usage of the available primitives in the DUN-DT profile. The primitives presented in this section will be described further in section 4 where more details of the parameters of each message are available. The MSCs used throughout this chapter depict both a *DUN-DT Data App* and a *BT Control App*, which illustrate the concept of having both a Bluetooth® control and data path.

NOTE: In the following MSCs in this chapter, a *duncInstanceId* parameter is included in all messages going from the DUN-DT profile to the application. The *duncInstanceId* is necessary because DUN-DT is prepared for multiple instances, and therefore it is necessary to inform the application layer about what DUN-DT instance it has received a message from. To avoid triviality and because this parameter is included in all up-going messages, it will not be described for each MSC.

3.1 Connect

Before being able to start a dialup networking session a Bluetooth® connection to the DUN-GW must be established.

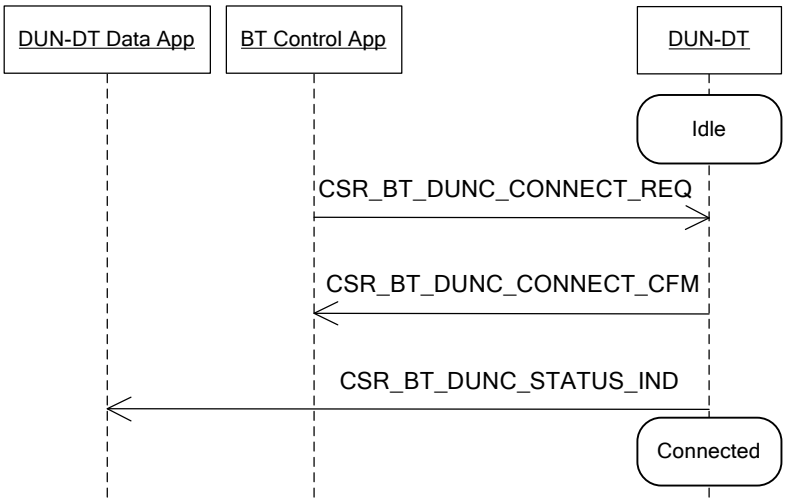


Figure 4: DUN connection establishment

A connection establishment is always initiated by the application at the client side and the CSR_BT_DUNC_CONNECT_REQ and DUNC CONNECT_CFM primitives are used during this phase. Initially, the application sends a CSR_BT_DUNC_CONNECT_REQ containing the Bluetooth® device address of the server to connect to and the desirable low power mode for the profile to use. Currently sniff and active mode is supported.

When the RFCOMM connection is established successfully, a CSR_BT_DUNC_CONNECT_CFM is sent to the *BT Control App,* which was the application that issued the CSR_BT_DUNC_CONNECT_REQ. The result of the connection establishment is indicated in the *resultCode* and *resultSupplier* parameters of the confirm message. If the connection is successful, *resultCode* will equal CSR_BT_RESULT_CODE_DUNC_SUCCESS and *resultSupplier* will equal CSR_BT_SUPPLIER_DUNC.

Possible values for *resultSupplier* are found in csr_bt_result.h. If e.g. the *resultSupplier* equals CSR_BT_SUPPLIER_CM, the possible result codes are found in csr_bt_cm_prim.h. All values which are currently not specified in the relevant prim.h file are regarded as reserved and the application should consider them as errors.

Furthermore, the CSR_BT_DUNC_CONNECT_CFM contains a *maxMsgSize* parameter, which is the maximum message size[2] that the profile is capable of receiving from the application layer in a CSR_BT_DUNC_DATA_REQ.

If the *result* parameter in CSR_BT_DUNC_CONNECT_CFM is CSR_BT_SUCCESS, a CSR_BT_DUNC_STATUS_IND is sent to the *DUN_DT Data App* informing that the connection to the gateway is established. The CSR_BT_DUNC_STATUS_IND is described in 3.9 and 4.10.

3.2 Cancel Connect

The DUN-DT profile offers functionality for cancellation of an already ongoing connection establishment. This is done by sending a CSR_BT_DUNC_CANCEL_CONNECT_REQ from the application to the profile. The scenario is illustrated below.

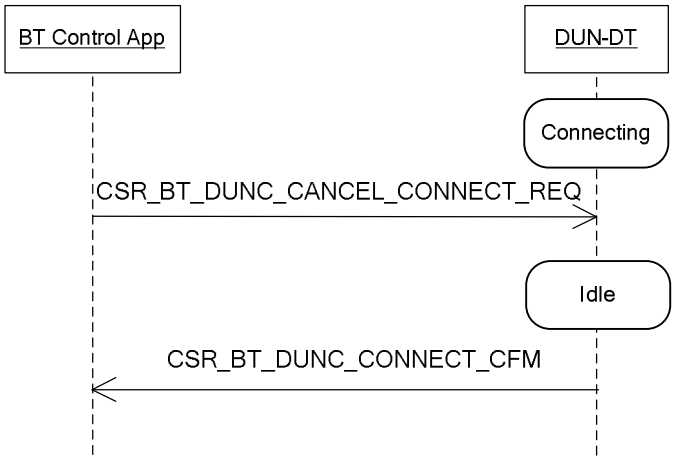


Figure 5: Cancel the establishment of a connection

In response to the request, the DUN-DT profile will immediately cancel the ongoing connection establishment. When the process is finished, the DUN-DT will issue a CSR_BT_DUNC_CONNECT_CFM to the *BT Control App* with a result code different from “CSR_BT_RESULT_CODE_DUNC_SUCCESS” and/or a result supplier different from CSR_BT_SUPPLIER_DUNC. When the application receives the CSR_BT_DUNC_CONNECT_CFM, it can immediately try to connect again. The cancel operation can be useful to save time if a CSR_BT_DUNC_CONNECT_REQ is sent with a wrong Bluetooth® device address of the DUN gateway, because then the application will not have to wait for the search procedure to timeout.

3.3 Register Data Path Handle

As described in 2.3, the DUN-DT profile provides functionality for dividing the message flow into a Bluetooth® control path and a DUN-DT data path. As default, all messages are sent to the application handle, which was used as *ctrlHandle* parameter in the CSR_BT_DUNC_CONNECT_REQ (see 4.2). For registration of the separate data path application handle, the CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_REQ and CSR_BT_DUNC_REGISTER_DATA_PATH_CFM primitives must be used. The usage of these primitives is illustrated in the figure below.

[2] The maximum message size in CSR_BT_DUNC_CONNECT_CFM corresponds to the maximum RFCOMM message size that the two Bluetooth® devices have agreed upon during RFCOMM connection establishment.

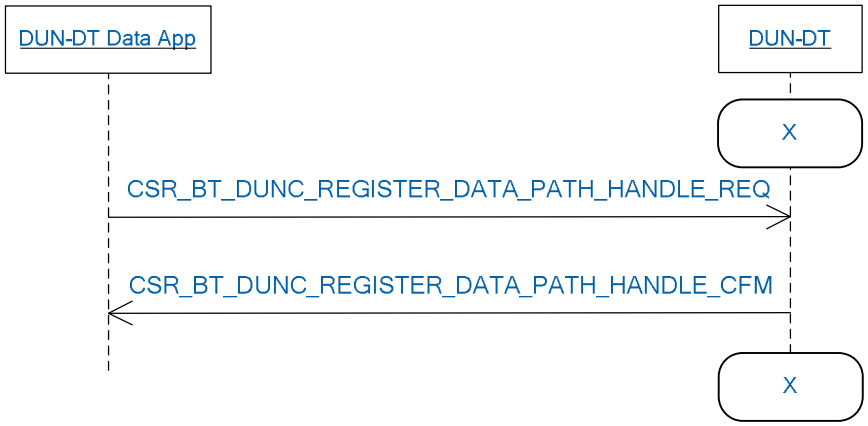


Figure 6: Registering the data path application handle

The *DUN-DT Data App* must send a CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_REQ to the DUN-DT profile whenever it wants to register a separate data path. The CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_REQ contains the application handle of the *DUN-DT Data App*, and will be used by the DUN-DT profile whenever it shall deliver messages related to the data path. When the registration is completed, the profile responses with a CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_CFM. Currently, the *resultCode* of this message will always equal CSR_BT_RESULT_CODE_DUNC_SUCCESS and the *resultSupplier* will always equal CSR_BT_SUPPLIER_DUNC.

3.4 Data Path Status Change

The *BT Control App* must be informed about changes in the status of the *DUN-DT Data App*. For this purpose the CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ and CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_IND primitives are used. Two scenarios can result in a change of the data path status:

- The *DUN-DT Data App* informs if the device driver[3] has been opened (someone uses the driver) or closed (no one uses the driver)
- The *DUN-DT Data App* has been terminated, and the *BT Control App* must be informed about it

The first scenario is depicted in Figure 7.

[3] As mentioned in 2.3, the philosophy of having two message paths is primarily to enable the possibility for easy implementation of a Dialup Networking device driver appearing as a serial port in the OS, which will make the Bluetooth connection transparent for the user.

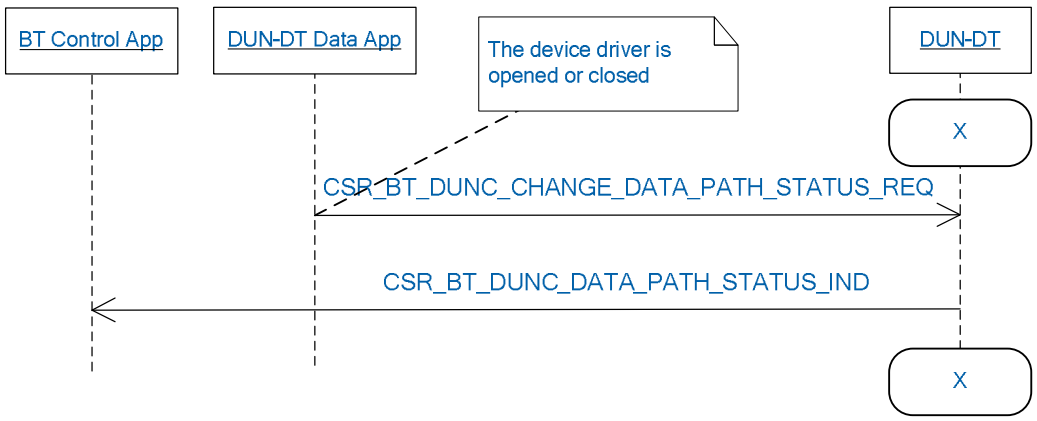


Figure 7: DUN-DT Data App informs the BT Control App about a change in its status

As the figure depicts, the *DUN-DT Data App* informs the *BT Control App* about a change in its status by sending a CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ message to the DUN-DT profile. The CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ contains a *status* parameter, which must be set to an appropriate value; CSR_BT_DATA_PATH_STATUS_OPEN if the device driver is opened and CSR_BT_DATA_PATH_STATUS_CLOSED if the device driver is closed. The values for *status* are defined in csr_bt_profiles.h. When the DUN-DT profile receives the CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ, it sends a CSR_BT_DUNC_DATA_PATH_STATUS_IND to the *BT Control App* containing a *status* parameter equals to the *status* of CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ.

The second scenario, with the *DUN-DT Data App* being terminated, is shown in Figure 8.

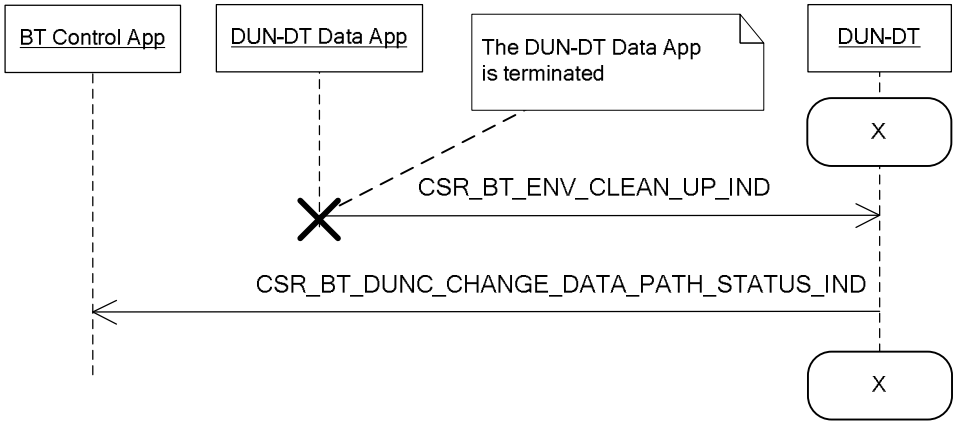


Figure 8: Termination of DUN-DT Data App results in a CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_IND

In the above scenario the *DUN-DT Data App* is terminated, hence a CSR_ENV_CLEANUP_IND[4] must be broadcasted to all the tasks running in the scheduler informing about its termination. When DUN-DT receives the CSR_ENV_CLEANUP_IND it sends a CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_IND with the *status* parameter set to CSR_BT_DATA_PATH_STATUS_LOST. This indicates that the *DUN-DT Data App* is no longer valid, and therefore messages for the data path will be sent to the *BT Control App*, as was the case before the *DUN-DT Data App* was registered.

[4] The CSR_ENV_CLEANUP_IND is described in [SCHED].

3.5 Disconnect

If a disconnection of the Bluetooth® connection is desirable, the CSR_BT_DUNC_DISCONNECT_REQ and CSR_BT_DUNC_DISCONNECT_IND primitives must be used. The usage model of the two primitives is depicted below.

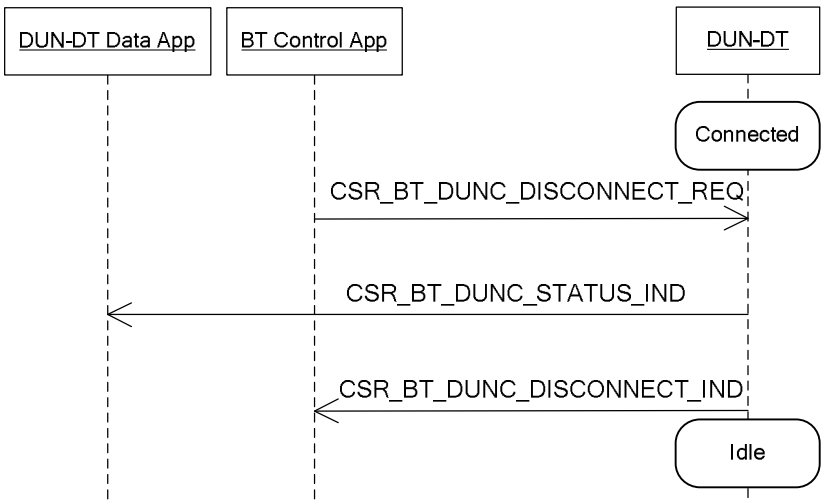


Figure 9: Client side initiated disconnect

The *BT Control App* requests a disconnect by sending a CSR_BT_DUNC_DISCONNECT_REQ message to the profile. When a disconnect is requested, the DUN-DT profile sends a CSR_BT_DUNC_STATUS_IND to the *DUN.DT Data App* informing that the connection is closed. DUN-DT responses with a CSR_BT_DUNC_DISCONNECT_IND informing about the reason for the disconnection in the *reasonCode* and *reasonSupplier* parameters.

The *reasonSupplier* specifies the supplier of the reason given in *reasonCode*. Possible values for *reasonSupplier* are found in csr_bt_result.h. If e.g. the *reasonSupplier* equals CSR_BT_SUPPLIER_CM, the possible reason codes are found in csr_bt_cm_prim.h. All values which are currently not specified in the relevant prim.h file are regarded as reserved and the application should consider them as errors.

Note: The CSR_BT_DUNC_DISCONNECT_REQ will only disconnect the Bluetooth® connection, whereas the Dialup Networking (to the internet, etc.) connection must be terminated manually by the application.

3.6 Data Transferal

Transferral of data between the DUN-DT and the DUN gateway is divided into two scenarios; data transfer initiated by the DUN-DT and data transfer initiated by the DUN gateway. Figure 10 shows the scenario of data transfer initiated by the DUN-DT.

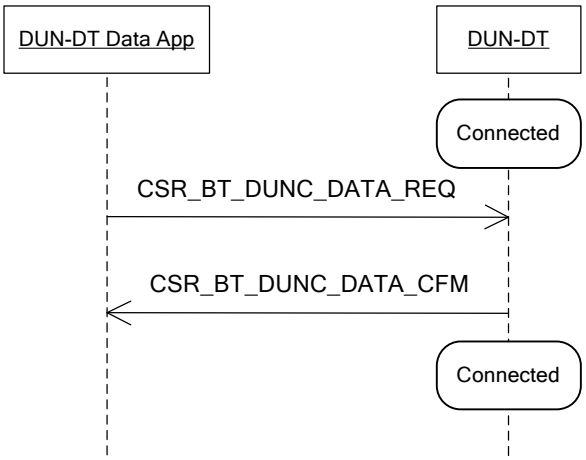


Figure 10: Data transferal send from the DUN-DT application

Data transferral initiated by the *DUN-DT Data App* implies the use of the CSR_BT_DUNC_DATA_REQ and CSR_BT_DUNC_DATA_CFM primitives. The CSR_BT_DUNC_DATA_REQ shall be used by the application whenever it wants to send data to the DUN-GW. The CSR_BT_DUNC_DATA_REQ primitive simply contains a pointer to the data being sent and a length of the data.

NOTE: The length of the data in the CSR_BT_DUNC_DATA_REQ must never exceed the *maxMsgSize* parameter in the CSR_BT_DUNC_CONNECT_CFM (see 4.2) and CSR_BT_DUNC_STATUS_IND (see 4.10).

DUN-DT responses with a CSR_BT_DUNC_DATA_CFM indicating that the data has been received and that the profile is ready to receive yet another CSR_BT_DUNC_DATA_REQ. The CSR_BT_DUNC_DATA_REQ/CFM primitives are also used as flow control between the application and the DUN-DT profile.

NOTE: The application is never allowed to have more than one outstanding CSR_BT_DUNC_DATA_REQ at any time.

The scenario for data transferral initiated by the DUN gateway is illustrated in Figure 11.

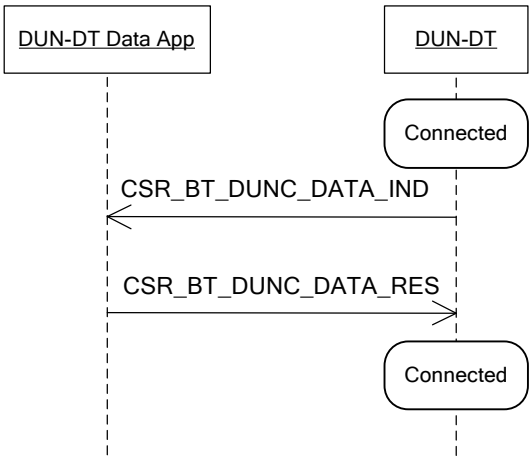


Figure 11: Data transferal initiated by the DUN-GW application

From the *DUN-DT Data App's* point of view data sent from the DUN-GW will appear as a CSR_BT_DUNC_DATA_IND message received from DUN-DT. The CSR_BT_DUNC_DATA_IND contains a pointer to the data and a length of the data.

NOTE: The data received in the CSR_BT_DUNC_DATA_IND must be freed by the *DUN-DT Data App*. The data allocated and sent using CSR_BT_DUNC_DATA_REQ is freed in CSR Synergy Bluetooth.

When the application has processed the data, and is ready to receive yet another CSR_BT_DUNC_DATA_IND from the DUN-DT profile, it must send a CSR_BT_DUNC_DATA_RES to the profile. The application shall always send a CSR_BT_DUNC_DATA_RES when it is ready to receive more data, otherwise it will never receive more data from the profile.

3.7 Port Negotiation

If the remote side requests remote port negotiation this is indicated to the application layer in a CSR_BT_DUNC_PORTNEG_IND signal. This signal can be received at any time, even before a connection request is confirmed. The scenario is shown in Figure 12.

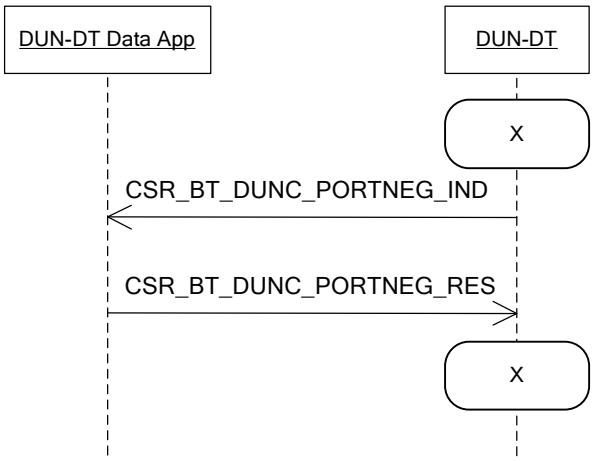


Figure 12: DUN-GW initiated port negotiation request

The remote side can either indicate its own current values or request values from the local side. The CSR_BT_DUNC_PORTNEG_IND primitive contains a port negotiation parameter, which will be described in 4.8.

The *DUN-DT Data App* is also capable of requesting port negotiation. This scenario is shown in Figure 13.

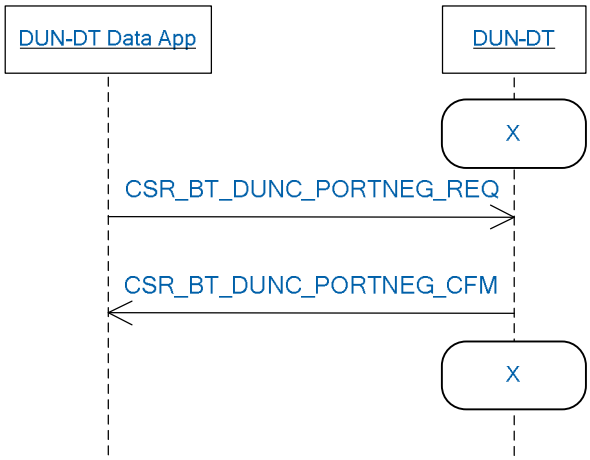


Figure 13: DUN-DT initiated port negotiation request

The CSR_BT_DUNC_PORTNEG_REQ primitive contains a port negotiation parameter, which will be described in 4.8. The DUN-DT profile will response with a CSR_BT_DUNC_PORTNEG_CFM.

NOTE: Currently, the CSR_BT_DUNC_PORTNEG_REQ/CFM is not yet implemented in CSR Synergy Bluetooth, but the DUN-DT profile is prepared for it. Hence, no port negotiation will be initiated when sending a CSR_BT_DUNC_PORTNEG_REQ to the profile.

3.8 Port Control

The local modem line status can be transferred to the remote side, which is depicted in Figure 14.

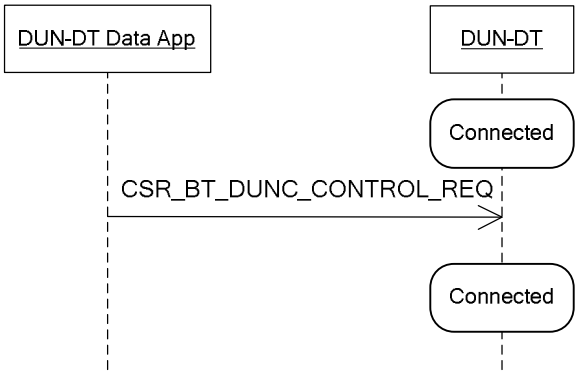


Figure 14: The local modem status is send to remote side

The CSR_BT_DUNC_CONTROL_REQ contains information about the modem status and break signal. The CSR_BT_DUNC_CONTROL_REQ is not confirmed by the DUN-DT profile.

Figure 15 shows how the *DUN-DT Data App* can receive an indication of the status of the remote modem status.

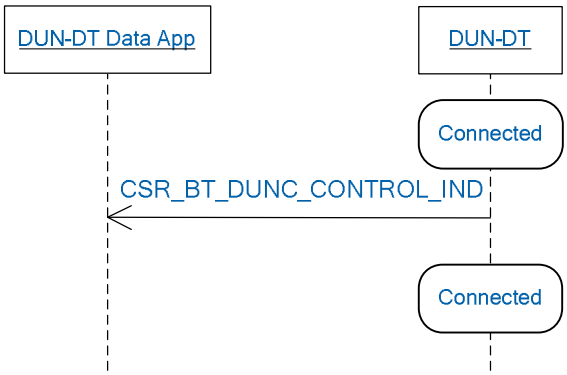


Figure 15: Indication of the remote modem status

NOTE: After a CSR_BT_DUNC_STATUS_IND is received with TRUE in the *connected* parameter, the *DUN-DT Date App* is required to send information on all changes in RS232 control signals with the modem status command, i.e. the CSR_BT_DUNC_CONTROL_REQ.

3.9 Connection Status

In order to inform the *DUN-DT Data App* whether it is connected to the DUN-GW, the CSR_BT_DUNC_STATUS_IND primitive is used. The scenario is depicted in the figure below.

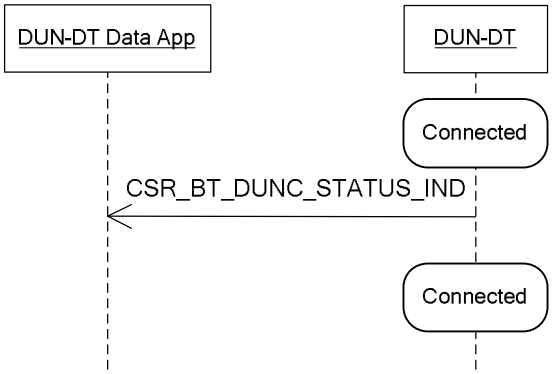


Figure 16: The status of the connection is sent to the DUN-DT Data App

The CSR_BT_DUNC_STATUS_IND is sent to the *DUN-DT Data App* every time changes in the Bluetooth® connection occurs. The status of the connection is indicated in the *connected* parameter. The parameter is TRUE if the Bluetooth® connection is established, whereas it is FALSE if the connection is down. *DUN-DT Data App* is only allowed to send data if it has received a CSR_BT_DUNC_STATUS_IND with *connected* set to TRUE. If *connected* is set to TRUE, the *maxMsgSize* parameter is set to the maximum data size that must be sent in one CSR_BT_DUNC_DATA_REQ. If the *connected* parameter is FALSE the *maxMsgSize* shall be ignored.

4 DUN 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_dunc_prim.h file.

4.1 List of All Primitives

Primitives:	Reference:
CSR_BT_DUNC_CONNECT_REQ	See section 4.2
CSR_BT_DUNC_CONNECT_CFM	See section 4.2
CSR_BT_DUNC_CANCEL_CONNECT_REQ	See section 4.3
CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_REQ	See section 4.4
CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_CFM	See section 4.4
CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ	See section 4.5
CSR_BT_DUNC_DATA_PATH_STATUS_IND	See section 4.5
CSR_BT_DUNC_DATA_REG	See section 4.6
CSR_BT_DUNC_DATA_RES	See section 4.6
CSR_BT_DUNC_DATA_IND	See section 4.6
CSR_BT_DUNC_DATA_CFM	See section 4.6
CSR_BT_DUNC_CONTROL_REQ	See section 4.7
CSR_BT_DUNC_CONTROL_IND	See section 4.7
CSR_BT_DUNC_PORTNEG_IND	See section 4.8
CSR_BT_DUNC_PORTNEG_RES	See section 4.8
CSR_BT_DUNC_PORTNEQ_REQ	See section 4.8
CSR_BT_DUNC_PORTNEQ_CFM	See section 4.8
CSR_BT_DUNC_DISCONNECT_REQ	See section 4.9
CSR_BT_DUNC_DISCONNECT_CFM	See section 4.9
CSR_BT_DUNC_STATUS_IND	See section 4.10
CSR_BT_DUNC_SECURITY_OUT_REQ	See section 4.11
CSR_BT_DUNC_SECURITY_OUT_CFM	See section 4.11

Table 1: List of all primitives

4.2 CSR_BT_DUNC_CONNECT

Primitives \ Parameters	type	ctrlHandle	bdAddr	lowPowerSupport	resultCode	resultSupplier	maxMsgSize	btConnId
CSR_BT_DUNC_CONNECT_REQ	✓	✓	✓	✓				
CSR_BT_DUNC_CONNECT_CFM	✓				✓	✓	✓	✓

Table 2: CSR_BT_DUNC_CONNECT Primitives

Description

To start a DUN session against the DUN-GW, the application sends a CSR_BT_DUNC_CONNECT_REQ with the Bluetooth® address of the gateway and the desirable low power mode. The DUN-DT profile responds with a CSR_BT_DUNC_CONNECT_CFM indicating the result of the connection establishment and the maximum message size that the profile will accept from the application.

Parameters

type	Signal identity, CSR_BT_DUNC_CONNECT_REQ/CFM.
ctrlHandle	The identity of the calling process. It is possible to initiate the connect process by any higher layer process as the responses to all requests will be sent to the *ctrlHandle*. However, if a *dataHandle* is registered, the responses and indications related to the data path will not be sent to the *ctrlHandle* but to the *dataHandle* instead.
bdAddr	The Bluetooth® address of the gateway to connect to.
lowPowerSupport	Is used by the application to inform the profile whether it should support sniff mode or only active mode. If only active mode shall be supported FALSE must be used and use TRUE for sniff mode support.
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
maxMsgSize	The maximum packet size that the DUN-DT profile can accept from the application layer. The application layer must fragment messages larger than this size.
btConnId	Identifier used when moving the connection to another AMP controller, i.e. when calling the `CsrBtAmpmMoveReqSend`-function.

Library Function

The following library function is provided by CSR Synergy Bluetooth for DUN-DT, and can be used for creating the proper message and sending it to the DUN-DT profile. The library functions are defined in csr_bt_dunc_lib.h.

```
void CsrBtDuncConnectReqSend( CsrSchedQid   duncInstanceId,
                              CsrSchedQid   ctrlHandle,
                              deviceAddr_t  bdAddr,
                              CsrBool       lowPowerSupport    )
```

duncInstanceId	The queue of the DUN-DT instance to send the message to.
ctrlHandle	The same as in table above.
bdAddr	The same as in table above.
lowPowerSupport	The same as in table above.

4.3 CSR_BT_DUNC_CANCEL_CONNECT

Primitives \ Parameters	type
CSR_BT_DUNC_CANCEL_CONNECT_REQ	✓

Table 3: CSR_BT_DUNC_CANCEL_CONNECT Primitive

Description

An ongoing CSR_BT_DUNC_CONNECT_REQ can be cancelled using the CSR_BT_DUNC_CANCEL_CONNECT_REQ. In response to the request a CSR_BT_DUNC_CONNECT_CFM is sent to the application registered as the *ctrlHandle* parameter in the CSR_BT_DUNC_CONNECT_REQ message.

NOTE: The *ctrlHandle* parameter is not set until a CSR_BT_DUNC_CONNECT_REQ has been issued. Therefore a CSR_BT_DUNC_CANCEL_CONNECT_REQ will not be confirmed with a CSR_BT_DUNC_CONNECT_CFM unless a CSR_BT_DUNC_CONNECT_REQ has been issued.

Parameters

type Signal identity, CSR_BT_DUNC_CANCEL_CONNECT_REQ.

Library Function

The following library function is provided by CSR Synergy Bluetooth for DUN-DT, and can be used for creating the proper message and sending it to the DUN-DT profile. The library functions are defined in csr_bt_dunc_lib.h.

```
void CsrBtDuncCancelConnectReqSend(  CsrSchedQid  duncInstanceId )
```

duncInstanceId The queue of the DUN-DT instance to send the message to.

4.4 CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE

Primitives \ Parameters	type	dataAppHandle	resultCode	resultSupplier
CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_REQ	✓	✓		
CSR_BT_DUNC_ REGISTER_DATA_PATH_HANDLE _CFM	✓		✓	✓

Table 4: CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE Primitives

Description

For the *DUN-DT Data App* to register its application handle in the DUN-DT profile and get the profile to send data path related messages to the *DUN-DT Data App*, it must send a CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_REQ to the profile. The profile will register the application handle given as the *dataAppHandle* parameter in the request, and then response with a CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_CFM. Hereafter, all messages related to the data path will be sent to the *DUN-DT Data App*.

Parameters

type	Signal identity, CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_REQ/CFM.
dataAppHandle	The handle (task queue id) of the application to receive messages related to the data path in the future.
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

The following library function is provided by CSR Synergy Bluetooth for DUN-DT, and can be used for creating the proper message and sending it to the DUN-DT profile. The library functions are defined in csr_bt_dunc_lib.h.

```
void CsrBtDuncRegisterDataPathHandleReqSend(  CsrSchedQid  duncInstanceId,
                                              CsrSchedQid  dataAppHandle )
```

duncInstanceId	The queue of the DUN-DT instance to send the message to.
dataAppHandle	The same as in table above.

4.5 CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS / CSR_BT_DUNC_DATA_PATH_STATUS

Primitives \ Parameters	type	duncInstanceId	status	resultCode	resultSupplier
CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ	✓		✓		
CSR_BT_DUNC_DATA_PATH_STATUS_IND	✓	✓	✓	✓	✓

Table 5: CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS / CSR_BT_DUNC_DATA_PATH_STATUS Primitives

Description

The *DUN-DT Data App* can inform the *BT Control App* about changes in its state by sending a CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ to the DUN-DT profile. The profile will upon reception of the primitive send a CSR_BT_DUNC_DATA_PATH_STATUS_IND to the *BT Control App* indicating the state of the *DUN-DT Data App.*

Parameters

type	Signal identity, CSR_BT_DUNC_CHANGE_DATA_PATH_STATUS_REQ / CSR_BT_DUNC_DATA_PATH_STATUS_IND
duncInstanceId	The handle of the DUNC profile instance that sends the message.
status	The status of the *DUN-DT Data App*. **CSR_BT_DATA_PATH_STATUS_OPEN:** The data path device driver was opened. **CSR_BT_DATA_PATH_STATUS_CLOSED:** The data path device driver was closed. **CSR_BT_DATA_PATH_STATUS_LOST:** The data path was lost, i.e. no device driver is using utilizing the data path. All status codes are defined in csr_bt_profiles.h.
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

The following library function is provided by CSR Synergy Bluetoothfor DUN-DT, and can be used for creating the proper message and sending it to the DUN-DT profile. The library functions are defined in csr_bt_dunc_lib.h.

```
void CsrBtDuncChangeDataPathStatusReqSend( CsrSchedQid  duncInstanceId,
                                           CsrUint8     status )
```

duncInstanceId	The queue of the DUN-DT instance to send the message to.
status	The same as in table above.

4.6 CSR_BT_DUNC_DATA

Primitives \ Parameters	type	dataLength	*data
CSR_BT_DUNC_DATA_REQ	✓	✓	✓
CSR_BT_DUNC_DATA_RES	✓		
CSR_BT_DUNC_DATA_IND	✓	✓	✓
CSR_BT_DUNC_DATA_CFM	✓		

Table 6: CSR_BT_DUNC_DATA Primitives

Description

The application on the DUN-DT side can send Dialup Networking data to the DUN-GW by sending a CSR_BT_DUNC_DATA_REQ message to the DUN-DT profile containing the data to send and its length. The profile will respond with a CSR_BT_DUNC_DATA_CFM when the data is sent, and the profile is ready to receive yet another CSR_BT_DUNC_DATA_REQ.

The application at the DUN-DT side receives data from the DUN-GW by means of a CSR_BT_DUNC_DATA_IND. The CSR_BT_DUNC_DATA_IND contains the data to be received and its length. The data must be released by the application, and when the received data has been processed and the application is ready to receive another CSR_BT_DUNC_DATA_IND it must send a CSR_BT_DUNC_DATA_RES to the DUN-DT profile.

Parameters

type	Signal identity, CSR_BT_DUNC_DATA_REQ/RES/IND/CFM.
dataLength	The length of the data received (CSR_BT_DUNC_DATA_IND) or the data to be sent (CSR_BT_DUNC_DATA_REQ).
*data	A pointer to the memory containing the data to be sent or received. For receiving data, the *data* pointer in the CSR_BT_DUNC_DATA_IND shall be released by the application.

Library Functions

The following library functions are provided by CSR Synergy Bluetooth for DUN-DT, and can be used for creating the proper message and sending it to the DUN-DT profile. The library functions are defined in csr_bt_dunc_lib.h.

```
void CsrBtDuncDataReqSend   ( CsrSchedQid    duncInstanceId,
                              CsrUint8      *data,
                              CsrUint16     dataLength )
```

duncInstanceId	The queue of the DUN-DT instance to send the message to.
data	The same as in table above.
dataLength	The same as in table above.

```
void CsrBtDuncDataResSend   (   CsrSchedQid      duncInstanceDataId   )
```

duncInstanceId	The queue of the DUN-DT instance to send the message to.

4.7 CSR_BT_DUNC_CONTROL

Primitives \ Parameters	type	modemstatus	breakSignal
CSR_BT_DUNC_CONTROL_REQ	✓	✓	✓
CSR_BT_DUNC_CONTROL_IND	✓	✓	✓

Table 7: CSR_BT_DUNC_CONTROL Primitives

Description

Send and receive modem status information.

Parameters

type — Signal identity, CSR_BT_DUNC_CONTROL_REQ/IND.

modemStatus — The *modemStatus* 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.

breakSignal — The *breakSignal* 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

The following library function is provided by CSR Synergy Bluetooth for DUN-DT, and can be used for creating the proper message and sending it to the DUN-DT profile. The library functions are defined in csr_bt_dunc_lib.h.

```
void CsrBtDuncControlReqSend( CsrSchedQid      duncInstanceId,
                              CsrUint8         modemStatus,
                              CsrUint16        breakSignal )
```

duncInstanceId — The queue of the DUN-DT instance to send the message to.

modemStatus — The same as in table above.

breakSignal — The same as in table above.

4.8 CSR_BT_DUNC_PORTNEG

Primitives \ Parameters	type	portPar	request
CSR_BT_DUNC_PORTNEG_IND	✓	✓	✓
CSR_BT_DUNC_PORTNEG_RES	✓	✓	
CSR_BT_DUNC_PORTNEG_REQ	✓	✓	✓
CSR_BT_DUNC_PORTNEG_CFM	✓		

Table 8: CSR_BT_DUNC_PORTNEG Primitives

Description

Send and receive port set-up information. If the remote device changes the port settings or requests the current settings, the DUN-DT will send a CSR_BT_DUNC_PORTNEG_IND to the application. The application shall then answer with a CSR_BT_DUNC_PORTNEG_RES. If the application wants to change the port settings after a connection has been established, it shall use the CSR_BT_DUNC_PORTNEG_REQ. Once the operation is performed it will receive a CSR_BT_SPP_PORTNEG_CFM with the outcome. If the CSR_BT_DUNC_PORTNEG_REQ is issued while no connection is present, the CSR_BT_DUNC_PORTNEG_CFM will not be issued.

Parameters

type	Signal identity, CSR_BT_DUNC_PORTNEG_IND/RES/REQ/CFM.
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	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. If the *request* is FALSE, the remote device must confirm the settings or propose new ones. Setting new suggested values also includes setting the proper parameter mask.

```
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 Functions

The following library functions are provided by CSR Synergy Bluetoothfor DUN-DT, and can be used for creating the proper message and sending it to the DUN-DT profile. The library functions are defined in csr_bt_dunc_lib.h.

```
void CsrBtDuncPortnegResSend(  CsrSchedQid duncInstanceId,
                               RFC_PORTNEG_VALUES_T    * portPar )
```

duncInstanceId	The queue of the DUN-DT instance to send the message to.
portPar	The same as in table above.

```
void CsrBtDuncPortneqReqSend(              CsrSchedQid     duncInstanceId,
                                           RFC_PORTNEG_VALUES_T    * portPar,
                                           CsrBool   request )
```

duncInstanceId	The queue of the DUN-DT instance to send the message to.
portPar	The same as in table above.
request	The same as in table above.

4.9 CSR_BT_DUNC_DISCONNECT

Primitives \ Parameters	type	duncInstanceId	localTerminated	reasonCode	reasonSupplier
CSR_BT_DUNC_DISCONNECT_REQ	✓				
CSR_BT_DUNC_DISCONNECT_IND	✓	✓	✓	✓	✓

Table 9: CSR_BT_DUNC_DISCONNECT Primitives

Description

The application requests the Bluetooth® connection to be terminated by sending a CSR_BT_DUNC_DISCONNECT_REQ to DUN-DT, which will response with a CSR_BT_DUNC_DISCONNECT_IND when the connection is closed. The CSR_BT_DUNC_DISCONNECT_IND can also be sent to the application sporadic if the DUN gateway has requested a disconnection of the Bluetooth® connection.

Parameters

type	Signal identity, CSR_BT_DUNC_DISCONNECT_REQ/IND.
duncInstanceId	The handle of the DUNC profile instance that send the message.
localTerminated	TRUE if termination of connection happened on request from the local host; FALSE otherwise.
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 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

Library Function

```
void CsrBtDuncDisconnectReqSend( CsrSchedQid  duncInstanceId )
```

duncInstanceId	The queue of the DUN-DT instance to send the message to.

4.10 CSR_BT_DUNC_STATUS

Primitives \ Parameters	type	duncInstanceId	deviceAddr	connected	maxMsgSize
CSR_BT_DUNC_STATUS_IND	✓	✓	✓	✓	✓

Table 10: CSR_BT_DUNC_STATUS Primitives

Description

Is used for informing the *DUN-DT Data App* about changes in the Bluetooth® connection.

This signal is only send to the DUN_DT Data Application and not to the Bluetooth Management application, not even if the Bluetooth Management Application has registered its application handle using the CSR_BT_DUNC_REGISTER_DATA_PATH_HANDLE_REQ.

Parameters

type	Signal identity, CSR_BT_DUNC_STATUS_IND.
duncInstanceId	Queue id of the DUNC instance.
deviceAddr	The Bluetooth® address of the gateway to connect to.
connected	TRUE if the connection is established, and FALSE if the connection is disconnected.
maxMsgSize	The maximum packet size that the DUN-DT profile can accept from the application layer. The application layer must fragment messages larger than this size. The *maxMsgSize* parameter must be ignored if the *connected* parameter is FALSE.

4.11 CSR_BT_DUNC_SECURITY_OUT

Primitives \ Parameters	type	appHandle	secLevel	resultCode	resultSupplier
CSR_BT_DUNC_SECURITY_OUT_REQ	✓	✓	✓		
CSR_BT_DUNC_SECURITY_OUT_CFM	✓			✓	✓

Table 11: CSR_BT_DUNC_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_DUNC_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. 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	Reference
CSR Synergy Bluetooth. CM – Connection Manager API Description, doc. no. api-0101-cm	[CM]
Specification of the Bluetooth System, ver 1.1, Profiles K:7	[DUNSPEC]
CSR Synergy Bluetooth, SC – Security Controller API Description	[SC]
Scheduler API, api-0004-coal	[SCHED]

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
CM	Connection Manager
DUN	Dial-up Networking
DT	Data Terminal
GW	Gateway
HMSC	High-level Message Sequence Chart
MSC	Message Sequence Chart
SC	Security Controller
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.