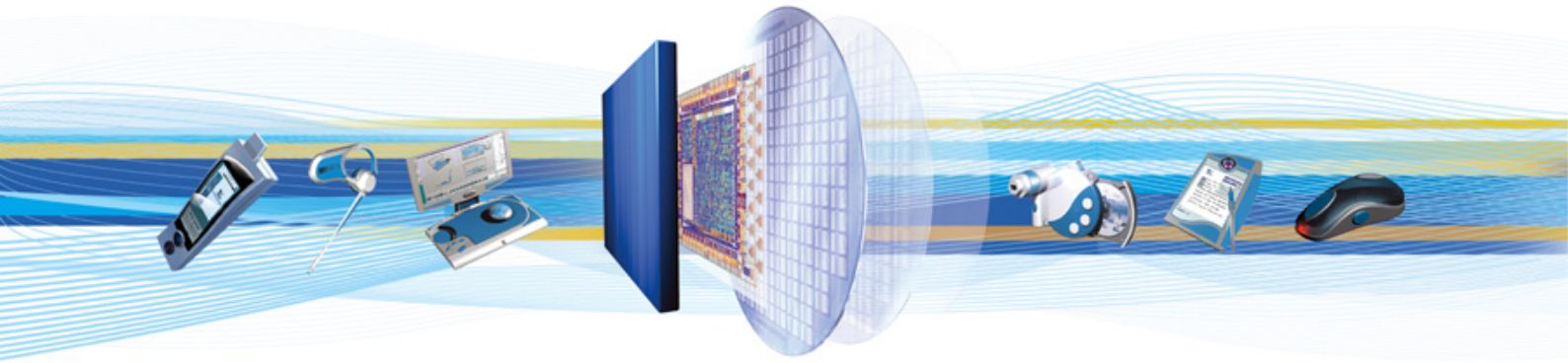
CSR Synergy Bluetooth 18.2.0

JSR82 Java™

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

No table of figures entries found.

1 Introduction

1.1 Introduction and Scope

This document describes the functionality and the message interface of the JSR82 layer of CSR Synergy Bluetooth.

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. the JSR82 task and the application, for communication between the components
- Only one instance of the JSR82 task is active
- For peer-initiated security operations (i.e. authentication and authorization) there is a working default security manager, as described in api-0102-sc [SC], concerning the security controller

NOTE: Due to resource constraints, the L2CAP API of the JSR82 layer may not work on RFCOMM builds of CSR Synergy Bluetooth. This depends on the number and types of other profiles included on the system.

2 Description

The JSR82 layer of CSR Synergy Bluetooth is based on the “Java™ APIs for Bluetooth™ Wireless Technology (JSR-82)” Specification [JSR82SPEC]. It provides a signal-based interface to allow Java applications to make use of Bluetooth, and includes functionality for:

- Device inquiry
- Service discovery
- Create and maintain L2CAP- and RFCOMM connections
- L2CAP- and RFCOMM-based data transfer
- Setup of the local Bluetooth device

Where possible, the signals of the JSR82 layer have been made to correspond one-to-one with the methods of the Java classes in [JSR82SPEC]. The signal names reflect this. For example, the startInquiry() method of the DiscoveryAgent Java class corresponds to the CSR Synergy Bluetooth signal CSR_BT_JSR82_DA_START_INQUIRY_REQ.

2.1 Reference Model

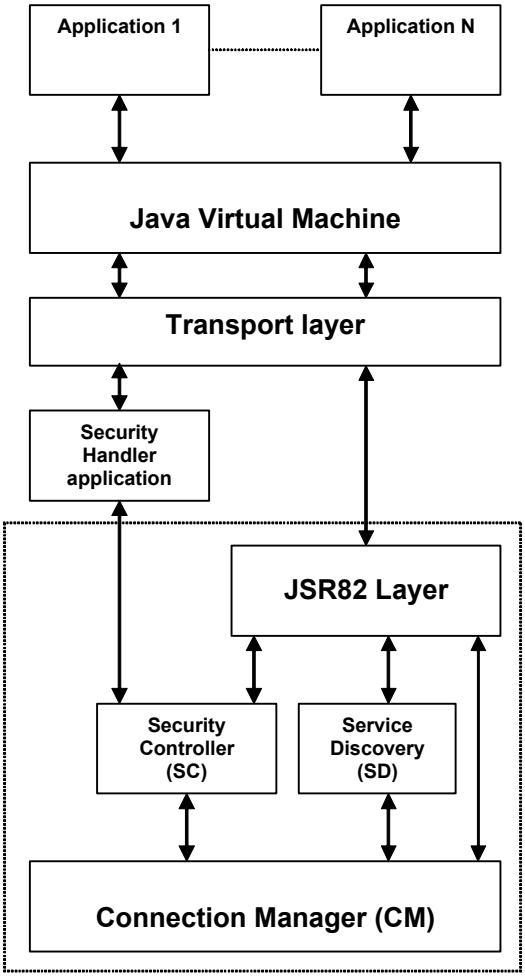


Figure 1: Reference model

As illustrated in the reference model of Figure 1, the developer of a Java application needs to be concerned about two interfaces. One is for the JSR82 layer of CSR Synergy Bluetooth, using the signals described in this

document. The other is for an application security handler, which must allow the end user to respond to security related signals sent from the security controller of CSR Synergy Bluetooth. More on this in section 3.5.

The JSR82 layer does not provide any direct interface to the Java Virtual Machine (JVM), or any process running inside of this. It makes use of the standard CSR Synergy Bluetooth message transport, and it is the user's responsibility to forward messages to the JVM. Where multiple applications require access to the JSR82 layer, responses to individual signals can be distinguished by a request ID, which must be supplied from the transport layer. This request ID is echoed back unchanged with the confirm signals.

2.2 Sequence Overview

Figure 2 presents a state machine model of the JSR82 layer.

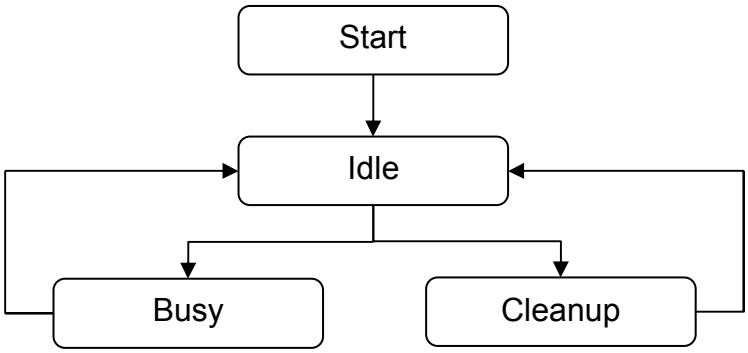


Figure 2: Sequence overview

The model has 3 significant states: Idle, Busy and Cleanup.

2.2.1 The Idle State

The JSR82 layer is in idle state when not processing requests from the transport layer. In this state, all signals that are not cancel signals will be processed immediately when received.

2.2.2 The Busy State

In the busy state, most signals received will be put on an internal queue for later processing. Exceptions are buffer status and data retrieval for RFCOMM/L2CAP connections and cancel signals.

2.2.3 The Cleanup State

When the JSR82 layer is in the cleanup state, all new messages will be discarded.

3 Interface Description

3.1 The Inquiry and Service Discovery API

Device inquiry and service discovery are covered by the DiscoveryAgent and DiscoveryListener classes of [JSR82SPEC]. Signals corresponding to these classes are prefixed with “CSR_BT_JSR82_DA_” and “CSR_BT_JSR82_DL” in the JSR82 layer.

3.1.1 Inquiry

The JSR82 layer is set into inquiry mode using the CSR_BT_JSR82_DA_START_INQUIRY_REQ signal. Once in inquiry mode CSR_BT_JSR82_DL_DEVICE_DISCOVERED_IND signals will be sent to the Java application, whenever a device is discovered (repeating any devices that are discovered multiple times). When the JSR82 layer leaves inquiry mode, it sends a CSR_BT_JSR82_DL_INQUIRY_COMPLETED_IND signal. This may happen in response to one of three conditions: The Java application has sent a CSR_BT_JSR82_DA_CANCEL_INQUIRY_REQ signal, the inquiry has failed, or an internal timer set to 20 seconds has timed out. After a CSR_BT_JSR82_DL_INQUIRY_COMPLETED_IND signal is received, no more CSR_BT_JSR82_DL_DEVICE_DISCOVERED_IND signals will be received.

In addition to the signals in the general busy state, requests for the remote device’s friendly name will also be enabled during inquiry.

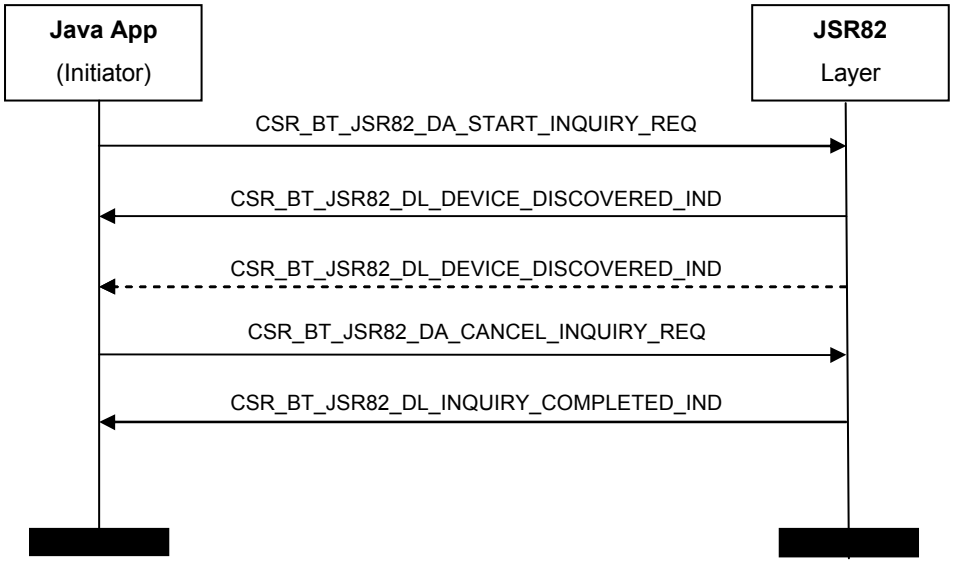


Figure 3: Inquiry

3.1.2 Service Discovery

The CSR_BT_JSR82_DA_SEARCH_SERVICES_REQ signal starts a search on a remote device for any service records that contain all of a given set of universally unique identifiers (UUIDs – see [BT]). If any such service records are found, the JSR82 layer automatically attempts to retrieve the default attributes specified by [JSR82SPEC], as well as a user defined set of extra attributes. If at least one of the requested attributes can be retrieved for a matching service record, a CSR_BT_JSR82_DL_SERVICES_DISCOVERED_IND signal is passed to the Java application. The sequence ends with a CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED_IND signal, which may be sent as a result of either a completion/failure, or as a response to a CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH_REQ signal from the Java application.

The completion indication carries two variables called serviceDataBaseState, and serviceDBStateValid. The transport layer must save these for any records that it may need to retrieve additional attributes for later (see section 3.3.3)

Until the CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED_IND signal is received, the JSR82 layer is in a service search state, where all signals will be queued. Exceptions are the cancel signal, and buffer status and data requests for L2CAP and RFCOMM connections.

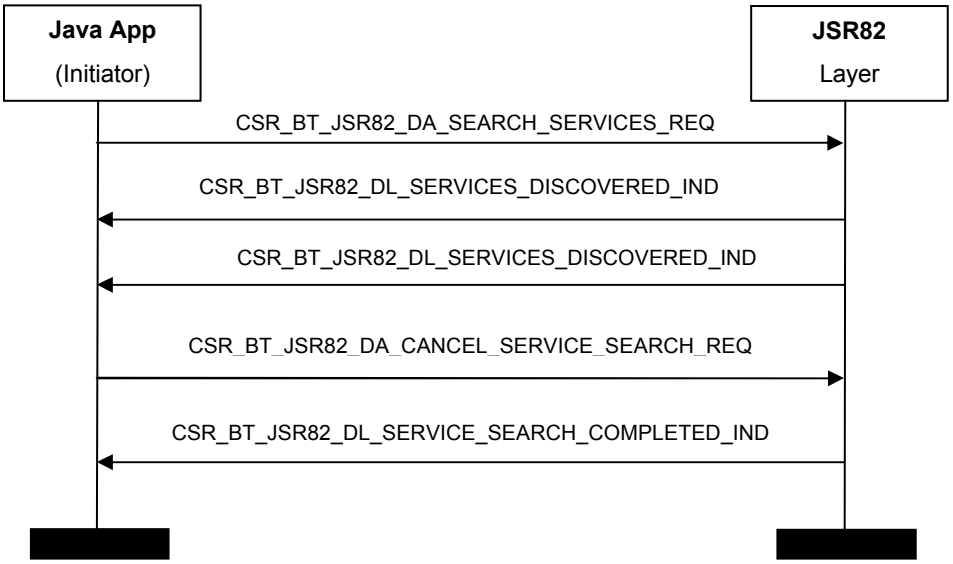


Figure 4: Service Search

3.1.3 Device Retrieval

An alternative to the device inquiry process exists in the CSR_BT_JSR82_DA_RETREIVE_DEVICES_REQ. Using this signal, the JSR82 layer can return either all devices that have previously been found by a Java application, or all pre-known devices – i.e. all devices that have been paired in the SC.

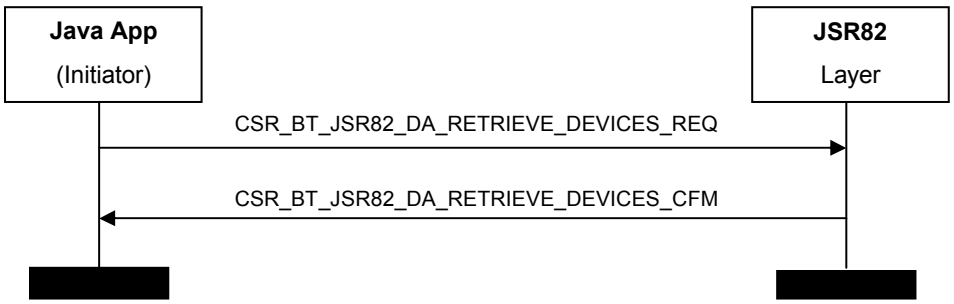


Figure 5: Device Retrieval

3.1.4 Service Select

An alternative to the service discovery process is the service select process initiated by CSR_BT_JSR82_DA_SELECT_SERVICE_REQ. This process searches all nearby devices for a specified service, and when one is found, it is returned by the CSR_BT_JSR82_DA_SELECT_SERVICE_CFM_SIGNAL.

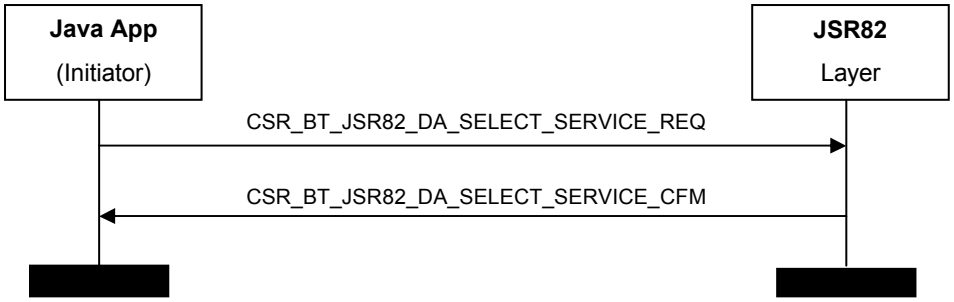


Figure 6: Service Select

3.2 The Local Device API

The local device API of the JSR82 layer provides support for reading status and changing settings of the local Bluetooth device. Additionally, it allows the Java application to update service records that it has placed in the service discovery database (SDDB) of the local device.

3.2.1 Get Local Bluetooth Address

The Bluetooth address of the local device can be retrieved using the CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS_REQ/CFM signal pair.

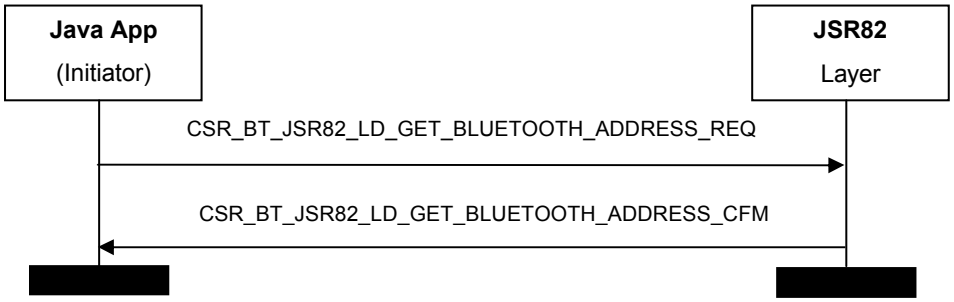


Figure 7: Get local bluetooth address

3.2.2 Update Record

The CSR_BT_JSR82_LD_UPDATE_RECORD_REQ attempts to update a service record in the local SDDB. Note that while the service record handle is unchanged from the perspective of the Java application, the update process will cause the service record handle to be changed from the perspective of any remote devices reading the record.

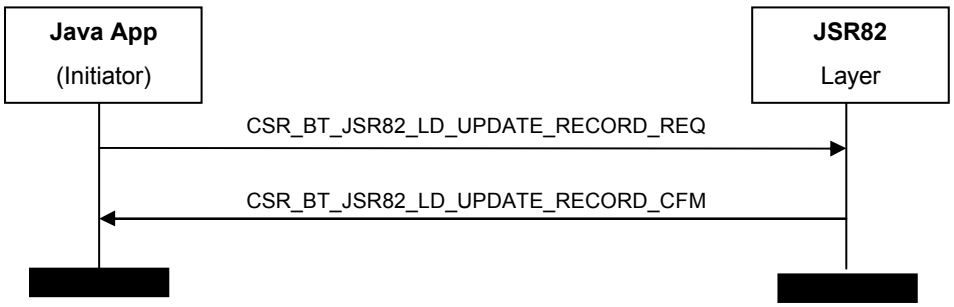


Figure 8: Update service record

3.2.3 Get Friendly Name

The CSR_BT_JSR82_LD_GET_FRIENDLY_NAME_REQ/CFM signal pair requests the friendly name of the local Bluetooth device.

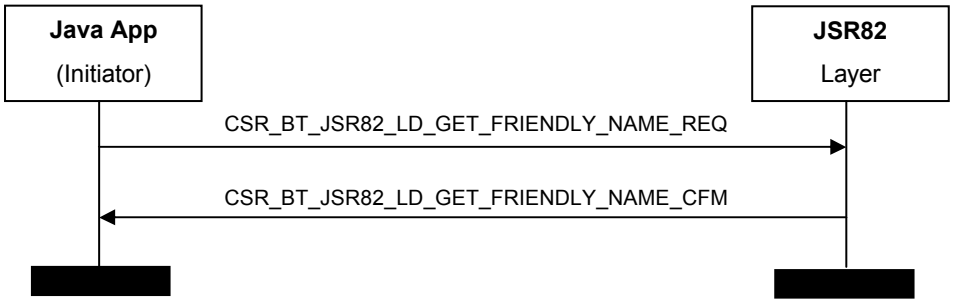


Figure 9: Get friendly name

3.2.4 Get Discoverable

The CSR_BT_JSR82_LD_GET_DISCOVERABLE_REQ/CFM signal pair returns the current discoverable mode of the local device, indicating whether or not the device will respond to inquiries from other Bluetooth devices in the area.

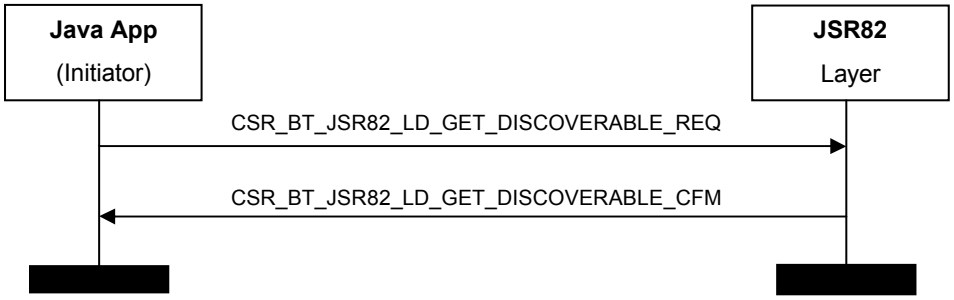


Figure 10: Get discoverable mode

3.2.5 Set Discoverable

The CSR_BT_JSR82_LD_SET_DISCOVERABLE_REQ/CFM signal pair sets the discoverable mode of the local device, determining if it will respond to inquiry scans of other Bluetooth devices in the area. Note that the ACCEPT_AND_OPEN signals of the L2CAP and RFCOMM APIs in this document automatically set the device discoverable.

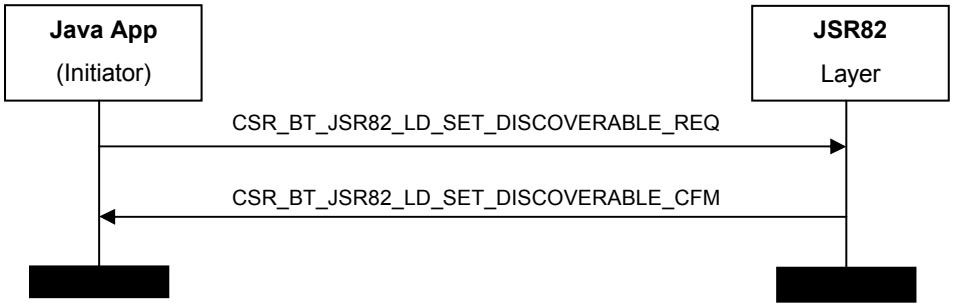


Figure 11: Set discoverable mode

3.2.6 Get Property

The CSR_BT_JSR82_LD_GET_PROPERTY_REQ/CFM signal pair allows the Java application to get a number of properties from the JSR82 layer of CSR Synergy Bluetooth. These properties include:

- The JSR82 specification API version
- Whether master/slave switch is supported
- The maximum number of concurrent attribute fetches in the service discovery layer
- The maximum number of concurrent ACL links
- The number of concurrent service discovery transactions
- Whether inquiry scanning during a connection is allowed
- Whether page scanning is allowed during connection
- Whether inquiry is allowed during a connection
- Whether paging is allowed during a connection

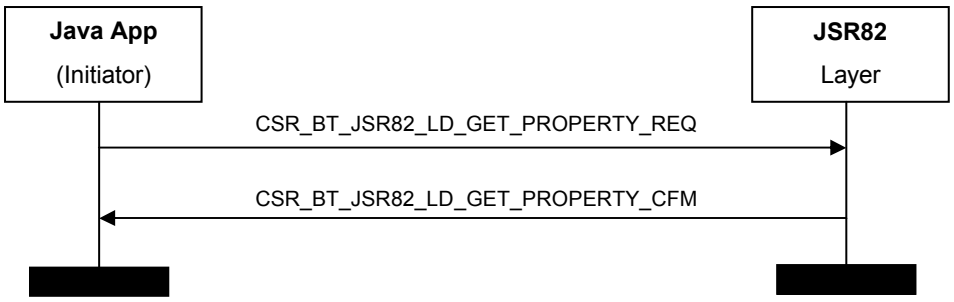


Figure 12: Get property

3.2.7 Get Device Class

The CSR_BT_JSR82_LD_GET_DEVICE_CLASS_REQ/CFM signal pair gets the current class of device (COD) from the local Bluetooth device.

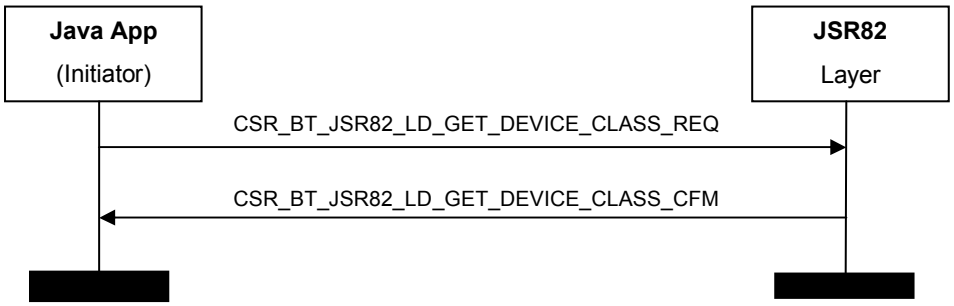


Figure 13: Get device class

3.2.8 Set Device Class

The CSR_BT_JSR82_LD_SET_DEVICE_CLASS_REQ/CFM signal pair sets the current class of device (COD) of the local Bluetooth device. Note that the JSR82 layer will not allow a Java application to clear bits in the COD that were set by other components of CSR Synergy Bluetooth.

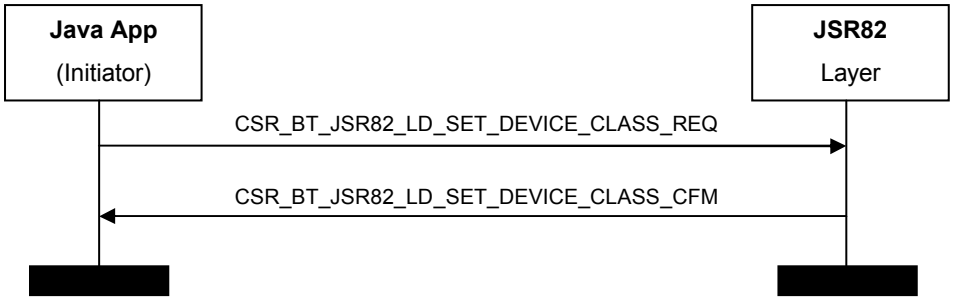


Figure 14: Set device class

3.2.9 Get Security Level

The CSR_BT_JSR82_LD_GET_SECURITY_LEVEL_REQ/CFM signal pair gets the security level of the local device.

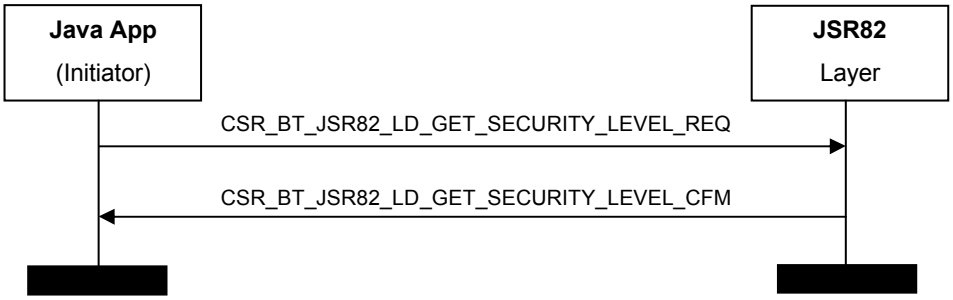


Figure 15: Get security level

3.2.10 Get Master/Slave Role

The CSR_BT_JSR82_LD_IS_MASTER_REQ/CFM signal pair gets the current master/slave role of the local device, in any connections to a remote device, the address of which is given in the request.

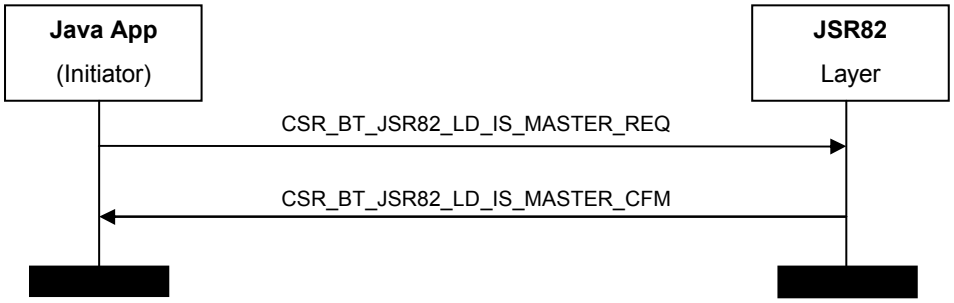


Figure 16: Get master/slave role

3.3 Service Record Manipulation API

The service record API contains signals to create, register and remove service records in the service discovery database of the local device.

3.3.1 Creating a Record

The CSR_BT_JSR82_SR_CREATE_RECORD_REQ/CFM signal pair reserves a service record handle locally in the JSR82 layer of CSR Synergy Bluetooth, to be used in a later registration to the SDDB. This service record handle is for use within the Java application, and does not necessarily correspond to the one that the later

registration will assign to this record in the SDDB. The mapping between these handles is transparent to the Java application.

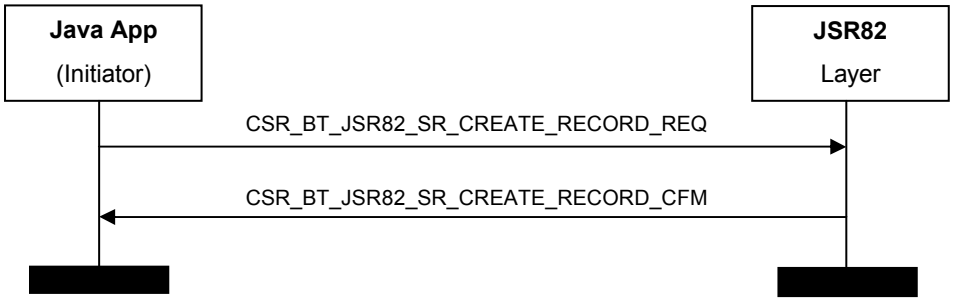


Figure 17: Create service record

3.3.2 Registering a Record

The CSR_BT_JSR82_SR_REGISTER_RECORD_REQ/CFM signal pair registers a service record created in a Java application to the local SDDB. To do this, it uses a service record handle that must be gotten with the CSR_BT_JSR82_SR_CREATE_RECORD_REQ signal beforehand.

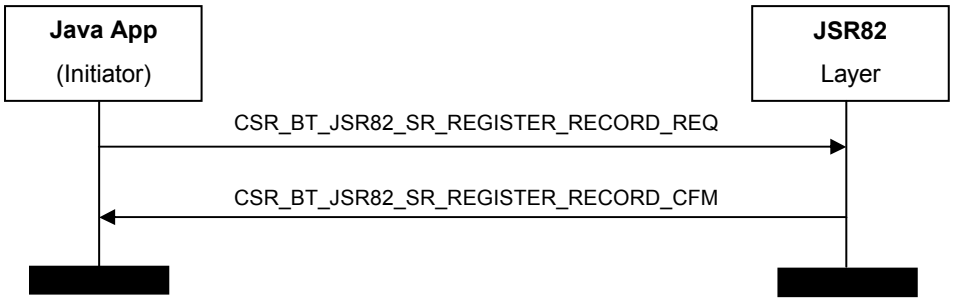


Figure 18: Register service record

3.3.3 Requesting Additional Attributes

For a service record obtained from a remote device through service discovery, it is possible to request additional attributes. This may be done using the CSR_BT_JSR82_SR_POPULATE_RECORD_REQ/CFM signal pair. Note that this operation will fail if there have been any changes to the SDDB of the remote device, as it is then no longer possible to guarantee the validity of the service record. To determine if the SDDB of the remote device has changed, information about its state must be saved by the transport layer when getting or updating (populating) the service record.

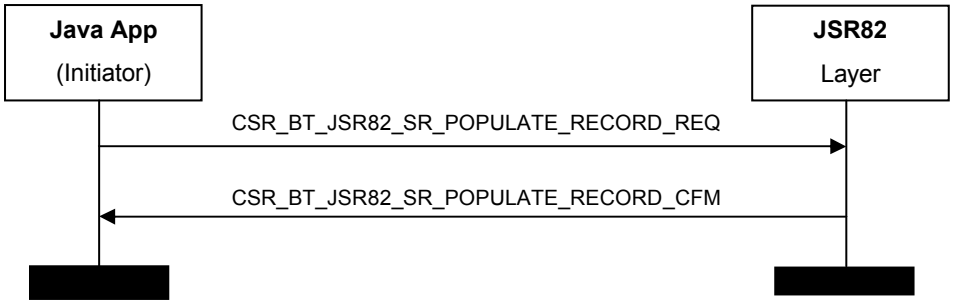


Figure 19: Requesting additional attributes

3.3.4 Removing a Record

The CSR_BT_JSR82_SR_REMOVE_RECORD_REQ/CFM signal pair unregisters a service record from the local SDDB.

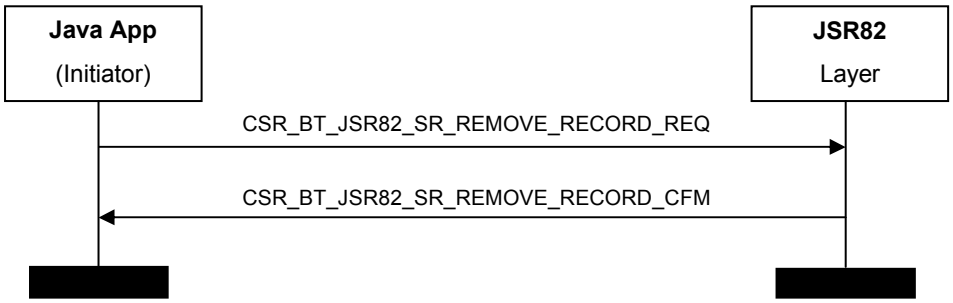


Figure 20: Removing a service record

3.4 The L2CAP API

The JSR82 layer L2CAP API contains the signals that are necessary to set up a L2CAP connection from a Java application using the L2CapConnection and L2CapConnectionNotifier interfaces.

From the server side, setting up a L2CAP connection entails three steps:

- Registering a service record. See section 3.3
- Reserving a PSM
- Setting the device ready to accept connections (CSR_BT_JSR82_L2CA_ACCEPT_AND_OPEN_REQ)

From the client side, connecting to a server also entails two steps:

- Finding a service record on a remote device, and extracting the PSM to connect to (see section 3.1)
- Connecting (CSR_BT_JSR82_L2CA_CONNECT_REQ)

3.4.1 Reserving a PSM

Before the JSR82 low-level profile can start accepting a connection, a PSM must be reserved for the server in the local L2CAP layer. A PSM can be requested by sending a CSR_BT_JSR82_L2CA_GET_PSM_REQ to the JSR82 layer. The PSM will then be returned in a CSR_BT_JSR82_L2CA_GET_PSM_CFM signal. This PSM must then be stored in a service record in order to let a connecting client find the service.

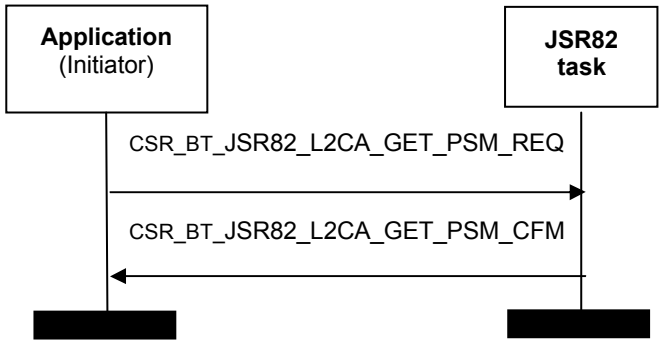


Figure 21: PSM reservation sequence

Please note that the PSM obtained must be used in all subsequent L2CAP signals relating to the service.

3.4.2 Accepting a Connection from a Client

When a server is ready to accept connections to a service, it sends the CSR_BT_JSR82_L2CA_ACCEPT_REQ to the JSR82 layer. When a client then connects to this service, a CSR_BT_JSR82_L2CA_ACCEPT_CFM signal is returned containing the address of the connecting device. This signal also contains the connection ID needed when using this connection to transfer data.

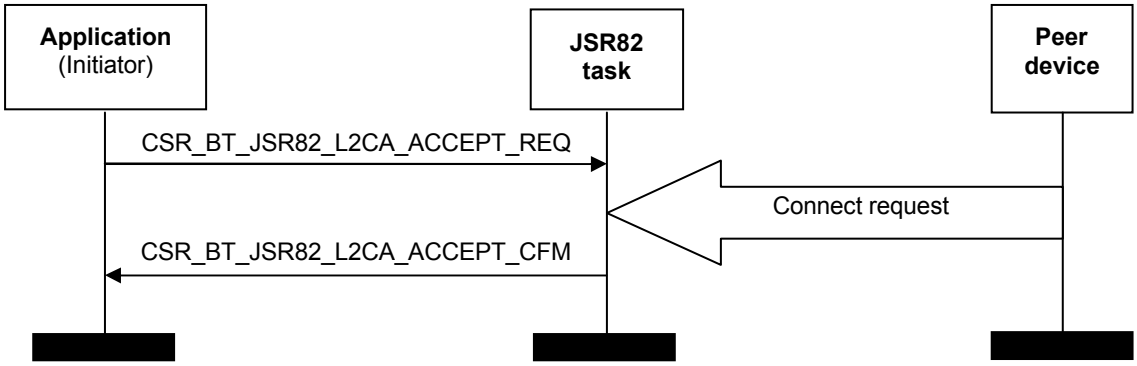


Figure 22: Connection from server

3.4.3 Connecting to a Server

When a client wants to connect to a L2CAP server, it sends the CSR_BT_JSR82_L2CA_OPEN_REQ to the JSR82 layer. A connection is then confirmed by a CSR_BT_JSR82_L2CA_OPEN_CFM signal. This signal contains a connection ID that must be used when referring to the connection.

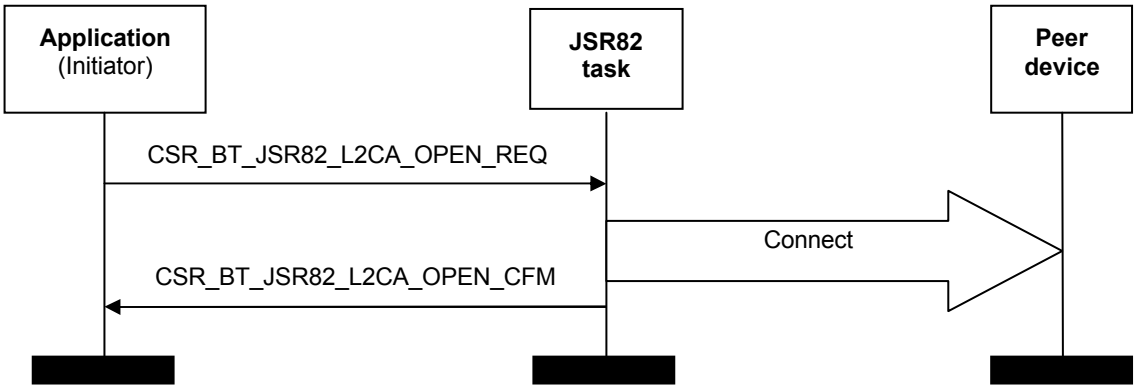


Figure 23: Connection from client

3.4.4 Disconnecting a L2CAP Connection Locally

If the local device wants to disconnect a L2CAP connection, it sends a CSR_BT_JSR82_L2CA_DISCONNECT_REQ to the JSR82 layer. Disconnect is then confirmed by a CSR_BT_JSR82_L2CA_DISCONNECT_CFM signal. Note that the disconnect closes all connections to a PSM and unregisters the PSM.

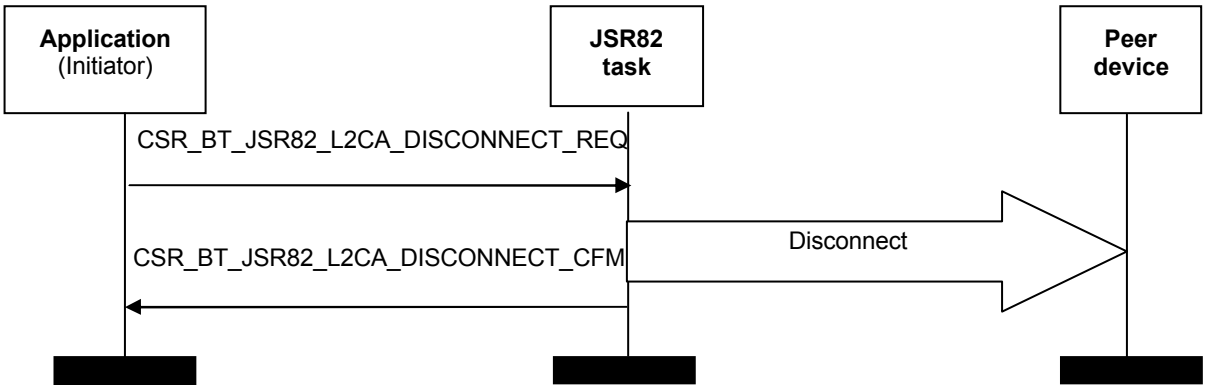


Figure 24: Disconnect initiated locally

3.4.5 Disconnect Initiated from Peer

If the peer device closes a L2CAP connection, the JSR82 layer will not immediately close the connection locally. The JSR82 layer will keep the connection's receive buffer until the local application sends the CSR_BT_JSR82_L2CA_DISCONNECT_REQ signal. A disconnect can be detected when the data operations start returning disconnected status.

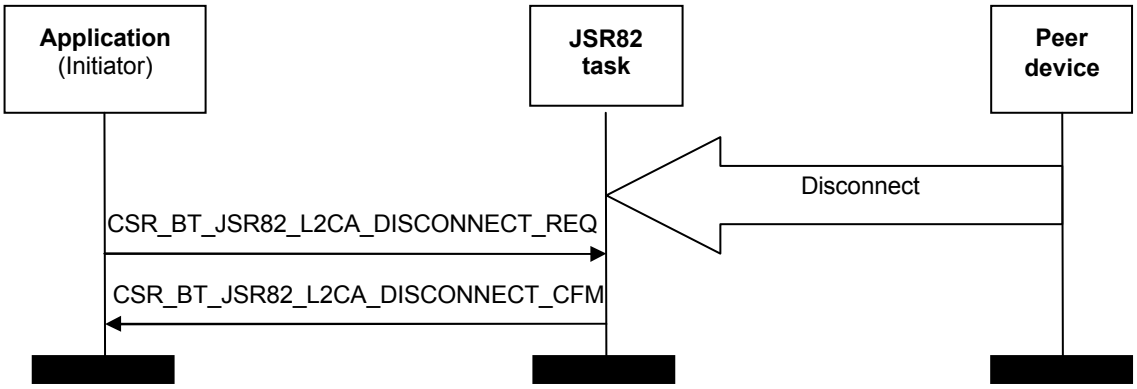


Figure 25: Disconnect initiated remotely

3.4.6 Closing a L2CAP Connection Locally

If the local device wants to close a L2CAP connection, it sends a CSR_BT_JSR82_L2CA_CLOSE_REQ to the JSR82 layer. Disconnect is then confirmed by a CSR_BT_JSR82_L2CA_CLOSE_CFM signal. Note that if it is a server connection, the PSM for the connection will still be registered after this, and can therefore be used for accepting another connection.

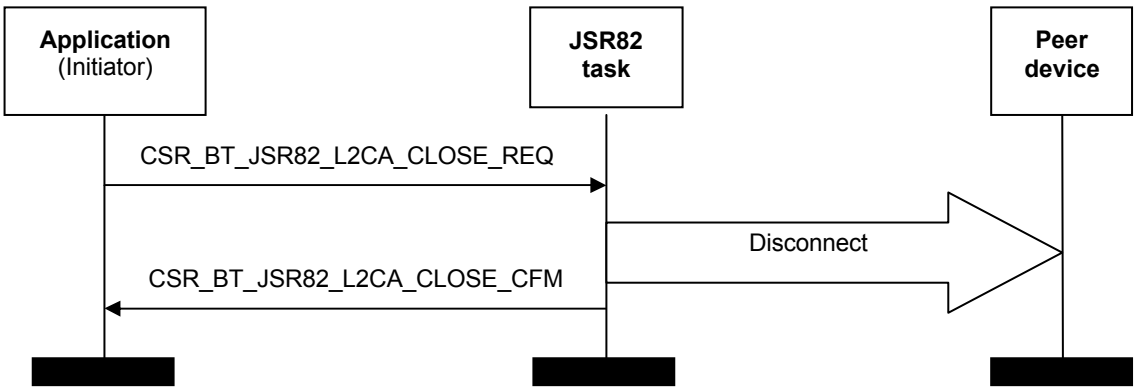


Figure 26: Close initiated locally

3.4.7 Close Initiated from Peer

If the peer device closes a L2CAP connection, the JSR82 layer will not immediately close the connection locally. The JSR82 layer will keep the connection's receive buffer until the local application sends the CSR_BT_JSR82_L2CA_CLOSE_REQ signal. A disconnect can be detected when the data operations start returning disconnected status.

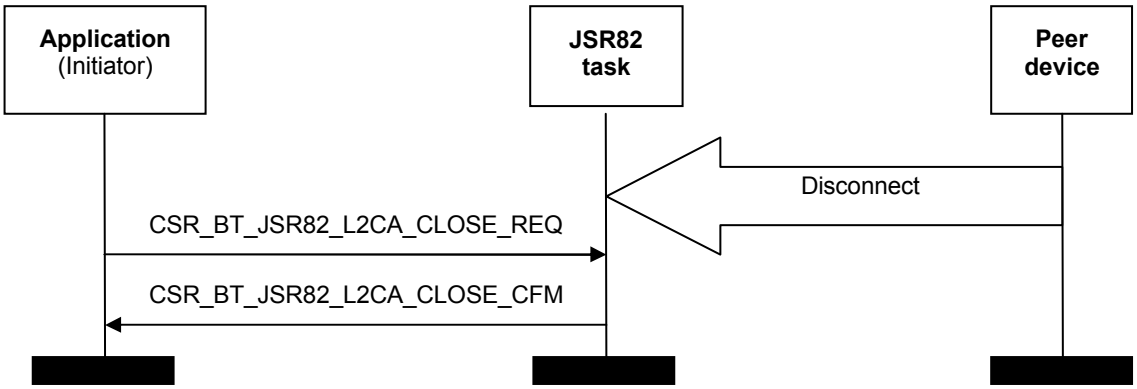


Figure 27: Close initiated remotely

3.4.8 Sending L2CAP Data

When the local device wants to send a L2CAP data packet, it sends a CSR_BT_JSR82_L2CA_TX_DATA_REQ to the JSR82 layer. When the data has been sent, it responds with a CSR_BT_JSR82_L2CA_TX_DATA_CFM.

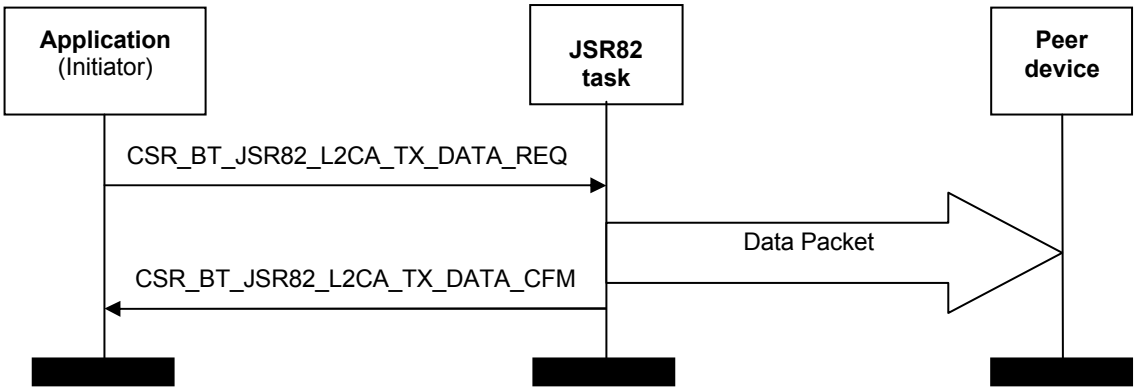


Figure 28: Send data to peer

3.4.9 Receiving L2CAP Data – Blocking

If the local device wants to receive data, it sends a CSR_BT_JSR82_L2CA_RX_DATA_REQ to the JSR82 layer. When data is ready, this will then result in a CSR_BT_JSR82_L2CA_RX_DATA_CFM signal to be returned. If there is already data in the data buffer, this signal is returned immediately. If there is no data in the buffer, the signal will return when data arrives, or the connection closes.

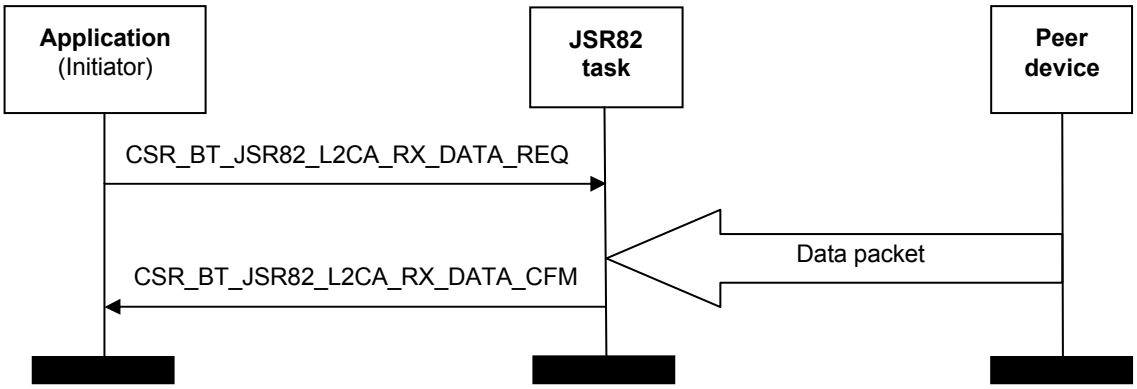


Figure 29: Receive data from peer – blocking

3.4.10 Receiving L2CAP Data – Non-blocking

If the local device wants to receive data, but does not want to risk blocking while waiting for data, it can poll the data buffer before reading it. The data buffer can be polled by sending a CSR_BT_JSR82_L2CA_RX_READY_REQ to the JSR82 layer. The status of the buffer will then be returned in a CSR_BT_JSR82_L2CA_RX_READY_CFM signal.

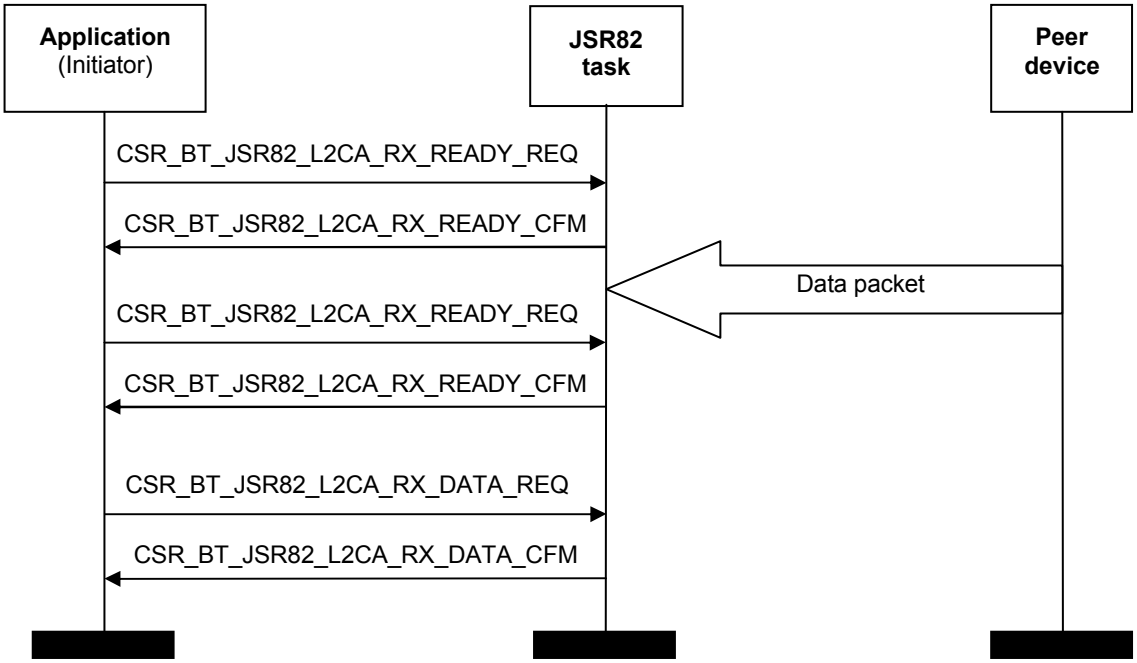


Figure 30: Receive data from peer – non-blocking

3.4.11 Retrieving Connection Parameters

Status of a connection can be retrieved by sending the CSR_BT_JSR82_L2CA_GET_CONFIG_REQ to the JSR82 layer. The MTUs and peer device address are then returned in the CSR_BT_JSR82_L2CA_GET_CONFIG_CFM.

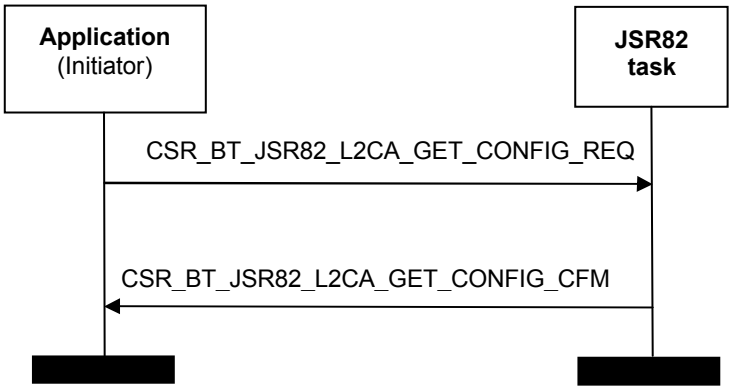


Figure 31: Get the connection configuration

3.5 Remote Device API

The remote device API contains signals to change or check the security status of connections to a remote device, and for getting the friendly name of a remote device.

It is assumed that a security handler application exists on the system, to handle signals sent from the CSR Synergy Bluetooth security controller [SC]. When a remote device initiates an authentication process, the SC will request a pin key. Handling of this pin key request should happen in the default security handler application.

If a Java application needs to initiate an authorisation process, this must also be supported by the default security handler application. This means that any authorisation requests should be routed to the default security handler application, rather than the JSR82 layer.

3.5.1 Get Friendly Name

The CSR_BT_JSR82_RD_GET_FRIENDLY_NAME_REQ/CFM signal pair gets the friendly name of a remote device.

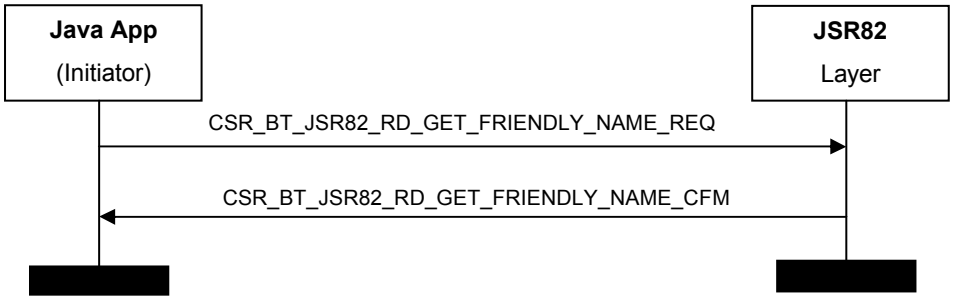


Figure 32: Getting the remote friendly name

3.5.2 Authenticate a Remote Device

The CSR_BT_JSR82_RD_AUTHENTICATE_REQ signal starts a process to authenticate a remote device. Once this process has been started, the user should expect the default application security handler (described in [SC]) to be contacted for a pin key. Once authentication has been completed, the remote device is added to the JSR82 layer's list of authenticated devices, for use with the CSR_BT_JSR82_RD_IS_AUTHENTICATED_REQ signal.

Figure 33: Authenticating a remote device

3.5.3 Check Authentication Status of a Remote Device

The CSR_BT_JSR82_RD_IS_AUTHENTICATED_REQ/CFM signal pair queries the JSR82 layer for the authentication status of a remote device. Note that only devices that have been authenticated as a result of a CSR_BT_JSR82_RD_AUTHENTICATE_REQ from the local device will be present in the JSR82 layer's list of authenticated devices. I.e. any devices which have completed authentication on their own initiative will be reported as not authenticated.

A remote device is removed from the list of authenticated devices once all connections to the device have been terminated.

Figure 34: Checking authentication status

3.5.4 Encrypt Connections to a Remote Device

The CSR_BT_JSR82_RD_ENCRYPT_REQ/CFM signal pair can be used for turning on encryption on all connections to a remote device. Note that because the JSR82 layer has no information about the connections of other profiles included in CSR Synergy Bluetooth, it is not possible to turn encryption off using this signal pair. Attempting to do so will simply result in a confirmation signal with an error code.

Encryption may also be turned on at the creation of a connection.

Figure 35: Encrypting connections

3.5.5 Check Encryption Status of a Remote Device

The CSR_BT_JSR82_RD_IS_ENCRYPTED_REQ/CFM signal pair queries the encryption status of a connection to a remote device. Note that if encryption has been enabled by a CSR Synergy Bluetooth profile outside of the JSR82 layer, this will not be reflected in the result of this query. The effect of this, is that the query may report encryption to be off when in fact it is on. The opposite cannot occur. If encryption is in fact off, it will be reported to be so.

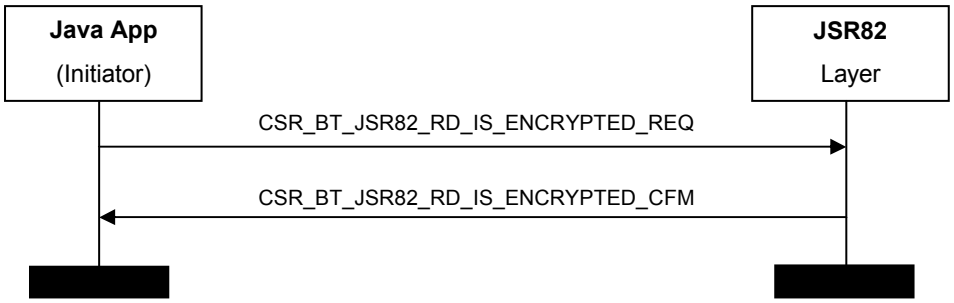


Figure 36: Checking encryption status

3.5.6 Check Trusted Status of a Remote Device

The CSR_BT_JSR82_RD_IS_TRUSTED_REQ/CFM signal pair queries the trusted status of a remote device. This is done by accessing the database of the Service Discovery component [SD] of CSR Synergy Bluetooth. This means that devices that are marked trusted in profiles outside of the JSR82 layer will also be reported trusted by this query.

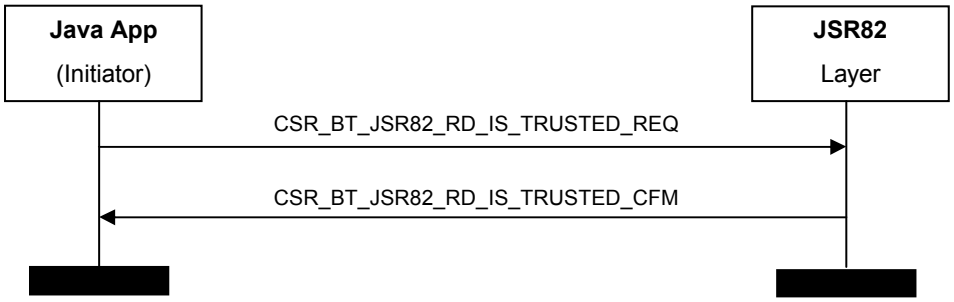


Figure 37: Checking trusted status

3.5.7 Check Connection Status of a Remote Device

The CSR_BT_JSR82_RD_IS_CONNECTED_REQ/CFM signal pair may be used for determining if a L2CAP or RFCOMM connection exists to a remote device. The confirm signal returns a handle, which is tied to the Bluetooth address of the device queried, and which will remain unchanged as long as at least one connection is alive.

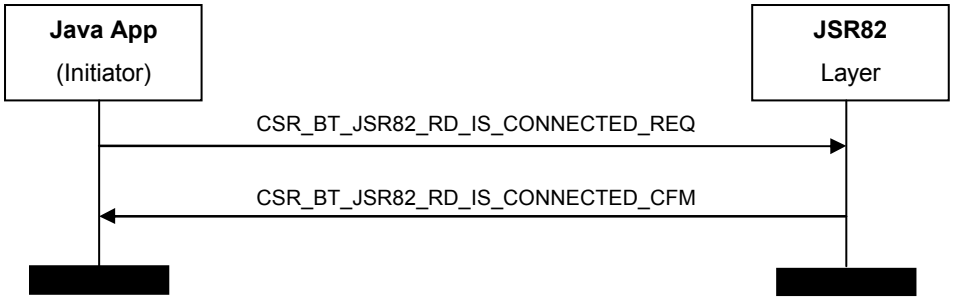


Figure 38: Checking connected status

3.6 RFCOMM API

The RFCOMM API of the JSR82 layer provides signals to set up serial connections to remote Bluetooth devices.

Starting a RFCOMM connection as server requires three steps:

- Reserving a server channel
- Registering a service record with the server channel just obtained
- Setting a RFCOMM server ready to accept connections

Starting a connection as client, also requires two steps:

- Finding a service record on a remote device, and extracting the service channel
- Connecting to the server

To find the record of a service to connect to, the service discovery API of the JSR82 layer, described in section 3.1 should be used.

3.6.1 Reserving a Server Channel

The CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL_REQ/CFM signal pair registers a server channel on the local Bluetooth device, and returns it to the Java application. This server channel may now be inserted in a service record, and registered in the SDDB, using the service record API of JSR82, in section 3.3.

If the user does not wish to use the server channel after getting it, a normal disconnect must be used for unregistering the service channel.

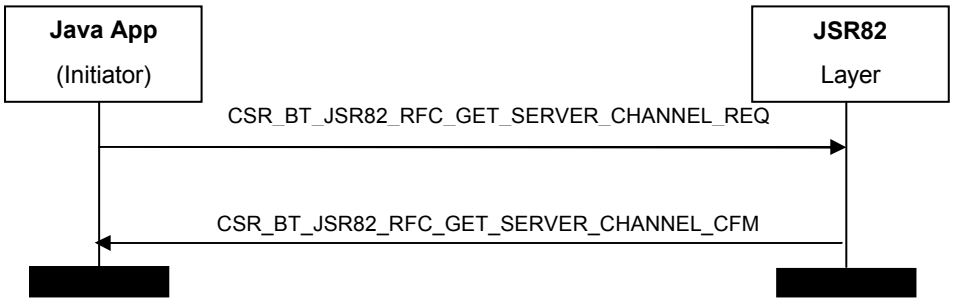


Figure 39: Reserving a server channel

3.6.2 Setting a RFCOMM Server ready to accept Connections

The CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_REQ signal causes the local device to be set connectable, and ready to accept a RFCOMM connection on the specified server channel. Before sending this signal to the JSR82 layer, a service record containing the server channel in question must be registered in the SDDB. The CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_CFM will be returned only when a client connects to the server.

If the user no longer wishes to accept client connections, the CSR_BT_JSR82_RFC_DISCONNECT_REQ signal may be used. Note that no CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_CFM will be sent if a disconnect is requested before a client connects. Remember to remove the service record corresponding to this server, to stop clients from finding it, and possibly attempting to connect to it.

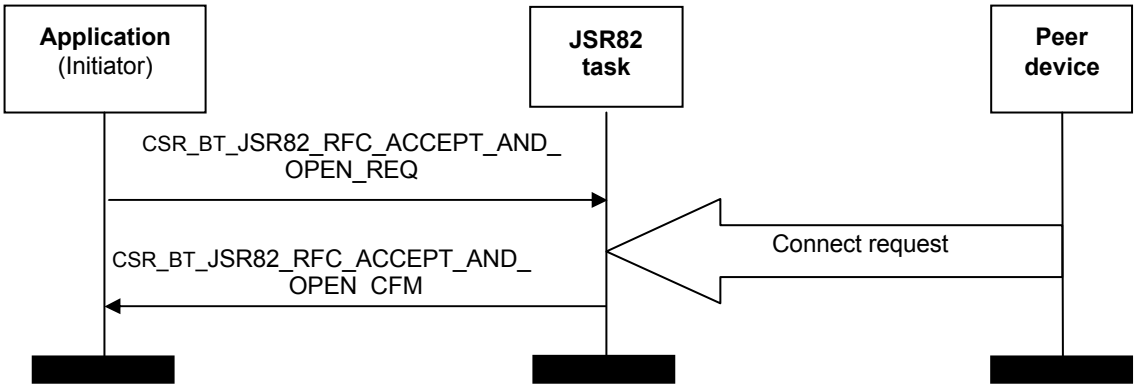


Figure 40: Setting a RFCOMM server ready to accept connections

3.6.3 Connecting to the Server

Once a server channel has been read from a service record, a client may connect to it using the CSR_BT_JSR82_RFC_CONNECT_REQ signal. During the connect, the JSR82 layer will register a local server channel on the local Bluetooth device and return this in the confirm signal. The local server channel is used for identifying this RFCOMM connection on all subsequent transactions.

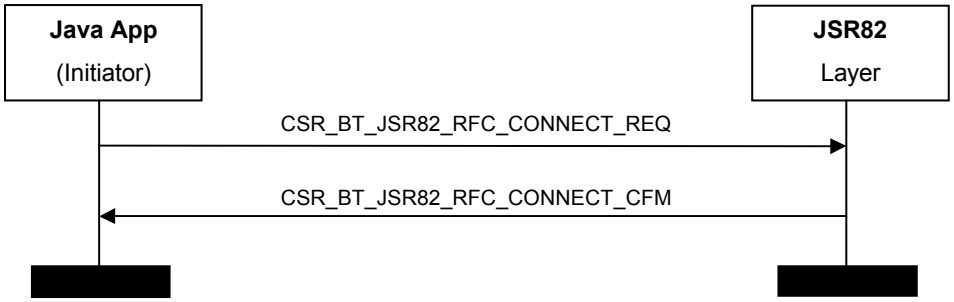


Figure 41: Connecting to a RFCOMM server

3.6.4 Disconnecting a RFCOMM Connection

To disconnect a RFCOMM connection, the CSR_BT_JSR82_RFC_DISCONNECT_REQ signal must be sent on both sides of the connection.

If the peer device disconnects, this is registered by the JSR82 layer. The connection will not be closed until the local application sends a CSR_BT_JSR82_RFC_DISCONNECT_REQ signal. Until this happens, the receive buffer for the connection will be valid, and any data previously received can be retrieved.

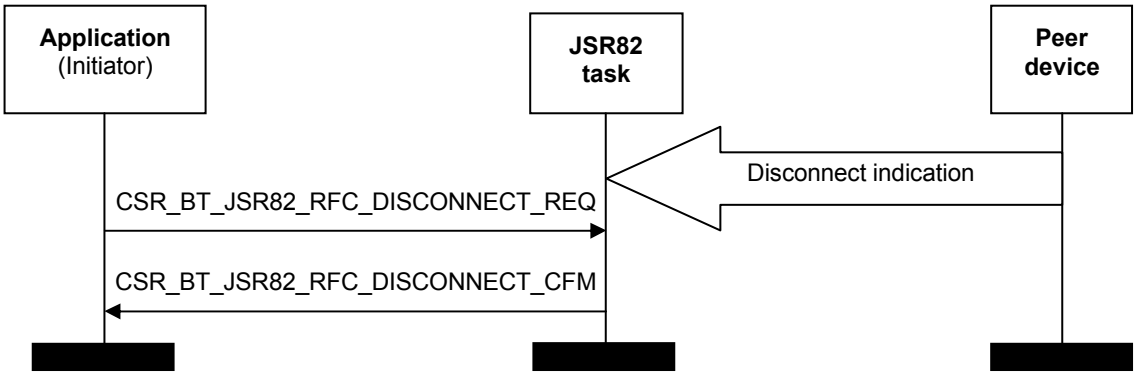


Figure 42: Disconnecting a RFCOMM connection

3.6.5 Closing a RFCOMM Connection

To disconnect a RFCOMM connection, the CSR_BT_JSR82_RFC_CLOSE_REQ signal must be sent on both sides of the connection.

If the peer device disconnects, this is registered by the JSR82 layer. The connection will not be closed until the local application sends a CSR_BT_JSR82_RFC_CLOSE_REQ signal. Until this happens, the receive buffer for the connection will be valid, and any data previously received can be retrieved.

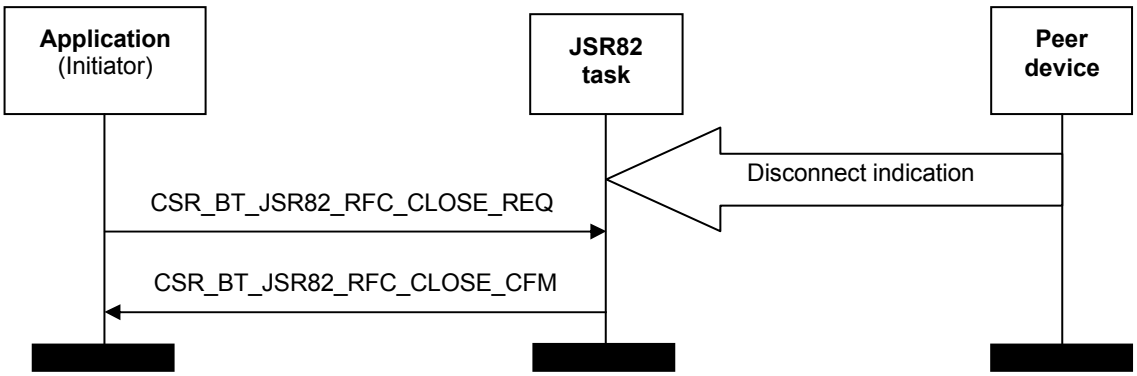


Figure 43: Closing a RFCOMM connection

3.6.6 Sending Data

The CSR_BT_JSR82_RFC_SEND_DATA_REQ starts transmission of its data payload. If the size of the payload is greater than the maximum RFCOMM frame size, the JSR82 layer will segment it, and send it in multiple frames. The user must not deallocate the payload of the request signal, as this is used and deallocated in the JSR82 layer. A CSR_BT_JSR82_RFC_SEND_DATA_CFM is returned when the send process finishes, reporting the actual number of bytes the JSR82 layer managed to send. Since the confirm signal is sent only after the send process has finished, the Java “flush" method is redundant, and no flush signal is implemented.

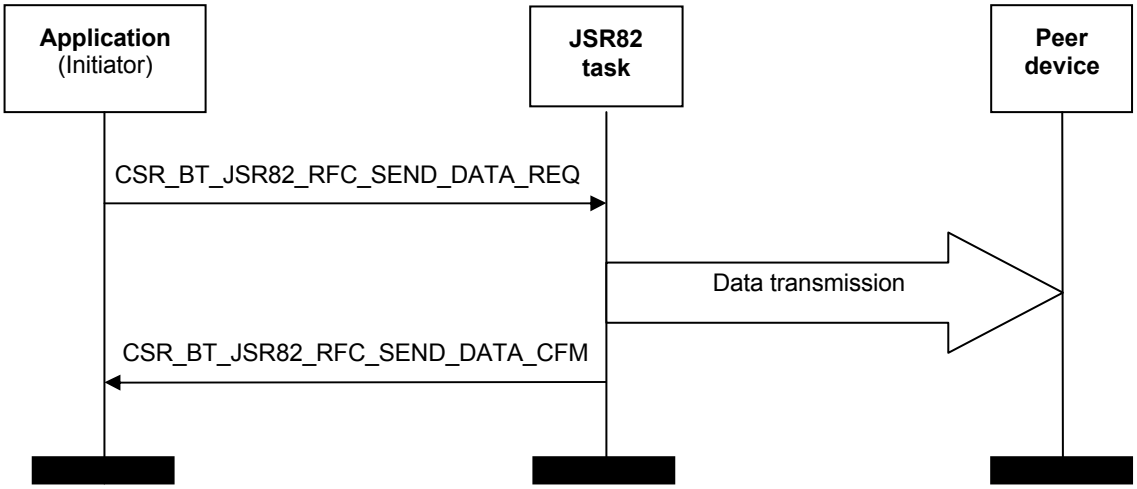


Figure 44: Sending data

3.6.7 Receiving Data

The RFCOMM data receive operation in the JSR82 layer is blocking. When a CSR_BT_JSR82_RFC_RECEIVE_DATA_REQ signal is sent, the JSR82 layer will not respond before data becomes available, or the connection is broken. If fewer bytes than specified in the request signal arrive, the confirm will still be sent. If more bytes than requested arrive, the amount requested will be returned, and the rest will be buffered for the next request.

Note that the JSR82 layer will always accept incoming data on an active connection, regardless of whether an application exists to handle the data. If no such application exists, the receive buffer of the JSR82 layer may become full. If this happens, the flow control of the RFCOMM protocol will cause the remote device to be unable to send data, and may cause a deadlock. The size of the receive buffers for RFCOMM connections in the JSR82 layer can be set in usr_config.h.

To avoid blocking a Java application while waiting for data, the CSR_BT_JSR82_RFC_GET_AVAILABLE_REQ signal may be used.

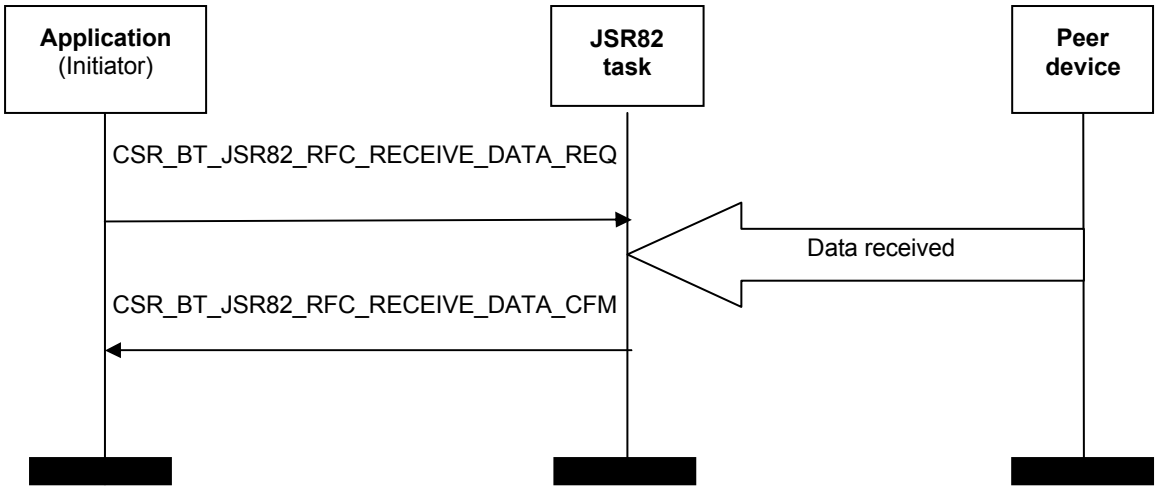


Figure 45: Receiving data

3.6.8 Checking Buffer Status

The CSR_BT_JSR82_RFC_GET_AVAILABLE_REQ/CFM signal pair queries the buffer status of a RFCOMM connection. The confirm signal returns the number of bytes currently available in the receive buffer.

Figure 46: Checking buffer status

3.7 Cleanup Signal

The CSR_BT_JSR82_CLEANUP_REQ signal should be sent to the JSR82 task if all Java applications using Bluetooth have been stopped, or if the JVM itself is being stopped. On receiving the cleanup signal, the JSR82 layer will:

- Disconnect all active L2CAP and RFCOMM connections
- Flush the cache of found devices
- Flush the lists of authenticated and encrypted devices
- Unregister all Java service records
- Delete all pending signals

After the cleanup, the JSR82 layer returns to its idle state, and is ready for a new connection from a Java application.

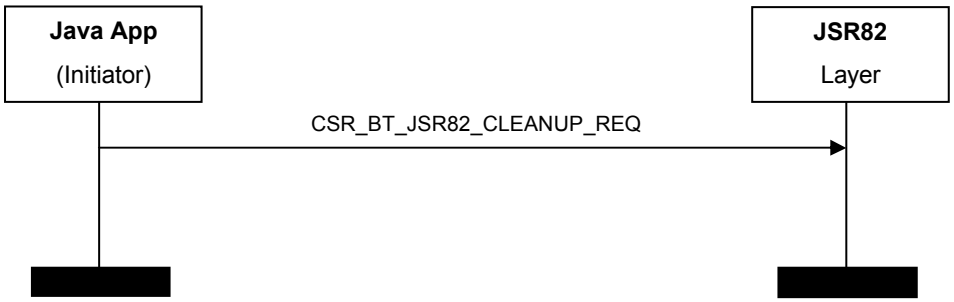


Figure 47: Cleanup signal

3.8 Set Event Mask

This signal can be used for setting which extended information the application will subscribe for, and may be sent momentarily from the application. In the request message the application can defined which extended information it will subscribe for, and the confirm message defines which event has been set. The extended information the application can subscribe for is defined in section 4.51.

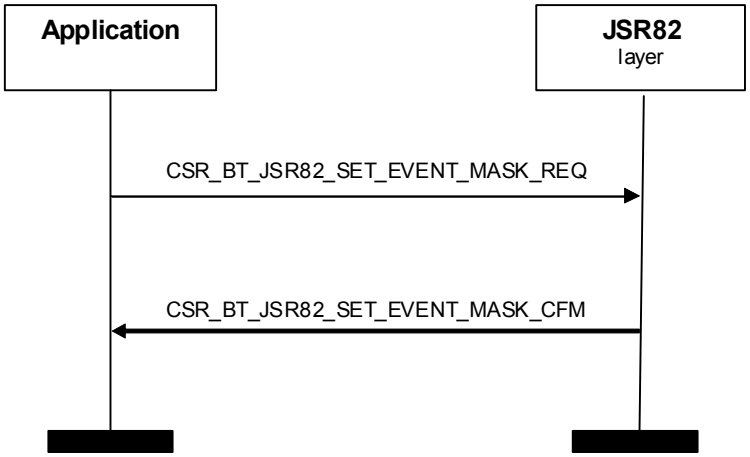


Figure 48: Set Event Mask

4 JSR-82 Primitives

This section gives an overview of the primitives and parameters in the interface. Detailed information can be found in the corresponding csr_bt_jsr82_prim.h file.

4.1 List of All Primitives

Primitives:	Reference:
CSR_BT_JSR82_DA_START_INQUIRY_REQ	See section 4.2
CSR_BT_JSR82_DA_CANCEL_INQUIRY_REQ	See section 4.3
CSR_BT_JSR82_DA_SEARCH_SERVICES_REQ	See section 4.4
CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH_REQ	See section 4.5
CSR_BT_JSR82_DA_RETRIEVE_DEVICES_REQ	See section 4.6
CSR_BT_JSR82_DA_RETRIEVE_DEVICES_CFM	See section 4.6
CSR_BT_JSR82_DA_SELECT_SERVICE_REQ	See section 4.7
CSR_BT_JSR82_DA_SELECT_SERVICE_CFM	See section 4.7
CSR_BT_JSR82_DL_DEVICE_DISCOVERED_IND	See section 4.8
CSR_BT_JSR82_DL_INQUIRY_COMPLETED_IND	See section 4.9
CSR_BT_JSR82_DL_SERVICES_DISCOVERED_IND	See section 4.10
CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED_IND	See section 4.11
CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS_REQ	See section 4.12
CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS_CFM	See section 4.12
CSR_BT_JSR82_LD_GET_PROPERTY_REQ	See section 4.13
CSR_BT_JSR82_LD_GET_PROPERTY_CFM	See section 4.13
CSR_BT_JSR82_LD_SET_DISCOVERABLE_REQ	See section 4.14
CSR_BT_JSR82_LD_GET_DISCOVERABLE_REQ	See section 4.15
CSR_BT_JSR82_LD_GET_DISCOVERABLE_CFM	See section 4.15
CSR_BT_JSR82_LD_GET_FRIENDLY_NAME_REQ	See section 4.16
CSR_BT_JSR82_LD_GET_FRIENDLY_NAME_CFM	See section 4.16
CSR_BT_JSR82_LD_UPDATE_RECORD_REQ	See section 4.17
CSR_BT_JSR82_LD_UPDATE_RECORD_CFM	See section 4.17
CSR_BT_JSR82_LD_GET_DEVICE_CLASS_REQ	See section 4.18
CSR_BT_JSR82_LD_GET_DEVICE_CLASS_CFM	See section 4.18
CSR_BT_JSR82_LD_SET_DEVICE_CLASS_REQ	See section 4.19
CSR_BT_JSR82_LD_SET_DEVICE_CLASS_CFM	See section 4.19
CSR_BT_JSR82_LD_GET_SECURITY_LEVEL_REQ	See section 4.20
CSR_BT_JSR82_LD_GET_SECURITY_LEVEL_CFM	See section 4.20
CSR_BT_JSR82_LD_IS_MASTER_REQ	See section 4.21
CSR_BT_JSR82_LD_IS_MASTER_CFM	See section 4.21
CSR_BT_JSR82_SR_CREATE_RECORD_REQ	See section 4.22
CSR_BT_JSR82_SR_CREATE_RECORD_CFM	See section 4.22
CSR_BT_JSR82_SR_REGISTER_RECORD_REQ	See section 4.23
CSR_BT_JSR82_SR_REGISTER_RECORD_CFM	See section 4.23
CSR_BT_JSR82_SR_REMOVE_RECORD_REQ	See section 4.24
CSR_BT_JSR82_SR_REMOVE_RECORD_CFM	See section 4.24
CSR_BT_JSR82_SR_POPULATE_RECORD_REQ	See section 4.25

Primitives:	Reference:
CSR_BT_JSR82_SR_POPULATE_RECORD_CFM	See section 4.25
CSR_BT_JSR82_L2CA_GET_PSM_REQ	See section 4.26
CSR_BT_JSR82_L2CA_GET_PSM_CFM	See section 4.26
CSR_BT_JSR82_L2CA_ACCEPT_REQ	See section 4.27
CSR_BT_JSR82_L2CA_ACCEPT_CFM	See section 4.27
CSR_BT_JSR82_L2CA_OPEN_REQ	See section 4.28
CSR_BT_JSR82_L2CA_OPEN_CFM	See section 4.28
CSR_BT_JSR82_L2CA_DISCONNECT_REQ	See section 4.29
CSR_BT_JSR82_L2CA_DISCONNECT_CFM	See section 4.29
CSR_BT_JSR82_L2CA_CLOSE_REQ	See section 4.30
CSR_BT_JSR82_L2CA_CLOSE_CFM	See section 4.30
CSR_BT_JSR82_L2CA_TX_DATA_REQ	See section 4.31
CSR_BT_JSR82_L2CA_TX_DATA_CFM	See section 4.31
CSR_BT_JSR82_L2CA_RX_DATA_REQ	See section 4.32
CSR_BT_JSR82_L2CA_RX_DATA_CFM	See section 4.32
CSR_BT_JSR82_L2CA_RX_READY_REQ	See section 4.33
CSR_BT_JSR82_L2CA_RX_READY_CFM	See section 4.33
CSR_BT_JSR82_L2CA_GET_CONFIG_REQ	See section 4.34
CSR_BT_JSR82_L2CA_GET_CONFIG_CFM	See section 4.34
CSR_BT_JSR82_RD_GET_FRIENDLY_NAME_REQ	See section 4.35
CSR_BT_JSR82_RD_GET_FRIENDLY_NAME_CFM	See section 4.35
CSR_BT_JSR82_RD_AUTHENTICATE_REQ	See section 4.36
CSR_BT_JSR82_RD_AUTHENTICATE_CFM	See section 4.36
CSR_BT_JSR82_RD_IS_AUTHENTICATED_REQ	See section 4.37
CSR_BT_JSR82_RD_IS_AUTHENTICATED_CFM	See section 4.37
CSR_BT_JSR82_RD_ENCRYPT_REQ	See section 4.38
CSR_BT_JSR82_RD_ENCRYPT_CFM	See section 4.38
CSR_BT_JSR82_RD_IS_ENCRYPTED_REQ	See section 4.39
CSR_BT_JSR82_RD_IS_ENCRYPTED_CFM	See section 4.39
CSR_BT_JSR82_RD_IS_TRUSTED_REQ	See section 4.40
CSR_BT_JSR82_RD_IS_TRUSTED_CFM	See section 4.40
CSR_BT_JSR82_RD_IS_CONNECTED_REQ	See section 4.41
CSR_BT_JSR82_RD_IS_CONNECTED_CFM	See section 4.41
CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL_REQ	See section 4.42
CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL_CFM	See section 4.42
CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_REQ	See section 4.43
CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_CFM	See section 4.43
CSR_BT_JSR82_RFC_CONNECT_REQ	See section 4.44
CSR_BT_JSR82_RFC_CONNECT_CFM	See section 4.44
CSR_BT_JSR82_RFC_DISCONNECT_REQ	See section 4.45
CSR_BT_JSR82_RFC_DISCONNECT_CFM	See section 4.45
CSR_BT_JSR82_RFC_CLOSE_REQ	See section 4.46
CSR_BT_JSR82_RFC_CLOSE_CFM	See section 4.46
CSR_BT_JSR82_RFC_GET_AVAILABLE_REQ	See section 4.47

Primitives:	Reference:
CSR_BT_JSR82_RFC_GET_AVAILABLE_CFM	See section 4.47
CSR_BT_JSR82_RFC_SEND_DATA_REQ	See section 4.48
CSR_BT_JSR82_RFC_SEND_DATA_CFM	See section 4.48
CSR_BT_JSR82_RFC_RECEIVE_DATA_REQ	See section 4.49
CSR_BT_JSR82_RFC_RECEIVE_DATA_CFM	See section 4.49
CSR_BT_JSR82_CLEANUP_REQ	See section 4.50
CSR_BT_JSR82_SET_EVENT_MASK_REQ	See section 4.51
CSR_BT_JSR82_SET_EVENT_MASK_CFM	See section 4.51
CSR_BT_JSR82_RFC_CLOSE_IND	See section 4.52
CSR_BT_JSR82_L2CA_CLOSE_IND	See section 4.53

Table 1: List of all primitives

4.2 CSR_BT_JSR82_DA_START_INQUIRY

Parameters / Primitives	type	appHandle	reqID	iac
CSR_BT_JSR82_DA_START_INQUIRY_REQ	✓	✓	✓	✓

Table 2: CSR_BT_JSR82_DA_START_INQUIRY primitives

Description

Start inquiry of remote Bluetooth devices in the area. Only devices that are visible will respond to this.

The success status of this request is contained in the CSR_BT_JSR82_DL_INQUIRY_COMPLETED_IND signal that comes when the inquiry process has been completed. If not cancelled, the inquiry process will run for 11 seconds.

Parameters

type	Signal identity, CSR_BT_JSR82_DA_START_INQUIRY_REQ.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the CSR_BT_JSR82_DL_DEVICE_DISCOVERED_IND and CSR_BT_JSR82_DL_INQUIRY_COMPLETED _IND signals
iac	The Inquiry Access Code to use. Valid codes are in the range 0x9e8b00 - 0x9e8b3f

The function:

```
void CsrBtJsr82DaStartInquiryReqSend ( CsrSchedQid  appHandle,
                                       CsrUint32       reqID)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_DA_START_INQUIRY_REQ primitive to the Jsr82 task.

4.3 CSR_BT_JSR82_DA_CANCEL_INQUIRY

Parameters / Primitives	type	appHandle	reqID
CSR_BT_JSR82_DA_CANCEL_INQUIRY_REQ	✓	✓	✓

Table 3: CSR_BT_JSR82_DA_CANCEL_INQUIRY primitives

Description

Cancel an ongoing inquiry. The CSR_BT_JSR82_DA_CANCEL_INQUIRY_REQ signal will be answered by a CSR_BT_JSR82_DL_INQUIRY_COMPLETED_IND containing the status of the operation. After receiving the CSR_BT_JSR82_DL_INQUIRY_COMPLETED_IND signal, the application will receive no more CSR_BT_JSR82_DL_DEVICE_DISCOVERED_IND events.

Parameters

type	Signal identity, CSR_BT_JSR82_DA_CANCEL_INQUIRY_REQ.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID of the inquiry to cancel. Note that only an active inquiry can be cancelled. Attempting to cancel a queued inquiry will have no effect.

The function:

```
void CsrBtJsr82DaCancelInquiryReqSend (   CsrSchedQid   appHandle,
                                          CsrUint32     reqID)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_DA_CANCEL_INQUIRY_REQ primitive to the Jsr82 task.

4.4 CSR_BT_JSR82_DA_SEARCH_SERVICES

Primitives \ Parameters	type	appHandle	reqID	deviceAddr	uuidSetLength	*uuidSet	attrSetLength	*attrSet
CSR_BT_JSR82_DA_SEARCH_SERVICES_REQ	✓	✓	✓	✓	✓	✓	✓	✓

Table 4: CSR_BT_JSR82_DA_SEARCH_SERVICES primitives

Description

Initiate the process of searching for services on a remote device. In addition, a number of service attributes may be retrieved. The confirm signal indicates that the service search has started. A service search may be cancelled using the CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH signal. When the search procedure finishes, the calling task will receive a CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED_IND, with a code indicating the result.

Parameters

type	Signal identity, CSR_BT_JSR82_DA_SEARCH_SERVICES_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification, passed down from Java, and echoed back in the confirm signal.
deviceAddr	The Bluetooth address of the remote device.
uuidSetLength	The length in bytes of the uuidSet data element sequence, including the sequence header.
*uuidSet	A list of services to search for. This is a data element sequence.
attrSetLength	The number of attributes to retrieve from the service records.
*attrSet	A list of attributes to retrieve from the service records. Please note that IDs 0x0000-0x0004 are retrieved no matter what this list contains. Also note that any duplicates will be removed from this set, which means that the return value will only contain one instance of each attribute ID/value pair.

The function:

```
void CsrBtJsr82DaSearchServicesReqSend(   CsrSchedQid  appHandle,
                                          CsrUint32       reqID,
                                          deviceAddr_t deviceAddr,
                                          CsrUint16       uuidSetLength,
```

```
                                    CsrUint8  *uuidSet,
                                    CsrUint16    attrSetLength,
                                    CsrUint16    *attrSet)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_DA_SEARCH_SERVICES_REQ primitive to the Jsr82 task.

4.5 CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH

Parameters / Primitives	type	appHandle	reqID
CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH_REQ	✓	✓	✓

Table 5: CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH primitives

Description

Cancel a service search process. This will trigger a CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED_IND, with a result code indicating that the search has been cancelled.

Parameters

type	Signal identity, CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH_REQ.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID of the service search to cancel.

The function:

```
void CsrBtJsr82DaCancelServiceSearchReqSend( CsrSchedQid  appHandle,
                                             CsrUint32       reqID);
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH_REQ primitive to the Jsr82 task.

4.6 CSR_BT_JSR82_DA_RETRIEVE_DEVICES

Primitives \ Parameters	type	appHandle	reqID	option	devicesCount	*devices
CSR_BT_JSR82_DA_RETRIEVE_DEVICES_REQ	✓	✓	✓	✓		
CSR_BT_JSR82_DA_RETRIEVE_DEVICES_CFM	✓		✓		✓	✓

Table 6: CSR_BT_JSR82_DA_RETRIEVE_DEVICES primitives

Description

Retrieve a list of known devices. The maximum number of devices returned in the confirm signal depends on the CSR_BT_JSR82_MAX_BYTES_IN_SD_CACHE_READ, found in csr_bt_usr_config.h.

Parameters

type	Signal identity, CSR_BT_JSR82_DA_ RETRIEVE_DEVICES_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification, passed down from Java, and echoed back in the confirm signal.
option	The type of list to retrieve. This can be: **CSR_BT_JSR82_DISCOVERY_AGENT_CACHED**: The devices returned result from previous inquiry processes. **CSR_BT_JSR82_DISCOVERY_AGENT_PAIRED**: The devices returned are devices that are pre-known to the local device. In this implementation, this resembles the devices that are paired in the SC.
devicesCount	The number of devices returned.
*devices	A list of device addresses.

The function:

```
void CsrBtJsr82DaRetrieveDevicesReqSend ( CsrSchedQid  appHandle,
                                          CsrUint32    reqID,
                                          CsrUint8  option  );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_DA_RETRIEVE_DEVICES_REQ primitive to the Jsr82 task.

4.7 CSR_BT_JSR82_DA_SELECT_SERVICE

Parameters / Primitives	type	appHandle	reqID	requestedUuidLength	*requestedUuid	deviceAddr	protocol	chanPsm	resultCode
CSR_BT_JSR82_DA_SELECT_SERVICE_REQ	✓	✓	✓	✓	✓				
CSR_BT_JSR82_DA_SELECT_SERVICE_CFM	✓		✓			✓	✓	✓	✓

Table 7: CSR_BT_JSR82_DA_SELECT_SERVICE primitives

Description

Attempt to locate a service on any device in the area. The process can potentially take a long time, as it starts with an 11 second long inquiry followed by service search on the found devices until the service has been found.

Parameters

type	Signal identity, CSR_BT_JSR82_DA_ SELECT_SERVICE_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification, passed down from Java, and echoed back in the confirm signal.
requestedUuidLength	The length of the UUID in bytes.
*requestedUuid	The UUID to search for. This is a simple byte array of the UUID in big-endian format. The search will match this UUID against the ServiceClassIDList of the service record when searching for the service.
deviceAddr	The Bluetooth address of the device providing the service.
protocol	The protocol of the service. The values are defined in csr_bt_jsr82_prim.h and can be: **CSR_BT_JSR82_PROTOCOL_L2CAP**: The service uses the L2CAP protocol and chanPsm contains a PSM value. **CSR_BT_JSR82_PROTOCOL_RFCOMM**: The service uses the RFCOMM protocol and chanPsm contains a server channel.
chanPsm	Contains information needed to connect to the found service. Depending on the contents of protocol, this is either a PSM or a server channel.
resultCode	The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82DaSelectServiceReqSend (   CsrSchedQid   appHandle,

                                          CsrUint32        reqID,

                                          CsrUint16
                                    requestedUuidLength

                                          CsrUint8      *requestedUuid);
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_DA_SELECT_SERVICE_REQ primitive to the Jsr82 task.

4.8 CSR_BT_JSR82_DL_DEVICE_DISCOVERED

Parameters / Primitives	type	reqID	deviceAddr	classOfDevice
CSR_BT_JSR82_DL_DEVICE_DISCOVERED_IND	✓	✓	✓	✓

Table 8: CSR_BT_JSR82_DL_DEVICE_DISCOVERED primitives

Description

This signal indicates that a device has responded to the inquiry. It includes necessary information for creating a RemoteDevice object.

This primitive is subject to change as soon as the SD module is implemented.

Parameters

type	Signal identity, CSR_BT_JSR82_DL_DEVICE_DISCOVERED_IND.
reqID	Request ID of the CSR_BT_JSR82_START_INQUIRY_REQ signal that started the current inquiry.
deviceAddr	The address of the discovered device.
classOfDevice	The Bluetooth class of device for the remote device.
rssi	The inquiry RSSI value.

4.9 CSR_BT_JSR82_DL_INQUIRY_COMPLETED

Primitives \ Parameters	type	reqID	respCode
CSR_BT_JSR82_DL_INQUIRY_COMPLETED_IND	✓	✓	✓

Table 9: CSR_BT_JSR82_DL_INQUIRY_COMPLETED primitives

Description

Indicate that the inquiry process is complete.

Parameters

type	Signal identity, CSR_BT_JSR82_DL_INQUIRY_COMPLETED_IND.
reqID	Request ID of the CSR_BT_JSR82_START_INQUIRY_REQ signal that started the inquiry that has been completed.
respCode	Java specific result code.

4.10 CSR_BT_JSR82_DL_SERVICES_DISCOVERED

Parameters / Primitives	type	reqID	atributesLength	*attributes
CSR_BT_JSR82_DL_SERVICES_DISCOVERED_IND	✓	✓	✓	✓

Table 10: CSR_BT_JSR82_DL_SERVICES_DISCOVERED primitives

Description

Indicate that a service of the requested type has been discovered on the remote device. Note that any attributes that could not be retrieved are left out of the attributes list.

Parameters

type	Signal identity, CSR_BT_JSR82_DL_SERVICES_DISCOVERED_IND.
reqID	Request ID of the service search running.
attributesLength	The number of bytes in the *attributes list.
*attributes	A data element list of the retrieved attributes.

4.11 CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED

Primitives \ Parameters	type	reqID	serviceDataBaseState	serviceDBStateValid	respCode
CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED_IND	✓	✓	✓	✓	✓

Table 11: CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED primitives

Description

Indicate that a service search operation has been completed.

Parameters

type — Signal identity, CSR_BT_JSR82_DL_SERVICE_SEARCH_COMPLETED_IND.

reqID — Request ID of the service search that is completed.

serviceDataBaseState — The state of the service database on the remote device.

serviceDBStateValid — Indicates if serviceDataBaseState was available on the remote device.

respCode — Contains the status of the completed service search. The values are defined in csr_bt_jsr82_prim.h, and can be:

CSR_BT_JSR82_SERVICE_SEARCH_COMPLETED: The service search completed normally.

CSR_BT_JSR82_SERVICE_SEARCH_TERMINATED: The service search was cancelled by a CSR_BT_JSR82_DA_CANCEL_SERVICE_SEARCH_REQ signal.

CSR_BT_JSR82_SERVICE_SEARCH_ERROR: An error occurred.

CSR_BT_JSR82_SERVICE_SEARCH_NO_RECORDS: No records corresponding to the requested UUID(s) were found during the service search.

CSR_BT_JSR82_SERVICE_SEARCH_DEVICE_NOT_REACHABLE: The requested device could not be contacted.

4.12 CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS

Parameters / Primitives	type	appHandle	reqID	deviceAddr
CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS_REQ	✓	✓	✓	
CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS_CFM	✓		✓	✓

Table 12: CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS primitives

Description

Read the local Bluetooth address.

Parameters

type	Signal identity, CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification, passed down from Java, and echoed back in the confirm signal.
deviceAddr	The Bluetooth address of the local device.

The function:

```
void CsrBtJsr82LdGetBluetoothAddressReqSend (CsrSchedQid  appHhandle,
                                             CsrUint32       reqID    );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_GET_BLUETOOTH_ADDRESS_REQ primitive to the Jsr82 task.

4.13 CSR_BT_JSR82_LD_GET_PROPERTY

Parameters / Primitives	type	appHandle	reqID	propertyName	propertyValue
CSR_BT_JSR82_LD_GET_PROPERTY_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_LD_GET_PROPERTY_CFM	✓		✓	✓	✓

Table 13: CSR_BT_JSR82_LD_GET_PROPERTY primitives

Description

Retrieves named Bluetooth system property values.

Parameters

type — The signal identity, CSR_BT_JSR82_LD_GET_PROPERTY_REQ/CFM.

appHandle — The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.

reqID — Request identification passed down from Java, and echoed back in the confirm signal.

propertyName — Identifies the property to read. The following properties are supported:

CSR_BT_BLUETOOTH_API_VERSION: The version of the JSR-82 API that is supported. The reply will contain the major version in the upper 8 bits, and the minor version in the lower 8 bits. Example: For version 1.0 this will be the value 0x0100.

CSR_BT_BLUETOOTH_MASTER_SWITCH: Indicates whether master/slave switch is allowed. Value is 0x0000 for “false” and value is 0x0001 for “true”.

CSR_BT_BLUETOOTH_SD_ATTR_RETRIEVABLE_MAX: Maximum number of attributes to be retrieved per service record.

CSR_BT_BLUETOOTH_CONNECTED_DEVICES_MAX: The maximum number of connected devices supported. This may be larger than 7 if the implementation handles parked connections.

CSR_BT_BLUETOOTH_L2CAP_RECEIVEMTU_MAX: The maximum ReceiveMTU size in bytes supported in L2CAP.

CSR_BT_BLUETOOTH_SD_TRANS_MAX: Maximum number of concurrent service discovery transactions.

CSR_BT_BLUETOOTH_CONNECTED_INQUIRY_SCAN: Indicates whether Inquiry scanning is allowed during connection. Value is 0x0000 for “false” and value is

0x0001 for “true”.

CSR_BT_BLUETOOTH_CONNECTED_PAGE_SCAN: Indicates whether Page scanning is allowed during connection. Value is 0x0000 for “false” and value is 0x0001 for “true”.

CSR_BT_BLUETOOTH_CONNECTED_INQUIRY: Indicates whether Inquiry is allowed during connection. Value is 0x0000 for “false” and value is 0x0001 for “true”.

CSR_BT_BLUETOOTH_CONNECTED_PAGE: Indicates whether Paging is allowed during connection. In other words, can a connection be established while being connected to another device? Value is 0x0000 for “false” and value is 0x0001 for “true”.

If the JSR82 layer does not recognize the requested property, it will return **CSR_BT_JSR82_ILLEGAL_PROPERTY_NAME** in this field of the confirm message. The propertyValue parameter will be set to 0 in this case.

propertyValue	A 16 bit numeric representation of the requested value. Values are outlined above.

The function:

```
void CsrBtJsr82LdGetPropertyReqSend (     CsrSchedQid   appHhandle,
                                          CsrUint32        reqID     );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_GET_PROPERTY _REQ primitive to the Jsr82 task.

The JSR-82 profile property don’t support any obex feature.

4.14 CSR_BT_JSR82_LD_SET_DISCOVERABLE

Parameters / Primitives	type	appHandle	reqID	mode	resultCode
CSR_BT_JSR82_LD_SET_DISCOVERABLE_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_LD_SET_DISCOVERABLE_CFM	✓		✓		✓

Table 14: CSR_BT_JSR82_LD_SET_DISCOVERABLE primitives

Description

Set the discoverable status of the local Bluetooth device. This determines if the device will respond to inquiry requests from remote devices.

Note that the confirm signal only indicates error if the requested mode is outside the allowed range (see below). Otherwise, the operation is considered successful.

Parameters

type — The signal identity, CSR_BT_JSR82_LD_SET_DISCOVERABLE_REQ/CFM.

appHandle — The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.

reqID — Request identification passed down from Java, and echoed back in the confirm signal

mode — The mode to set the device in. Possible values are:

CSR_BT_DISCOVERY_AGENT_GIAC (0x9E8B33)

CSR_BT_DISCOVERY_AGENT_LIAC (0x9E8B00) (not supported)

CSR_BT_DISCOVERY_AGENT_NOT_DISCOVERABLE (0x00)

Or any value in the range 0x9E8B00 to 0x9E8B3F (not supported)

If the mode is set to CSR_BT_DISCOVERY_AGENT_NOT_DISCOVERABLE, this affects **all** active applications, including non-Java profiles.

resultCode — The result of the operation. Values are defined in profiles.h, and include:

CSR_BT_RESULT_CODE_JSR82_SUCCESS: The requested mode change was successful.
CSR_BT_UNSUPPORTED_FEATURE: Requested mode is in the discoverable range, but not supported by CSR Synergy Bluetooth.
CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED: Requested mode is outside the legal range.

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82LdSetDiscoverableReqSend( CsrSchedQid  appHandle,
```

```
                                    CsrUint32       reqID,
                                    CsrUint24       mode    );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_SET_DISCOVERABLE _REQ primitive to the Jsr82 task.

4.15 CSR_BT_JSR82_LD_GET_DISCOVERABLE

Parameters / Primitives	type	appHandle	reqID	mode
CSR_BT_JSR82_LD_GET_DISCOVERABLE_REQ	✓	✓	✓	
CSR_BT_JSR82_LD_GET_DISCOVERABLE_CFM	✓		✓	✓

Table 15: CSR_BT_JSR82_LD_GET_DISCOVERABLE primitives

Description

Retrieve the discoverable status of the local Bluetooth device. This determines if the device will respond to inquiry requests from remote devices.

Parameters

Type	The signal identity, CSR_BT_JSR82_LD_GET_DISCOVERABLE_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
Mode	The mode the device is in. Possible values are: **CSR_BT_DISCOVERY_AGENT_GIAC (0x9E8B33)** **CSR_BT_DISCOVERY_AGENT_LIAC (0x9E8B00)** **CSR_BT_DISCOVERY_AGENT_NOT_DISCOVERABLE (0x00)** **Or any value in the range 0x9E8B00 to 0x9E8B3F**

The function:

```
void CsrBtJsr82LdGetDiscoverableReqSend( CsrSchedQid  appHandle,
                                         CsrUint32       reqID);
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_GET_DISCOVERABLE _REQ primitive to the Jsr82 task.

4.16 CSR_BT_JSR82_LD_GET_FRIENDLY_NAME

Parameters / Primitives	type	appHandle	reqID	localName
CSR_BT_JSR82_LD_GET_FRIENDLY_NAME_REQ	✓	✓	✓	
CSR_BT_JSR82_LD_GET_FRIENDLY_NAME_CFM	✓		✓	✓

Table 16: CSR_BT_JSR82_LD_GET_FRIENDLY_NAME primitives

Description

Read the friendly name of the local device.

Parameters

Type	The signal identity, CSR_BT_JSR82_LD_GET_FRIENDLY_NAME_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
localName	A pointer to a (zero-terminated) string containing the name of the local device.

The function:

```
void CsrBtJsr82LdGetFriendlyNameSend(     CsrSchedQid   appHandle,
                                          CsrUint32        reqID     );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_GET_FRIENDLY_NAME_REQ primitive to the Jsr82 task.

4.17 CSR_BT_JSR82_LD_UPDATE_RECORD

Parameters / Primitives	type	appHandle	reqID	*serviceRecord	serviceRecordSize	serviceRecHandle	resultCode
CSR_BT_JSR82_LD_UPDATE_RECORD_REQ	✓	✓	✓	✓	✓	✓	
CSR_BT_JSR82_LD_UPDATE_RECORD_CFM	✓		✓			✓	✓

Table 17: CSR_BT_JSR82_LD_UPDATE_RECORD primitives

Description

Updates a service record, registered in the service database of the local device.

Parameters

Type — The signal identity, CSR_BT_JSR82_LD_UPDATE_RECORD_REQ/CFM.

appHandle — The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.

reqID — Request identification passed down from Java, and echoed back in the confirm signal.

*serviceRecord — The service record which is a list of attribute IDs and values.

Note that when the serviceRecord has been processed, it will be freed, and the pointer is no longer valid for use in the requesting application.

serviceRecordSize — The number of bytes in the service record, in the range 0x000A to 0xFFFF.

serviceRecHandle — The service record handle that uniquely identifies each service record. It is a 32-bit value in the range 0x00010000– 0xFFFFFFFF.

resultCode — Reports the result when updating the service record. The result values can be:

CSR_BT_RESULT_CODE_JSR82_SUCCESS: The service record was updated successfully.

CSR_BT_JSR82_LD_UPDATE_RECORD_ERROR: There was an error. The service record has not been changed.

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82LdUpdateRecordReqSend (CsrSchedQid   appHandle,
                                      CsrUint32     reqID,
                                      CsrUint32     serviceRecHandle,
```

```
                                        CsrUint8  * serviceRecord,
                                        CsrUint16    serviceRecordSize);
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_UPDATE_RECORD_REQ primitive to the Jsr82 task.

4.18 CSR_BT_JSR82_LD_GET_DEVICE_CLASS

Primitives \ Parameters	type	appHandle	reqID	fieldsMask	deviceClass	resultCode
CSR_BT_JSR82_LD_GET_DEVICE_CLASS_REQ	✓	✓	✓	✓		
CSR_BT_JSR82_LD_GET_DEVICE_CLASS_CFM	✓		✓		✓	✓

Table 18: CSR_BT_JSR82_LD_GET_DEVICE_CLASS primitives

Description

Request the class of device for the local device.

Parameters

type — The signal identity, CSR_BT_JSR82_LD_GET_DEVICE_CLASS _REQ/CFM.

appHandle — The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.

reqID — Request identification passed down from Java, and echoed back in the confirm signal.

fieldsMask — This is a bitwise OR of the following values, specifying which fields are requested:

CSR_BT_JSR82_COD_MAJOR: Request the major device class bits

CSR_BT_JSR82_COD_MINOR: Request the minor device class bits

CSR_BT_JSR82_COD_SERVICES: Request the service classes bits

CSR_BT_JSR82_COD_FULL: All of the above

deviceClass — The class of device for the local device. Only the requested fields are valid.

resultCode — Reports the result of reading the class of device for the local device. The result values are defined in csr_bt_profiles.h and can be:

CSR_BT_RESULT_CODE_JSR82_SUCCESS: The class of device is read with success.

CSR_BT_HARDWARE_ERROR: The class of device could not be read because of a hardware error.

CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED: Indicates that the local device is in a state where the class of device could not be read.

CSR_BT_RESULT_CODE_JSR82_UNSPECIFIED_ERROR: Indicates that no other error codes specified in csr_bt_profiles.h are appropriate to use.

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82LdGetDeviceClassReqSend(   CsrSchedQid   appHandle,
                                          CsrUint32     reqID,
                                          CsrUint8  fieldsMask  );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_GET_DEVICE_CLASS_REQ primitive to the Jsr82 task.

4.19 CSR_BT_JSR82_LD_SET_DEVICE_CLASS

Primitives \ Parameters	type	appHandle	reqID	deviceClass	resultCode
CSR_BT_JSR82_LD_SET_DEVICE_CLASS_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_LD_SET_DEVICE_CLASS_CFM	✓		✓		✓

Table 19: CSR_BT_JSR82_LD_SET_DEVICE_CLASS primitives

Description

Write the class of device for the local device. Only the service classes bits are affected by this operation.

Parameters

Type	The signal identity, CSR_BT_JSR82_LD_SET_DEVICE_CLASS _REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
deviceClass	The class of device to write to the local device.
resultCode	Reports the result of writing the class of device to the local device. The result values are defined in csr_bt_profiles.h and can be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS**: The class of device is written with success. **CSR_BT_HARDWARE_ERROR**: The class of device could not be written because of a hardware error. **CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED**: Indicates that the local device is in a state where the class of device could not be written. **CSR_BT_RESULT_CODE_JSR82_UNSPECIFIED_ERROR**: Indicates that no other error codes specified in csr_bt_profiles.h are appropriate to use. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82LdSetDeviceClassReqSend(   CsrSchedQid      appHandle,
                                          CsrUint32           reqID,
                                          device_class_t   deviceClass  );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_SET_DEVICE_CLASS_REQ primitive to the Jsr82 task.

4.20 CSR_BT_JSR82_LD_GET_SECURITY_LEVEL

Parameters / Primitives	type	appHandle	reqID	secLevel
CSR_BT_JSR82_LD_GET_SECURITY_LEVEL_REQ	✓	✓	✓	
CSR_BT_JSR82_LD_GET_SECURITY_LEVEL_CFM	✓		✓	✓

Table 20: CSR_BT_JSR82_LD_GET_SECURITY_LEVEL primitives

Description

Request the security level for the local device. **Note:** This signal pair reports the default security level. If other (non-Java) applications change the security level of the local device at run-time, this will not be reflected in the result.

Parameters

Type	The signal identity, CSR_BT_JSR82_LD_GET_SECURITY_LEVEL_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
secvLevel	The security level of the local service. This is one of the following: **CSR_BT_SEC_MODE0_OFF**: Security is disabled. **CSR_BT_SEC_MODE1_NON_SECURE**: Security level 1 (No security) **CSR_BT_SEC_MODE2_SERVICE**: Security level 2 (Service level security) **CSR_BT_SEC_MODE3_LINK**: Security level 3 (Link level security) **CSR_BT_SEC_MODE4_SSP**: Security level 4 (Service level security)

The function:

```
void CsrBtJsr82LdGetSecurityLevelReqSend(  CsrSchedQid  appHandle,
                                           CsrUint32    reqID     );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_GET_SECURITY_LEVEL_REQ primitive to the Jsr82 task.

4.21 CSR_BT_JSR82_LD_IS_MASTER

Primitives \ Parameters	type	appHandle	reqID	deviceAddr	role
CSR_BT_JSR82_LD_IS_MASTER_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_LD_IS_MASTER_CFM	✓		✓		✓

Table 21: CSR_BT_JSR82_LD_IS_MASTER primitives

Description

Request the master/slave role of the local device.

Parameters

Type	The signal identity, CSR_BT_JSR82_LD_IS_MASTER_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
deviceAddr	Address of the remote device for which to check the master/slave relationship.
Role	Result of the request. Values are defined in profiles.h, and may be: **MASTER_ROLE** **SLAVE_ROLE** **UNDEFINED_ROLE:** The operation failed for some reason.

The function:

```
void CsrBtJsr82LdIsMasterReqSend(  CsrSchedQid   appHandle,
                                   CsrUint32     reqID,
                                   deviceAddr_t deviceAddr   );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_LD_IS_MASTER_REQ primitive to the Jsr82 task.

4.22 CSR_BT_JSR82_SR_CREATE_RECORD

Parameters / Primitives	type	appHandle	reqID	serviceRecHandle
CSR_BT_JSR82_SR_CREATE_RECORD_REQ	✓	✓	✓	
CSR_BT_JSR82_SR_CREATE_RECORD_CFM	✓		✓	✓

Table 22: CSR_BT_JSR82_SR_CREATE_RECORD primitives

Description

Reserve a serviceRecordHandle in the SDP layer.

Parameters

Type	The signal identity, CSR_BT_JSR82_SR_CREATE_RECORD_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
serviceRecHandle	The service record handle that uniquely identifies each service record. It is a 32-bit value in the range 0x00010000– 0xFFFFFFFF.

The function:

```
void CsrBtJsr82SrCreateRecordReqSend(     CsrSchedQid   appHandle,
                                          CsrUint32        reqID    );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_SR_CREATE_RECORD_REQ primitive to the Jsr82 task.

4.23 CSR_BT_JSR82_SR_REGISTER_RECORD

Primitives \ Parameters	type	appHandle	reqID	serviceRecordLength	*serviceRecord	serviceRecHandle
CSR_BT_JSR82_SR_REGISTER_RECORD_REQ	✓	✓	✓	✓	✓	
CSR_BT_JSR82_SR_REGISTER_RECORD_CFM	✓		✓			✓

Table 23: CSR_BT_JSR82_SR_REGISTER_RECORD primitives

Description

Register a service record in the SDP layer. The service record must contain the serviceRecHandle from CSR_BT_JSR82_SR_CREATE_RECORD.

Parameters

Type	The signal identity, CSR_BT_JSR82_SR_REGISTER_RECORD_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
serviceRecordLength	The size of the service record in bytes.
*serviceRecord	The service record which is a list of attribute IDs and values. Please note that when the serviceRecord has been processed it is up to the recipient to handle the serviceRecord.
serviceRecHandle	The service record handle that uniquely identifies each service record. It is a 32-bit value in the range 0x00010000– 0xFFFFFFFF. A serviceRecHandle of 0 indicates an error. Either, the SDDB was locked by another process, or the serviceRecord is invalid.

The function:

```
void CsrBtJsr82SrRegisterRecordReqSend(   CsrSchedQid   appHandle,

                                          CsrUint32        reqID,

                                          CsrUint16
                                          serviceRecordLength,

                                          CsrUint8     *serviceRecord  );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_SR_CREATE_RECORD_REQ primitive to the Jsr82 task.

4.24 CSR_BT_JSR82_SR_REMOVE_RECORD

Parameters / Primitives	type	appHandle	reqID	serviceRecHandle
CSR_BT_JSR82_SR_REMOVE_RECORD_REQ	✓	✓	✓	✓
CSR_BT_JSR82_SR_REMOVE_RECORD _CFM	✓		✓	✓

Table 24: CSR_BT_JSR82_SR_REMOVE_RECORD primitives

Description

Remove a service record from the SDP layer.

Parameters

Type	The signal identity, CSR_BT_JSR82_SR_REMOVE_RECORD_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
serviceRecHandle	The service record handle that uniquely identifies each service record. It is a 32-bit value in the range 0x00010000– 0xFFFFFFFF. A serviceRecHandle of 0 in the return signal indicates an error. Either, the SDDB was locked by another process, or the serviceRecord is invalid.

The function:

```
void CsrBtJsr82SrRemoveRecordReqSend( CsrSchedQid   appHandle,
                                      CsrUint32     reqID,
                                      CsrUint32     serviceRecHandle  );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_SR_REMOVE_RECORD_REQ primitive to the Jsr82 task.

4.25 CSR_BT_JSR82_SR_POPULATE_RECORD

Primitives \ Parameters	type	appHandle	reqID	deviceAddr	serviceDataBaseState	serviceRecHandle	attrSetLength	*attrSet	attributesLength	*attributes	resultCode
CSR_BT_JSR82_SR_POPULATE_RECORD_REQ	✓	✓	✓	✓	✓	✓	✓	✓			
CSR_BT_JSR82_SR_POPULATE_RECORD_CFM	✓		✓						✓	✓	✓

Table 25: CSR_BT_JSR82_SR_POPULATE_RECORD primitives

Description

Retrieves attributes for a service record found in a previous search. **Note:** If the validity of the serviceRecordHandle can no longer be guaranteed, the confirm signal will return empty (attributesLength = 0, attributes = NULL). This is the case if the remote SDP server does not provide the serviceDataBaseState attribute, or if this attribute has changed. Consider the CSR_BT_JSR82_DA_SEARCH_SERVICES signal as an alternative.

Parameters

Type	The signal identity, CSR_BT_JSR82_SR_POPULATE_RECORD_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request identification passed down from Java, and echoed back in the confirm signal.
deviceAddr	The Bluetooth address of the remote device.
serviceDataBaseState	The state of the remote server when the service record in question was discovered. This is compared to a 32-bit attribute found on the remote SDP server.
serviceRecHandle	The service record handle that uniquely identifies each service record. It is a 32-bit value in the range 0x00010000– 0xFFFFFFFF.
attrSetLength	The number of attributes to retrieve.
*attrSet	A list of the attributes to retrieve. Note that any duplicates will be removed from this set, which means that the return value will only contain one instance of each attribute ID/value pair.
attributesLength	The size of the attributes list in bytes.
*attributes	A list of attribute IDs and values corresponding to the attribute IDs.
resultCode	Result of the operation. Values are defined in the profiles.h file, and may be:

CSR_BT_RESULT_CODE_JSR82_SUCCESS

CSR_BT_RESULT_CODE_JSR82_SDC_CONNECTION_RESPONSE_ERROR – connection to remote device failed

CSR_BT_RESULT_CODE_JSR82_SDC_DATABASE_STATE_CHANGED – service records on remote device may not be valid, and consequently have not been read.

CSR_BT_RESULT_CODE_JSR82_CSR_BT_SDC_EMPTY_RESPONSE_DATA_ERROR – no attributes could be retrieved

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82SrPopulateRecordReqSendt(   CsrSchedQid  appHandle,

                                           CsrUint32    reqID,

                                           deviceAddr_t    deviceAddr,

                                           CsrUint32    serviceRecHandle,
                                           CsrUint32    serviceDataBaseState,
                                           CsrUint16    attrSetLength,

                                           CsrUint16    *attrSet  );
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_SR_POPULATE_RECORD_REQ primitive to the Jsr82 task.

4.26 CSR_BT_JSR82_L2CA_GET_PSM

Parameters / Primitives	type	appHandle	reqID	localPsm
CSR_BT_JSR82_L2CA_GET_PSM_REQ	✓	✓	✓	
CSR_BT_JSR82_L2CA_GET_PSM_CFM	✓		✓	✓

Table 26: CSR_BT_JSR82_L2CA_GET_PSM primitives

Description

Request a PSM for JSR82. This reserves a local PSM for the requesting application. This is needed for a server application before it can accept incoming connections. The client-side PSM is handled internally in the JSR82 layer, and does not need this signal. Note that the maximum number of reserved PSMs is limited by CSR_BT_JSR82_MAX_ALLOCATED_PSMS as defined in usr_config.h. On RFCOMM builds of CSR Synergy Bluetooth, the total number of available PSMs is limited by the firmware of the BlueCore chip, and increasing the CSR_BT_JSR82_MAX_ALLOCATED_PSMS definition may have no effect. **Note:** If a PSM has been reserved, and no connection is made subsequently, the user must use CSR_BT_JSR82_L2CA_DISCONNECT to free the reserved PSM.

Parameters

Type	The signal identity, CSR_BT_JSR82_L2CA_GET_PSM_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localPsm	The PSM that was reserved. A return value of zero indicates that the operation has failed.

The function:

```
void CsrBtJsr82L2caGetPsmReqSend ( CsrSchedQid  appHandle,
                                   CsrUint32       reqID     )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_GET_PSM_REQ primitive to the Jsr82 task.

4.27 CSR_BT_JSR82_L2CA_ACCEPT

Primitives \ Parameters	type	appHandle	reqID	localPsm	btConnId	receiveMtu	transmitMtu	attrs	deviceAddr	resultCode
CSR_BT_JSR82_L2CA_ACCEPT_REQ	✓	✓	✓	✓		✓	✓	✓		
CSR_BT_JSR82_L2CA_ACCEPT_CFM	✓		✓	✓	✓		✓		✓	✓

Table 27: CSR_BT_JSR82_L2CA_ACCEPT primitives

Description

Start accepting incoming L2CAP connections on localPsm.

The transmitMtu value returned is the negotiated MTU for the remote device. **Note:** If the process fails, it is the responsibility of the user to send a CSR_BT_JSR82_L2CA_DISCONNECT_REQ signal to the JSR82 layer, in order to free the previously reserved PSM.

Parameters

Type	The signal identity, CSR_BT_JSR82_L2CA_ACCEPT_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localPsm	The PSM to accept connections for. This PSM must be the result of a previous CSR_BT_JSR82_L2CA_GET_PSM procedure.
btConnId	The BT connection ID of the incoming connection. This must be used as an identifier for operations on the connection.
receiveMtu	The receive MTU that the local device supports.
transmitMtu	In the downstream signal, this is the suggested MTU for transmissions from the local device. In the upstream signal, it is the negotiated transmitMtu. Any data transmissions will not exceed this.
Attrs	This is a bitwise OR'ed pattern of the flags CSR_BT_JSR82_AUTHORIZE, CSR_BT_JSR82_AUTHENTICATE, CSR_BT_JSR82_ENCRYPT, and CSR_BT_JSR82_MASTER, defined in csr_bt_jsr82_prim.h. Please note that if both the local and the remote device are using security level 4 (**CSR_BT_SEC_MODE4_SSP)**, Bluetooth authentication and encryption are always required, even if it is not set by the application.
deviceAddr	The Bluetooth address of the connecting device.

resultCode	The result of the process. This can be one of the following codes: **CSR_BT_RESULT_CODE_JSR82_SUCCESS**: A device connected successfully. **CSR_BT_RESULT_CODE_JSR82_SECURITY_FAIL**: The security settings are incompatible with the security level of the local device. **CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED**: The command cannot be processed by the JSR82 low-level profile at this time. A likely reason is that localPsm has not been previously reserved by CSR_BT_JSR82_L2CA_GET_PSM. **CSR_BT_CONNECT_ATTEMPT_FAILED**: The connection could not be established. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82L2caAcceptSend (    CsrSchedQid  appHandle,
                                   CsrUint32        reqID,
                                   psm_t        localPsm,
                                   mtu_t        receiveMtu,
                                   mtu_t        transmitMtu,
                                   CsrUint32       attrs    )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_ACCEPT_REQ primitive to the Jsr82 task.

4.28 CSR_BT_JSR82_L2CA_OPEN

Parameters / Primitives	type	appHandle	reqID	deviceAddr	remotePsm	receiveMtu	transmitMtu	attrs	btConnId	resultCode
CSR_BT_JSR82_L2CA_OPEN_REQ	✓	✓	✓	✓	✓	✓	✓	✓		
CSR_BT_JSR82_L2CA_OPEN_CFM	✓		✓	✓			✓		✓	✓

Table 28: CSR_BT_JSR82_L2CA_OPEN primitives

Description

Connect to a remote Bluetooth device through L2CAP.

Parameters

Type	The signal identity, CSR_BT_JSR82_L2CA_OPEN_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
deviceAddr	The Bluetooth address of the remote device.
remotePsm	The PSM to connect to on the remote device.
receiveMtu	The receive MTU that the local device supports.
transmitMtu	In the downstream signal, this is the suggested MTU for transmissions from the local device. In the upstream signal, it is the negotiated transmitMtu. Any data transmissions will not exceed this.
Attrs	This is a bitwise OR'ed pattern of the flags CSR_BT_JSR82_AUTHENTICATE, CSR_BT_JSR82_ENCRYPT, and CSR_BT_JSR82_MASTER, defined in csr_bt_jsr82_prim.h. Please note that if both the local and the remote device are using security level 4 (**CSR_BT_SEC_MODE4_SSP)**, Bluetooth authentication and encryption are always required, even if it is not set by the application.
btConnId	The BT connection ID of the connection. This must be used as an identifier for operations on the connection.
resultCode	The result of the process. This can be one of the following codes: **CSR_BT_RESULT_CODE_JSR82_SUCCESS**: A device connected successfully. **CSR_BT_RESULT_CODE_JSR82_SECURITY_FAIL**: The security settings are incompatible with the security level of the local device.

CSR_BT_RESULT_CODE_JSR82_UNSPECIFIED_ERROR: An error occurred during the connection.

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82L2caOpenReqSend ( CsrSchedQid  appHandle,
                                 CsrUint32    reqID,
                                 deviceAddr_t    deviceAddr,
                                 psm_t     remotePsm,
                                 mtu_t     receiveMtu,
                                 mtu_t     transmitMtu,
                                 CsrUint32    attrs    )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_OPEN_REQ primitive to the Jsr82 task.

4.29 CSR_BT_JSR82_L2CA_DISCONNECT

Primitives \ Parameters	type	appHandle	reqID	localPsm	resultCode
CSR_BT_JSR82_L2CA_DISCONNECT_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_L2CA_DISCONNECT_CFM	✓		✓	✓	✓

Table 29: CSR_BT_JSR82_L2CA_DISCONNECT primitives

Description

Disconnect all L2CAP connections associated to a PSM and unregister this PSM. This corresponds to calling close on the L2CAPConnectionNotifier object on a server.

Parameters

Type	The signal identity, CSR_BT_JSR82_L2CA_DISCONNECT_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localPsm	The PSM for the connection to disconnect.
resultCode	The result of the process. This can be one of the following codes: **CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED**: Trying to disconnect a L2CAP connection that is not up and running. **CSR_BT_ABNORMAL_LINK_DISCONNECT**: Disconnected due to link loss. **CSR_BT_CONNECTION_TERM_BY_REMOTE_HOST**: The L2CAP connection is released from remote device. **CSR_BT_CONNECTION_TERM_BY_LOCAL_HOST**: The L2CAP connection is released from local device as a result of a CSR_BT_JSR82_L2CA_DISCONNECT_REQ message. **CSR_BT_CONNECT_ATTEMPT_FAILED:** A disconnect request was sent to the JSR82 layer before a connection could be completed. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82L2caDisconnectReqSend (CsrSchedQid appHandle, CsrUint32 reqID,
                                      psm_t   localPsm)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_DISCONNECT_REQ primitive to the Jsr82 task.

4.30 CSR_BT_JSR82_L2CA_CLOSE

Primitives \ Parameters	type	appHandle	reqID	btConnId	localTerminated	resultCode
CSR_BT_JSR82_L2CA_CLOSE_REQ	✓	✓	✓	✓		
CSR_BT_JSR82_L2CA_CLOSE_CFM	✓		✓	✓	✓	✓

Table 30: CSR_BT_JSR82_L2CA_CLOSE primitives

Description

End a L2CAP connection with a remote device. On the client side of a connection, this unregisters the associated PSM as well. This signal corresponds to calling close on the L2CAPConnection object for the connection.

Parameters

Type — Signal identity, CSR_BT_JSR82_L2CA_CLOSE_REQ/CFM

appHandle — The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.

reqID — Request ID passed down from Java, and echoed back in the confirm signal.

btConnId — The BT connection ID for the connection to close.

resultCode — Values are defined in profiles.h and can be:

CSR_BT_CONNECTION_TERM_BY_LOCAL_HOST: The local device initiated the disconnect procedure.

CSR_BT_CONNECTION_TERM_BY_REMOTE_HOST: The remote device initiated the disconnect procedure.

CSR_BT_NO_CONNECTION_TO_DEVICE: The connection being cancelled was not found.

CSR_BT_CONNECT_ATTEMPT_FAILED: Disconnected while waiting for a connection.

CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED: Close is called on a connection that is not up and running. This can only come from an error in the middle layers, as the L2CAPConnection object does not exist at this point in a connection.

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

localTerminated — Boolean that indicates whether the close connection has been released by the local device

The function:

```
void CsrBtJsr82L2caCloseReqSend (  CsrSchedQid  appHandle,
```

```
                                    CsrUint32       reqID,
                                    CsrBtConnId     btConnId)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_CLOSE_REQ primitive to the Jsr82 task.

4.31 CSR_BT_JSR82_L2CA_TX_DATA

Parameters / Primitives	type	appHandle	reqID	btConnId	payloadLength	*payload	resultCode
CSR_BT_JSR82_L2CA_TX_DATA_REQ	✓	✓	✓	✓	✓	✓	
CSR_BT_JSR82_L2CA_TX_DATA_CFM	✓		✓	✓			✓

Table 31: CSR_BT_JSR82_L2CA_TX_DATA primitives

Description

Send a L2CAP packet to the remote device.

Parameters

Type	The signal identity, CSR_BT_JSR82_L2CA_TX_DATA_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
btConnId	The BT connection ID of the connection.
payloadLength	The size of the data payload to send. This cannot be 0.
*payload	The data payload to send.
resultCode	The result code of the operation. This can be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS**: The data is sent with success. **CSR_BT_RESULT_CODE_JSR82_DATAWRITE_FAIL**: Indicates that, a Flush timeout has expired, the L2CAP connection is terminated or the L2CAP connection is not up running yet. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82L2caTxDataReqSend ( CsrSchedQid  appHandle,
                                   CsrUint32       reqID,
```

```
                                    CsrBtConnId      btConnId,
                                    CsrUint16        payloadLength,
                                    CsrUint8      *payload    )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_TX_DATA_REQ primitive to the Jsr82 task.

4.32 CSR_BT_JSR82_L2CA_RX_DATA

Primitives \ Parameters	type	appHandle	reqID	btConnId	dataLength	resultCode	payloadLength	*payload
CSR_BT_JSR82_L2CA_RX_DATA_REQ	✓	✓	✓	✓	✓			
CSR_BT_JSR82_L2CA_RX_DATA_CFM	✓		✓	✓		✓	✓	✓

Table 32: CSR_BT_JSR82_L2CA_RX_DATA primitives

Description

Receive a L2CAP packet from the remote device. If there are no packets in the receive buffer, the CFM signal will not appear until a packet has been received. The value CSR_BT_JSR82_MAX_UNPROCESSED_DATA_PACKETS defined in csr_bt_usr_config.h specifies how many packets that can be received in the buffer before the JSR82 layer starts dropping packets.

Parameters

Type	The signal identity, CSR_BT_JSR82_L2CA_RX_DATA_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
btConnId	The BT connection ID of the connection.
dataLength	Data payload length that can be received
resultCode	The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.
payloadLength	The size of the data payload that is received
*payload	The data payload that is received

The function:

```
        void CsrBtJsr82L2caRxDataReqSend (    CsrSchedQid   appHandle,
                                              CsrUint32       reqID,
```

```
                                    CsrBtConnId     btConnId,

                                    CsrUint16    dataLength);
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_RX_DATA_REQ primitive to the Jsr82 task.

4.33 CSR_BT_JSR82_L2CA_RX_READY

Parameters / Primitives	type	appHandle	reqID	btConnId	dataReady	resultCode	dataLength
CSR_BT_JSR82_L2CA_RX_READY_REQ	✓	✓	✓	✓			
CSR_BT_JSR82_L2CA_RX_READY_CFM	✓		✓	✓	✓	✓	✓

Table 33: CSR_BT_JSR82_L2CA_RX_READY primitives

Description

Check the status of the receive buffer for the L2CAP connection corresponding to btConnId.

Parameters

Type	The signal identity, CSR_BT_JSR82_L2CA_RX_READY_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
btConnId	The BT connection ID of the connection.
dataReady	Indicates if one or more data packets have been received on the connection.
resultCode	The result of the operation. Values are defined in profiles.h, and may be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS:** The operation succeeded. **CSR_BT_NO_CONNECTION_TO_DEVICE:** The connection to the remote device has been broken. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.
dataLength	The amount of data available in bytes

The function:

```
void CsrBtJsr82L2caRxReadyReqSend (    CsrSchedQid   appHandle,

                                       CsrUint32     reqID,
```

```
                                        CsrBtConnId  btConnId )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_RX_READY_REQ primitive to the Jsr82 task.

4.34 CSR_BT_JSR82_L2CA_GET_CONFIG

Primitives \ Parameters	type	appHandle	reqID	btConnId	receiveMtu	transmitMtu	resultCode
CSR_BT_JSR82_L2CA_GET_CONFIG_REQ	✓	✓	✓	✓			
CSR_BT_JSR82_L2CA_GET_CONFIG_CFM	✓		✓		✓	✓	✓

Table 34: CSR_BT_JSR82_L2CA_GET_CONFIG primitives

Description

Retrieve configuration for the L2CAP connection corresponding to btConnId.

Parameters

Type	The signal identity, CSR_BT_JSR82_L2CA_GET_CONFIG_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
btConnId	The BT connection ID of the connection.
receiveMtu	The incoming MTU of the connection.
transmitMtu	The outgoing MTU of the connection.
resultCode	The result of the operation. Values are defined in profiles.h, and may be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS:** The operation succeeded. **CSR_BT_NO_CONNECTION_TO_DEVICE:** The connection to the remote device has been broken. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82L2caGetConfigReqSend (     CsrSchedQid   appHandle,
                                          CsrUint32     reqID,
                                          CsrBtConnId      btConnId)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_L2CA_GET_CONFIG_REQ primitive to the Jsr82 task.

4.35 CSR_BT_JSR82_RD_GET_FRIENDLY_NAME

Primitives \ Parameters	type	appHandle	reqID	address	alwaysAsk	remoteName	resultCode
CSR_BT_JSR82_RD_GET_FRIENDLY_NAME_REQ	✓	✓	✓	✓	✓		
CSR_BT_JSR82_RD_GET_FRIENDLY_NAME_CFM	✓		✓			✓	✓

Table 35: CSR_BT_JSR82_RD_GET_FRIENDLY_NAME primitives

Description

Get the friendly name of a remote device.

Parameters

Type	Signal identity, CSR_BT_JSR82_RD_GET_FRIENDLY_NAME_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
Address	The Bluetooth address of the remote device.
alwaysAsk	A Boolean flag indicating if use of cached names is allowed, or the request should always be made via the air interface.
remoteName	A pointer to a string containing the friendly name of the remote device. This string is NULL if the procedure was not successful.
Resultcode	Indicates if the name of the remote device was read successfully. The result values are defined in csr_bt_profiles.h and can be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS**: The remote name was read succesfully. **CSR_BT_RESULT_CODE_JSR82_UNSPECIFIED_ERROR:** The remote name could not be read. E.g. the remote device was not reachable. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82RdGetFriendlyNameReqSend ( CsrSchedQid  appHandle,
                                          CsrUint32       reqID,
                                          deviceAddr_t address,
```

```
                                        CsrBool      alwaysAsk  )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RD_GET_FRIENDLY_NAME_REQ primitive to the Jsr82 task.

4.36 CSR_BT_JSR82_RD_AUTHENTICATE

Parameters / Primitives	type	appHandle	reqID	address	resultCode
CSR_BT_JSR82_RD_AUTHENTICATE_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_RD_AUTHENTICATE_CFM	✓		✓		✓

Table 36: CSR_BT_JSR82_RD_AUTHENTICATE primitives

Description

Request authentication of a remote device. If the remote device has not previously been authenticated, sending this primitive will cause a CSR_BT_SC_PASSKEY_IND to be sent from the security controller to the default application security handler defined in usrconfig.h. If the remote device has been authenticated before, a successful CSR_BT_JSR82_RD_AUTHENTICATE_CFM will be returned.

Parameters

Type	Signal identity, CSR_BT_JSR82_RD_AUTHENTICATE_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
Address	The Bluetooth address of the remote device.
resultCode	Result of the authentication. Values are defined in profiles.h, and may be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS** **CSR_BT_BONDING_FAIL** **CSR_BT_INTERNAL_ERROR** **CSR_BT_BONDING_NOT_ALLOWED** **CSR_BT_TIMEOUT** The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82RdAuthenticateReqSend ( CsrSchedQid   appHandle,
                                       CsrUint32       reqID,
                                       deviceAddr_t address   )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RD_AUTHENTICATE_REQ primitive to the Jsr82 task.

4.37 CSR_BT_JSR82_RD_IS_AUTHENTICATED

Parameters / Primitives	type	appHandle	reqID	deviceAddr	authenticated
CSR_BT_JSR82_RD_IS_AUTHENTICATED_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_RD_IS_AUTHENTICATED_CFM	✓		✓		✓

Table 37: CSR_BT_JSR82_RD_IS_AUTHENTICATED primitives

Description

Request the authentication status of the connection to the remote device.

Parameters

Type	Signal identity, CSR_BT_JSR82_RD_IS_AUTHENTICATED_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
deviceAddr	The Bluetooth address of the remote device.
authenticated	The authentication status. This can be either **TRUE** or **FALSE**. If there are no connections to the device, the status is **FALSE**.

The function:

```
void CsrBtJsr82RdIsAuthenticatedReqSend ( CsrSchedQid  appHandle,
                                          CsrUint32       reqID,
                                          deviceAddr_t address  )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RD_IS_AUTHENTICATED_REQ primitive to the Jsr82 task.

4.38 CSR_BT_JSR82_RD_ENCRYPT

Parameters / Primitives	type	appHandle	reqID	encrypt	address	resultCode
CSR_BT_JSR82_RD_ENCRYPT_REQ	✓	✓	✓	✓	✓	
CSR_BT_JSR82_RD_ENCRYPT_CFM	✓		✓			✓

Table 38: CSR_BT_JSR82_RD_ENCRYPT primitives

Description

Request that encryption is turned on, on an existing ACL link. Encryption may not be turned back off from Java while the ACL link exists. Requesting that encryption be turned off, will result in an immediate confirm signal, indicating failure.

Parameters

Type	Signal identity, CSR_BT_JSR82_RD_ENCRYPT_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
Encrypt	Boolean indicating if encryption should be turned on or off on the ACL link of the requested device (TRUE = on, FALSE = off).
Address	The Bluetooth address of the remote device.
resultCode	Values are defined in profiles.h, and may be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS:** The request was successful. (i.e. if encrypt=TRUE, encryption is now on) **CSR_BT_RESULT_CODE_JSR82_UNSPECIFIED_ERROR:** The request failed. Encryption status is unchanged. **CSR_BT_NO_CONNECTION_TO_DEVICE**: There is no connection to the specified device. **CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED**: The arguments are not allowed. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82RdEncryptReqSend (  CsrSchedQid   appHandle,
                                   CsrUint32        reqID,
                                   CsrBool       encrypt,
```

```
                                        deviceAddr_t address    )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RD_ENCRYPT_REQ primitive to the Jsr82 task.

4.39 CSR_BT_JSR82_RD_IS_ENCRYPTED

Primitives \ Parameters	type	appHandle	reqID	deviceAddr	encrypted
CSR_BT_JSR82_RD_IS_ENCRYPTED_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_RD_IS_ENCRYPTED_CFM	✓		✓		✓

Table 39: CSR_BT_JSR82_RD_IS_ENCRYPTED primitives

Description

Request the encryption status of the connection to the remote device.

Parameters

Type	Signal identity, CSR_BT_JSR82_RD_IS_ENCRYPTED_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
deviceAddr	The Bluetooth address of the remote device.
Encrypted	The encryption status. This can be either **TRUE** or **FALSE**. If there are no connections to the device, the status is **FALSE**.

The function:

```
void CsrBtJsr82RdIsEncryptedReqSend ( CsrSchedQid   appHandle,
                                      CsrUint32        reqID,
                                      deviceAddr_t  address     )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RD_IS_ENCRYPTED_REQ primitive to the Jsr82 task.

4.40 CSR_BT_JSR82_RD_IS_TRUSTED

Primitives \ Parameters	type	appHandle	reqID	deviceAddr	trusted
CSR_BT_JSR82_RD_IS_TRUSTED_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_RD_IS_TRUSTED_CFM	✓		✓		✓

Table 40: CSR_BT_JSR82_RD_IS_TRUSTED primitives

Description

Determine if a remote device is in the list of trusted devices.

Parameters

Type	Signal identity, CSR_BT_JSR82_RD_IS_TRUSTED_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
deviceAddr	The Bluetooth address of the remote device.
Trusted	The trusted status. This can be either **TRUE** or **FALSE**. If there are no connections to the device, the status is **FALSE**.

The function:

```
void CsrBtJsr82RdIsTrustedReqSend (  CsrSchedQid   appHandle,
                                     CsrUint32        reqID,
                                     deviceAddr_t  deviceAddr  )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RD_IS_TRUSTED_REQ primitive to the Jsr82 task.

4.41 CSR_BT_JSR82_RD_IS_CONNECTED

Parameters / Primitives	type	appHandle	reqID	deviceAddr	trusted
CSR_BT_JSR82_RD_IS_CONNECTED_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_RD_IS_CONNECTED_CFM	✓		✓		✓

Table 41: CSR_BT_JSR82_RD_IS_CONNECTED primitives

Description

Determine if there is an active (Java related) connection to a remote device.

Parameters

Type	Signal identity, CSR_BT_JSR82_RD_IS_CONNECTED_REQ/CFM.
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
deviceAddr	The Bluetooth address of the remote device.
Handle	A handle of 0 indicates that no connection is alive. A positive handle indicates that at least one connection is alive. The reported handle will remain unchanged as long as at least one connection is alive.

The function:

```
void CsrBtJsr82RdIsConnectedReqSend (CsrSchedQid   appHandle,
                                     CsrUint32     reqID,
                                     deviceAddr_t  deviceAddr  )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RD_IS_CONNECTED_REQ primitive to the Jsr82 task.

4.42 CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL

Primitives \ Parameters	type	appHandle	reqID	localServerChannel
CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL_REQ	✓	✓	✓	
CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL_CFM	✓		✓	✓

Table 42: CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL primitives

Description

Reserves a local server channel which is used when setting up a RFCOMM connection, using the CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN primitive. **Note:** If a server channel has been reserved, and no connection is to be made, the user must send a CSR_BT_JSR82_RFC_DISCONNECT_REQ signal to the JSR82 layer, in order to free the reserved server channel.

Parameters

Type	Signal identity, CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localServerChannel	Server channel reserved in the JSR82 layer. A value of 0 means that the process failed. This could indicate that there are no free server channels available.

The function:

```
void CsrBtJsr82RfcGetServerChannelReqSend (  CsrSchedQid   appHandle,
                                             CsrUint32       reqID  )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL_REQ primitive to the Jsr82 task.

4.43 CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN

Primitives \ Parameters	type	appHandle	reqID	localServerChannel	attrs	deviceAddress	resultCode
CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_REQ	✓	✓	✓	✓	✓		
CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_CFM	✓		✓	✓		✓	✓

Table 43: CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN primitives

Description

Sets the device in connectable mode and awaits connection from a remote device. The confirm signal is returned when a remote device has established a connection. If this operation is to be cancelled before a remote device connects, a CSR_BT_JSR82_RFC_DISCONNECT_REQ must be sent to the JSR82 layer. **Note:** If the process fails, it is the responsibility of the user to send a CSR_BT_JSR82_RFC_DISCONNECT_REQ signal to the JSR82 layer, in order to free the previously reserved server channel. If the connection attempt should be retried, this is also the user's responsibility.

Parameters

Type	Signal identity, CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localServerChannel	Server channel on the local device
Attrs	Requested attributes for the connection. Given as a CsrUint32, obtained by a bitwise OR of CSR_BT_JSR82_AUTHENTICATE, CSR_BT_JSR82_ENCRYPT and CSR_BT_JSR82_MASTER, defined in csr_bt_jsr82_prim.h Please note that if both the local and the remote device are using security level 4 (**CSR_BT_SEC_MODE4_SSP)**, Bluetooth authentication and encryption are always required, even if it is not set by the application.
deviceAddr	The Bluetooth address of the remote device which has connected.
resultCode	Values are defined in profiles.h, and may be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS:** Connection has been successfully established. **CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED**: The server channel is invalid. **CSR_BT_MAX_NUM_OF_CONNECTION**: The local device cannot accept more simultaneous RFCOMM connections. Refer to the CSR_BT_MAX_NUMBER_OF_SIMULTANIOUS_CONNECTIONS which is defined in csr_bt_usr_config.h. **CSR_BT_CONNECT_ATTEMPT_FAILED**: Failed to establish a RFCOMM

connection.

CSR_BT_CONNECTION_TERM_BY_REMOTE_HOST: The remote device has released the RFCOMM connection doing this procedure.

CSR_BT_PAGESCAN_REJECTED: The local device could not enter page scan mode.

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82RfcAcceptAndOpenReqSend( CsrSchedQid    appHandle,
                                        CsrUint32      reqID,
                                        CsrUint8    localServerChannel,
                                        CsrUint32      attrs        )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RFC_ACCEPT_AND_OPEN_REQ primitive to the Jsr82 task.

4.44 CSR_BT_JSR82_RFC_CONNECT

Primitives \ Parameters	type	appHandle	reqID	remoteServerChannel	localServerChannel	deviceAddr	attrs	resultCode	btConnId
CSR_BT_JSR82_RFC_CONNECT_REQ	✓	✓	✓	✓		✓	✓		
CSR_BT_JSR82_RFC_CONNECT_CFM	✓		✓		✓	✓		✓	✓

Table 44: CSR_BT_JSR82_RFC_CONNECT primitives

Description

Request a RFCOMM connection to a remote device

Parameters

type — Signal identity, CSR_BT_JSR82_RFC_CONNECT_REQ/CFM

appHandle — The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.

reqID — Request ID passed down from Java, and echoed back in the confirm signal.

remoteServerChannel — Server channel on the remote device to connect to. This should be found using the service discovery primitives.

localServerChannel — Server channel on the local device, assigned during the connection process. This is used for identifying this connection for subsequent data transfers, or disconnection. Returns 0 if the connection attempt fails.

deviceAddr — The Bluetooth address of the remote device.

Attrs — Requested attributes for the connection. Given as a CsrUint32, obtained by a bitwise OR of CSR_BT_JSR82_AUTHENTICATE, CSR_BT_JSR82_ENCRYPT and CSR_BT_JSR82_MASTER, defined in csr_bt_jsr82_prim.h.

Please note that if both the local and the remote device are using security level 4 (**CSR_BT_SEC_MODE4_SSP)**, Bluetooth authentication and encryption are always required, even if it is not set by the application.

resultCode — Values are defined in profiles.h and can be:

CSR_BT_RESULT_CODE_JSR82_SUCCESS: A RFCOMM connection is established with success.

CSR_BT_MAX_NUM_OF_CONNECTION: The local device cannot create more simultaneous RFCOMM connections. Refer to The MAX_NUMBER_OF_SIMULTANIOUS_CONNECTIONS which is defined in csr_bt_usr_config.h

CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED: A server channel

could not be registered locally. This may indicate that all channels are in use.

CSR_BT_CONNECT_ATTEMPT_FAILED: Failed to establish a RFCOMM connection.

CSR_BT_CONNECTION_TERM_BY_REMOTE_HOST: The remote device has released the RFCOMM connection doing this procedure.

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

btConnId — Global CSR Synergy Bluetooth connection identifier.

Can be used for controlling misc. connection related parameters and e.g. AMP.

CSR_BT_CONN_ID_INVALID (0) is used if no global connection identifier could be assigned.

The function:

```
void CsrBtJsr82RfcConnectReqSend (   CsrSchedQid  appHandle,
                                     CsrUint32       reqID,
                                     CsrUint8 remoteServerChannel,
                                     deviceAddr_t address,
                                     CsrUint32       attrs    )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RFC_CONNECT_REQ primitive to the Jsr82 task.

4.45 CSR_BT_JSR82_RFC_DISCONNECT

Primitives \ Parameters	type	appHandle	reqID	localServerChannel	resultCode
CSR_BT_JSR82_RFC_DISCONNECT_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_RFC_DISCONNECT_CFM	✓		✓		✓

Table 45: CSR_BT_JSR82_RFC_DISCONNECT primitives

Description

End a RFCOMM connection to a remote device, and unregister the server channel that was registered with CSR_BT_JSR82_RFC_GET_SERVER_CHANNEL. Corresponds to calling close on the StreamConnectionNotifier object in a server connection.

Parameters

Type	Signal identity, CSR_BT_JSR82_RFC_DISCONNECT_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localServerChannel	The local server channel of the connection to break.
resultCode	Values are defined in profiles.h and can be: **CSR_BT_CONNECTION_TERM_BY_LOCAL_HOST:** The local device initiated the disconnect procedure. **CSR_BT_CONNECTION_TERM_BY_REMOTE_HOST:** The remote device initiated the disconnect procedure. **CSR_BT_NO_CONNECTION_TO_DEVICE:** The connection being cancelled was not found. **CSR_BT_CONNECT_ATTEMPT_FAILED:** Disconnected while waiting for a connection. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82RfcDisconnectReqSend (CsrSchedQid appHandle, CsrUint32
                                     reqID, CsrUint8 localServerChannel,
                                     resultCode_t result)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RFC_DISCONNECT_REQ primitive to the Jsr82 task.

4.46 CSR_BT_JSR82_RFC_CLOSE

Primitives \ Parameters	type	appHandle	reqID	localServerChannel	resultCode
CSR_BT_JSR82_RFC_CLOSE_REQ	✓	✓	✓	✓	
CSR_BT_JSR82_RFC_CLOSE_CFM	✓		✓		✓

Table 46: CSR_BT_JSR82_RFC_CLOSE primitives

Description

End a RFCOMM connection with a remote device. On the client side of a connection, this unregisters the associated server channel as well. This signal corresponds to calling close on the StreamConnection object for the connection.

Parameters

Type	Signal identity, CSR_BT_JSR82_RFC_CLOSE_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localServerChannel	The local server channel of the connection to close.
resultCode	Values are defined in profiles.h and can be: **CSR_BT_CONNECTION_TERM_BY_LOCAL_HOST:** The local device initiated the disconnect procedure. **CSR_BT_CONNECTION_TERM_BY_REMOTE_HOST:** The remote device initiated the disconnect procedure. **CSR_BT_NO_CONNECTION_TO_DEVICE:** The connection being cancelled was not found. **CSR_BT_CONNECT_ATTEMPT_FAILED:** Disconnected while waiting for a connection. **CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED**: Close is called on a connection that is not up and running. This can only come from an error in the middle layers, as the StreamConnection object does not exist at this point in a connection. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82RfcCloseReqSend (CsrSchedQid appHandle, CsrUint32 reqID,
CsrUint8 localServerChannel, resultCode_t result)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RFC_CLOSE_REQ primitive to the Jsr82 task.

4.47 CSR_BT_JSR82_RFC_GET_AVAILABLE

Primitives \ Parameters	type	appHandle	reqID	localServerChannel	bytesAvailable	resultCode
CSR_BT_JSR82_RFC_GET_AVAILABLE_REQ	✓	✓	✓	✓		
CSR_BT_JSR82_RFC_GET_AVAILABLE_CFM	✓		✓	✓	✓	✓

Table 47: CSR_BT_JSR82_RFC_GET_AVAILABLE primitives

Description

Request the number of bytes presently available in the receive buffer of a RFCOMM connection.

Parameters

Type	Signal identity, CSR_BT_JSR82_RFC_GET_AVAILABLE_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localServerChannel	The local server channel of the connection.
bytesAvailable	The number of bytes available in the receive buffer of the requested connection. Returns 0 if the buffer is empty.
resultCode	The result of the operation. Values are defined in profiles.h, and may be: **CSR_BT_RESULT_CODE_JSR82_SUCCESS**: The number of bytes available was read successfully. **CSR_BT_NO_CONNECTION_TO_DEVICE**: The specified connection does not exist. The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

The function:

```
void CsrBtJsr82RfcGetAvailableReqSend (   CsrSchedQid  appHandle,
                                          CsrUint32       reqID ,
                                          CsrUint8 localServerChannel)
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RFC_GET_AVAILABLE_REQ primitive to the Jsr82 task.

4.48 CSR_BT_JSR82_RFC_SEND_DATA

Parameters / Primitives	type	appHandle	reqID	localServerChannel	payloadLength	*payload	bytesWritten
CSR_BT_JSR82_RFC_SEND_DATA_REQ	✓	✓	✓	✓	✓	✓	
CSR_BT_JSR82_RFC_SEND_DATA_CFM	✓		✓	✓			✓

Table 48: CSR_BT_JSR82_RFC_SEND_DATA primitives

Description

Send data over a RFCOMM connection. If the amount of data to be transmitted exceeds the maximum frame size of the connection, the JSR82 layer will segment the payload data.

Parameters

type	Signal identity, CSR_BT_JSR82_RFC_SEND_DATA_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localServerChannel	The local server channel of the connection to transmit data over.
payloadLength	The number of bytes to be transmitted.
*payload	A pointer to the data to transmit. Note: The application may not free the payload pointer – this is handled by the JSR82 layer.
bytesWritten	The number of bytes successfully sent over the connection. This will always match the payloadLength of the request, unless the connection is closed.

The function:

```
void CsrBtJsr82RfcSendDataReqSend (CsrSchedQid   appHandle,
                                   CsrUint32     reqID,
                                   CsrUint8 localServerChannel,
                                   CsrUint32     payloadLength,
                                   CsrUint8  *payload    )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RFC_SEND_DATA_REQ primitive to the Jsr82 task.

4.49 CSR_BT_JSR82_RFC_RECEIVE_DATA

Parameters / Primitives	type	appHandle	reqID	localServerChannel	bytesToRead	payloadLength	*payload
CSR_BT_JSR82_RFC_RECEIVE_DATA_REQ	✓	✓	✓	✓	✓		
CSR_BT_JSR82_RFC_RECEIVE_DATA_CFM	✓		✓	✓		✓	✓

Table 49: CSR_BT_JSR82_RFC_RECEIVE_DATA primitives

Description

Request to read data in the JSR-82 data receive buffer if any has been received. The confirm signal will not be received until data has been received in the buffer, or the connection is broken.

Parameters

Type	Signal identity, CSR_BT_JSR82_RFC_RECEIVE_DATA_REQ/CFM
appHandle	The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to appHandle.
reqID	Request ID passed down from Java, and echoed back in the confirm signal.
localServerChannel	Local server channel of the connection to receive data on.
bytesToRead	The (maximum) number of bytes to get from the specified connection.
payloadLength	The number of bytes read from the specified connection. May be less than the number requested in bytesToRead. Will return 0 if the connection was broken.
*payload	Data read from the specified connection.

The function:

```
void CsrBtJsr82RfcReceiveDataReqSend ( CsrSchedQid  appHandle,
                                       CsrUint32       reqID,
                                       CsrUint8 localServerChannel,
                                       CsrUint32       bytesToRead   )
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_RFC_RECEIVE_DATA _REQ primitive to the Jsr82 task.

4.50 CSR_BT_JSR82_CLEANUP_REQ

Primitives \ Parameters	type
CSR_BT_JSR82_CLEANUP_REQ	✓

Table 50: CSR_BT_JSR82_CLEANUP_REQ primitives

Description

Sending this signal indicates that the Java Virtual Machine (or possibly just the Bluetooth related part of it) has been shut down. It causes the JSR82 layer in CSR Synergy Bluetooth disconnect all L2CAP and RFCOMM connections, unregister any services offered by the Java application, and free all unnecessary data it holds. It then goes into an idle state, and is ready to react if the virtual machine is restarted.

If the Java Virtual Machine is shut down without sending this signal, connections and services created by the Java application will appear to be accessible to remote devices.

Parameters

type Signal identity, CSR_BT_JSR82_CLEANUP_REQ

The function:

```
void CsrBtJsr82CleanupReqSend ()
```

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_CLEANUP_REQ primitive to the Jsr82 task.

4.51 CSR_BT_JSR82_SET_EVENT_MASK

Primitives \ Parameters	type	phandle	eventMask	conditionMask
CSR_BT_JSR82_SET_EVENT_MASK_REQ	✓	✓	✓	✓
CSR_BT_JSR82_SET_EVENT_MASK_CFM	✓		✓	

Table 51: CSR_BT_JSR82_SET_ EVENT_MASK primitives

Description

This signal can be used for setting which extended information the application will subscribe for, and may be sent momentarily from the application.

The function:

CsrBtJsr82SetEventMaskReqSend (qid thePhandle, Jsr82_eventmask_t eventMask, Jsr82_eventmask_cond_t conditionMask);

defined in csr_bt_jsr82_lib.h, builds and sends the CSR_BT_JSR82_SET _EVENT_MASK_REQ primitive to the Jsr82 task.

Parameters

type — Signal identity, CSR_BT_JSR82_SET _EVENT_MASK_REQ /CFM.

thePhandle — The identity of the calling process. It is possible to initiate the procedure by any higher layer process as the response is returned to thePhandle.

eventMask — In the request message the eventMask parameter describes which extended information an application will subscribe for, and the confirm message describes which event has been set.

Please note that in cases where the application set the eventMask parameter to a value that are not supported then only the supported values will be set in the confirmed message. The event mask values are defined in csr_bt_jsr82_prim.h.

Value	Parameter description	Reference
CSR_BT_JSR82_EVENT_MASK_SUBSCRIBE_NONE	Stop subscribing for any extended information.	N/A
CSR_BT_JSR82_EVENT_MASK_SUBSCRIBE_RFC_CLOSE_IND	Subscribing for RFCOMMM close events	4.52
CSR_BT_JSR82_EVENT_MASK_SUBSCRIBE_L2CA_CLOSE_IND	Subscribing for L2CAP close events	4.53

conditionMask — In the request message the conditionMask parameter defines at which condition the extended information must be given to the application. The condition values are defined in in

csr_bt_jsr82_prim.h.

Value	Parameter description
CSR_BT_JSR82_EVENT_MASK_COND_ALL	No condition, e.g. the extended information that the application has subscribed for is always sent up. Example: If the application has subscribed for the RFCOMM Close event it will receive a CSR_BT_JSR82_RFC_CLOSE_IND message even if the attempt to release a RFCOMM connection fails.
CSR_BT_JSR82_EVENT_MASK_COND_SUCCESS	The extended information that the application has subscribed for is only sent up if status is success. Example: If the application has subscribed for the RFCOMM Close event it will only receive a CSR_BT_JSR82_RFC_DISCONNECT_IND message if an RFCOMM connection is released.

4.52 CSR_BT_JSR82_RFC_CLOSE_IND

Parameters / Primitives	type	localServerChannel	resultCode
CSR_BT_JSR82_RFC_CLOSE_IND	✓	✓	✓

Table 52: CSR_BT_JSR82_RFC_CLOSE event primitives

Description

If the application has subscribed for the RFCOMM Close Event by setting the eventMask parameter in the CSR_BT_JSR82_SET_EVENT_MASK_REQ message to CSR_BT_JSR82_EVENT_MASK_SUBSCRIBE_RFC_CLOSE_IND, it will receive the CSR_BT_JSR82_RFC_CLOSE_IND message whenever an RFCOMM connection has been released.

Parameters

type — Signal identity, CSR_BT_JSR82_RFC_CLOSE_IND.

localServerChannel — The local server channel of the RFCOMM connection which has been released

resultCode — Reports the result of releasing a RFCOMM connection. The result values are defined in csr_bt_profiles.h and can be:

CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED: Trying to disconnect a RFCOMM connection that is not up and running.

CSR_BT_ABNORMAL_LINK_DISCONNECT: Disconnected due to link loss.

CSR_BT_CONNECTION_TERM_BY_REMOTE_HOST: The RFCOMM connection is released from remote device.

CSR_BT_CONNECTION_TERM_BY_LOCAL_HOST: The RFCOMM connection is released from local device.

CSR_BT_CONNECTION_TERM_BY_LOCAL_HOST_FAIL: The attempt to disconnect a RFCOMM connection failed.

Please note that if the condition parameter in the CSR_BT_JSR82_SET_EVENT_MASK_REQ message is set to CSR_BT_JSR82_EVENT_MASK_COND_SUCCESS then the application will only receive this event if a RFCOMM connection has been released. E.g. it will not receive this event if the result is CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED or CSR_BT_CONNECTION_TERM_BY_LOCAL_HOST_FAIL

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

4.53 CSR_BT_JSR82_L2CA_CLOSE_IND

Primitives \ Parameters	type	btConnId	localTerminated	resultCode
CSR_BT_JSR82_L2CA_CLOSE_IND	✓	✓	✓	✓

Table 53: CSR_BT_JSR82_L2CA_CLOSE event primitives

Description

If the application has subscribed for the L2CAP Close Event by setting the eventMask parameter in the CSR_BT_JSR82_SET_EVENT_MASK_REQ message to CSR_BT_JSR82_EVENT_MASK_SUBSCRIBE_L2CA_CLOSE_IND, it will receive the CSR_BT_JSR82_L2CA_CLOSE_IND message whenever an L2CAP connection has been released.

Parameters

type — Signal identity, CSR_BT_JSR82_L2CA_CLOSE_IND.

btConnId — The BT connection ID of the connection that has been released.

localTerminated — Boolean to indicate whether the L2CAP release was provoked by the local device.

resultCode — Reports the result of releasing a L2CAP connection. The result values are defined in csr_bt_profiles.h and can be:

CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED: Trying to disconnect a L2CAP connection that is not up and running.

CSR_BT_ABNORMAL_LINK_DISCONNECT: Disconnected due to link loss.

CSR_BT_CONNECTION_TERM_BY_REMOTE_HOST: The L2CAP connection is released from remote device.

CSR_BT_CONNECTION_TERM_BY_LOCAL_HOST: The L2CAP connection is released from local device.

Please note that if the condition parameter in the CSR_BT_JSR82_SET_EVENT_MASK_REQ message is set to CSR_BT_JSR82_EVENT_MASK_COND_SUCCESS then the application will only receive this event if a L2CAP connection has been released. E.g. it will not receive this event if the result is CSR_BT_RESULT_CODE_JSR82_COMMAND_DISALLOWED

The valid values for this parameter are defined in csr_bt_jsr82_prim.h. CSR_BT_RESULT_CODE_JSR82_SUCCESS signals a successful operation.

5 Document References

Document	Reference
Specification of the Bluetooth System Version 1.1, 1.2 and 2.0	[BT]
Java™ APIs for Bluetooth™ Wireless Technology (JSR-82) Version 1.1.1	[JSR82SPEC]
CSR Synergy Bluetooth, SD - Service Discovery API Description, document no. api-0103-sd	[SD]
CSR Synergy Bluetooth, SC – Security Controller API Description, Document no. api-0102-sc	[SC]

Terms and Definitions

BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CSR	Cambridge Silicon Radio
SDDB	Service Discovery DataBase
J2ME	Java 2 Platform, Micro Edition
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