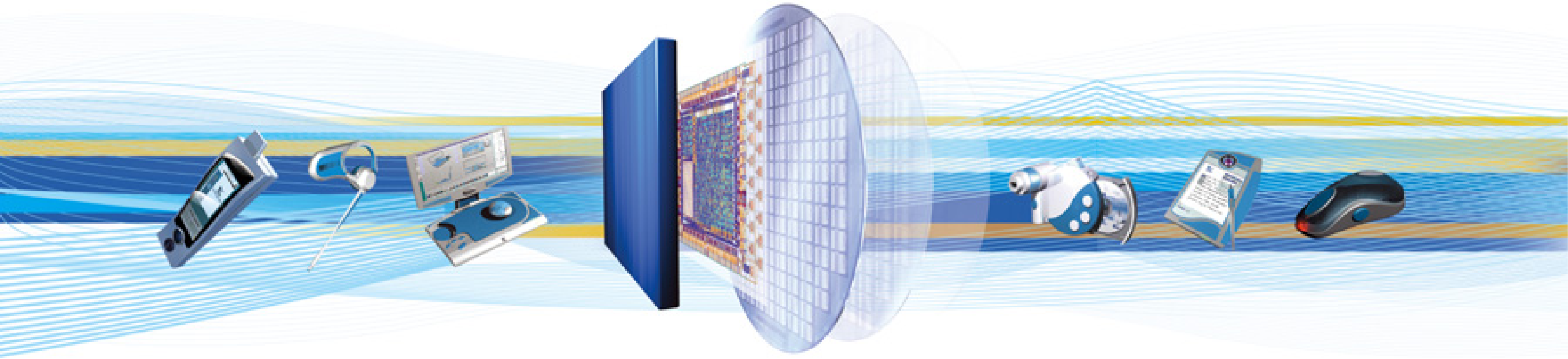
CSR Synergy Bluetooth 18.2.0

SD – Service Discovery

API Description

November 2011



Cambridge Silicon Radio Limited

Churchill House
Cambridge Business Park
Cowley Road
Cambridge CB4 0WZ
United Kingdom

Registered in England and Wales 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction and Scope

This document describes the functionality and message interface provided by the Service Discovery module (SD) in CSR Synergy Bluetooth.

1.2 Assumptions

The following assumptions and preconditions are made in the following:

- The SD shall only handle one search request per application at the time
- The SD shall only handle one read available service request per application at the time
- The SD shall at maximum handle 16 simultaneous applications
- Any data allocated by the application and referenced from downstream messages is owned by the profile and must not be freed by the application. Likewise, any data allocated by the profile and referenced in upstream messages is owned by and must by freed by the application. All pointers in downstream messages refer to data allocated by the application and all pointers in upstream messages refer to data allocated by the profile.

2 Description

2.1 Introduction

The SD module provides a set of functions, especially designed to speed up discovery of remote Bluetooth® devices including their names and available services. The SD module is designed to handle multiple instances, meaning that up to 16 applications can use the SD at the same time. Further, as an extended feature, the SD holds a list of all remote devices which have been paired with the local device. Therefore, if an application requests to determine which devices that have been paired, the application can read the entire list of these devices through the SD together with all known information related to those devices.

The services offered by the SD module to an application are:

- Search for remote devices, both previously paired and not paired
- Service search, for a specified device
- Listing of all remote devices, both paired and not paired

2.2 Reference Model

The application must communicate through the SD, which will take care of all necessary communication down towards the CM and SC afterwards. The SD will also take care of the communication with GATT if LE is enabled.

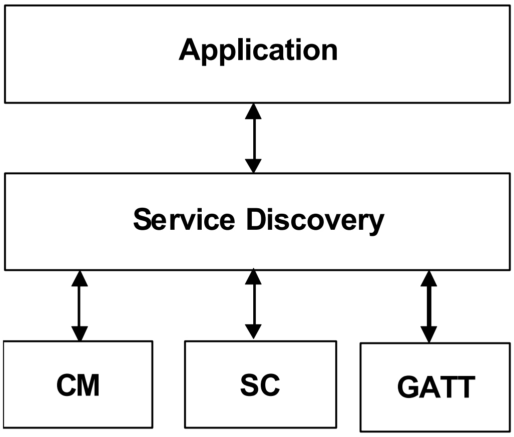
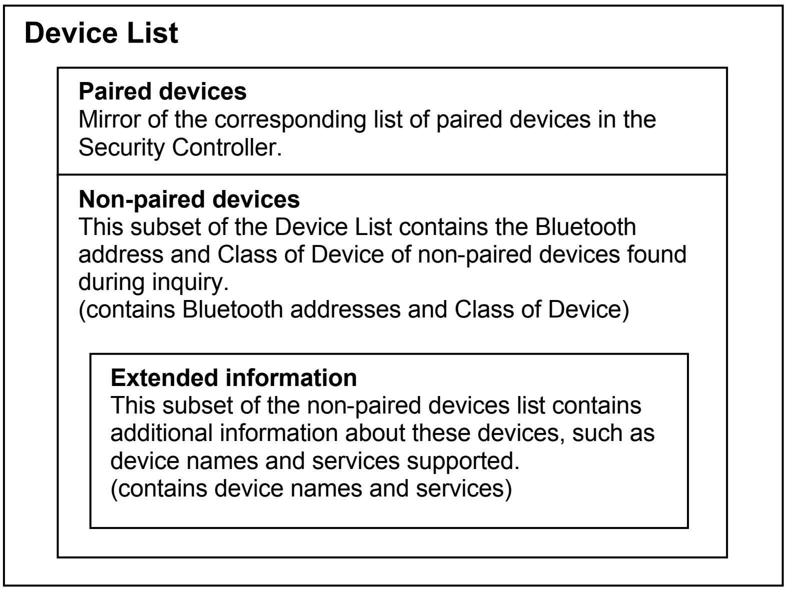


Figure 1: Reference model

3 Interface Description

3.1 The Device List

The SD contains an internal Device List, which it uses to store information about remote devices.



Device List

Paired devices
Mirror of the corresponding list of paired devices in the Security Controller.

Non-paired devices
This subset of the Device List contains the Bluetooth address and Class of Device of non-paired devices found during inquiry.
(contains Bluetooth addresses and Class of Device)

Extended information
This subset of the non-paired devices list contains additional information about these devices, such as device names and services supported.
(contains device names and services)

Figure 2: Device List overview

The layout of the Device List is depicted in Figure 2, which illustrates that the list contains both paired and non-paired devices. The subset of the list with paired devices is synchronized with a corresponding list in the Security Controller and can not be configured with respect to size. The rest of the list contains information about non-paired devices and is used both as a filter for preventing the same device being reported more than once and for making sure that remote devices are reported to the application according to their distance from the inquiring device (also referred to as RSSI-sorting).

Besides Bluetooth address and Class of Device the Device List can contain additional information such as device names and services supported. As this type of information can consume a relatively large amount of memory, it is stored as a subset of the non-paired device list. The size of this subset may independently configured using the *deviceListInfoMax* parameter in the CSR_BT_SD_MEMORY_CONFIG_REQ primitive as indicated in Figure 3. If no CSR_BT_SD_MEMORY_CONFIG_REQ is issued from an application, the number of devices to store additional information for is defined by CSR_BT_SD_MEMORY_LIST_INFO_MAX_DEFAULT in usr_config.h. Consequently these parameters relates to the size of the non-paired devices part of the Device List only. The size of the paired devices part is unaffected by these parameters.

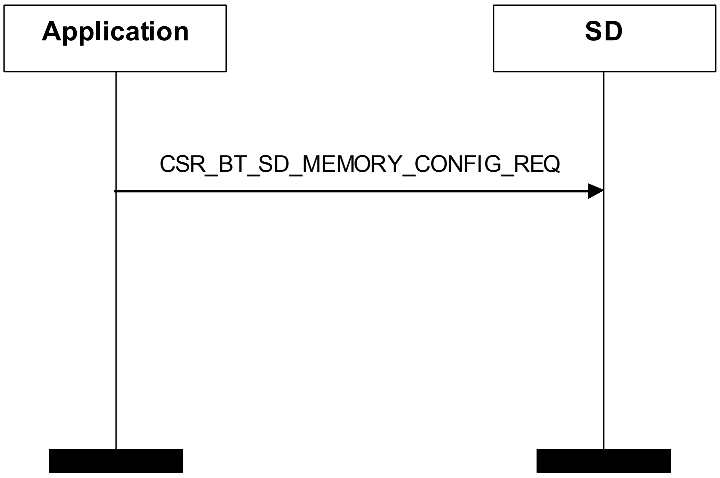


Figure 3: SD memory configuration sequence

The benefit of storing additional information such as names in the Device List is that it makes device searches faster as less time is spent on reading the names of remote devices, if names for the corresponding devices are already present in the Device List. Additionally, applications can benefit from a Device List with extended information since the applications themselves does not have to store this information but can rely on the SD to provide it.

3.2 Searching for Remote Devices

The SD can be instructed to search for remote devices by using the CSR_BT_SD_SEARCH_REQ primitive. When a search request has been sent from the application, the SD will return a result indication each time it finds a device. When the requested number of devices is found, or the search timer runs out, a close search indication (CSR_BT_SD_CLOSE_SEARCH_IND) is sent to the application.

The SD will automatically search for both classic Bluetooth BR/EDR devices, and for Bluetooth 4.0 “Low Energy” devices. The API allows for dynamic configuration of what search method shall be used.

Please note that an application can only have one search active at the time.

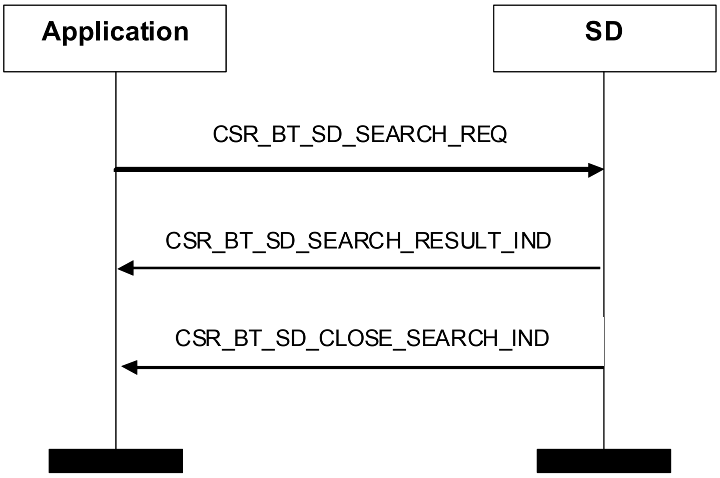


Figure 4: Search sequence

In each result indication (CSR_BT_SD_SEARCH_RESULT_IND), additional information related to the found device is included in the *info* field explained in the following.

Please be aware that the services contained in the search result are not necessarily up to date, as they will be read from the Device List. The exception is if the remote device is sending extended inquiry results in which case the service information will originate from this. If an updated list of the available services offered by a Bluetooth device is requested, the CSR_BT_SD_READ_AVAILABLE_SERVICES_REQ command should be used.

3.2.1 Device Info Format

The *info* field is a byte stream in a format that closely resembles the one used in extended inquiry responses from Bluetooth 2.1. The main difference is that the Length and Type fields are defined with two bytes for each field in little endian order instead of just one. An overview of this is shown in figure Figure 5, where one field inside the info sequence is outlined.

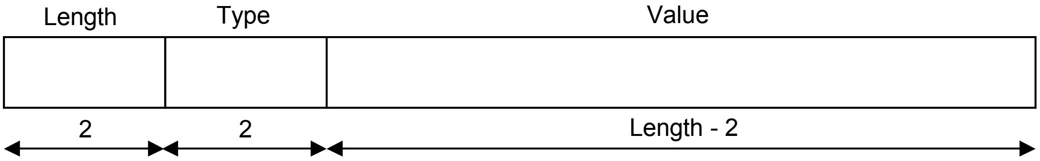


Figure 5: An overview of the data format in the info sequence. Values are measured in bytes

Data types are allocated for compatibility with the EIR format, which means that types of 0x00FF and below are reserved for identifiers allocated for EIR by the Bluetooth SIG. Types in the interval 0x0100 to 0xFFFF are reserved for future CSR Synergy Bluetooth extensions. In order to provide extensibility, any parser of this format must silently ignore types that it does not know, and skip to the next field.

3.2.2 Decoding the Device Info Format

Csr_bt_sd_lib.h includes a set of helper functions that can be used for extracting data from the device info field. Currently the fields can contain information about the friendly name of the remote device, and which services are available. The following functions are currently available for use:

Get the friendly name

```
CsrUint8 CsrBtSdDecodeFriendlyName( CsrUint8 *info,

                                    CsrUint32 infoLen,

                                    CsrUint8 **friendlyName);
```

This function will place the friendly name as a zero-terminated string in the friendlyName parameter. This parameter will be dynamically allocated inside the decoder function, and must then be freed by the caller. The only exception to this is when no name is present in the info field. In this case, the friendlyName parameter will be set to NULL. The function returns the length of the remote name excluding the terminating zero. Example:

```
CsrUint8 *name, nameLen;

nameLen = SdDecodeFriendlyName(info, infoLen, &name);

if ((0 < nameLen) && (NULL != name))

{

    printf(“Name: %s\n”, name);

}

CsrPfree(name);
```

Check if a specific service is available

```
CsrBool CsrBtSdIsServicePresent( CsrUint8 *info,
                                             CsrUint32 infoLen,

                                             CsrUint32 uuid);
```

This function determines if a specified UUID in 16- or 32 bit form is present in the device info field. Any 128 bit UUIDs that can be reduced to 16- or 32 bit form will also be considered. This information is then returned as a boolean. Example:

```
        if (CsrBtSdIsServicePresent(info, infoLen, SPP_PROFILE_UUID)

        {

            printf(“The remote device supports the Serial Port Profile\n”);

        }
```

The different profile UUIDs are defined in profiles.h.

Get a list of all supported services

```
        CsrUint16 CsrBtSdReadServiceList(CsrUint8 *info,

                                             CsrUint32 infoLen,

                                             CsrUint32 **serviceList);
```

This function returns a list of all 16- and 32 bit services present in the device info field in 32 bit form. Any 128 bit UUIDs that can be reduced to 32 bit form will also be included. The list of services will be included in the *serviceList* parameter that is dynamically allocated inside the decoder function, and must then be freed by the caller. The function returns the number of services in the service list. Example:

```
        CsrUint32 *serviceList;
        CsrUint16 serviceCount, index;

        serviceCount = CsrBtSdReadServiceList(info, infoLen, &serviceList);

        printf(“The following UUIDs are supported by the remote device: “);

        for (index = 0; index < serviceCount; index++)

        {

            printf(“0x%08X, ”, serviceList[index]);

        }

        CsrPfree (serviceList);
```

Check and retrieve miscellaneous tags

```
        CsrBool CsrBtSdInfoCheckTag(  CsrUint16 infoLen,
                       CsrUint8 *info,

                       CsrUint16 tag);
```

This function can be used for checking whether a specific tag is present in the info. It will return TRUE if the tag specified by ‘tag’ is available.

```
        CsrUint16 CsrBtSdInfoGetTag(  CsrUint16 infoLen,

                       CsrUint8 *info,
                       CsrUint8 **tagVal,

                       CsrUint16 tag);
```

This function can update a pointer to point to the value of the specified tag as well as return the length of the tag. It will not copy the tag value like CsrBtSdDecodeFriendlyName does but instead update the pointer *tagVal to point to the beginning of the tag value inside of the info. The following tags are currently available:

```
CSR_BT_SD_DEVICE_INFO_TYPE_UUID16_INCOMPLETE

CSR_BT_SD_DEVICE_INFO_TYPE_UUID16_COMPLETE

CSR_BT_SD_DEVICE_INFO_TYPE_UUID32_INCOMPLETE

CSR_BT_SD_DEVICE_INFO_TYPE_UUID32_COMPLETE

CSR_BT_SD_DEVICE_INFO_TYPE_UUID128_INCOMPLETE

CSR_BT_SD_DEVICE_INFO_TYPE_UUID128_COMPLETE

CSR_BT_SD_DEVICE_INFO_TYPE_NAME_SHORT

CSR_BT_SD_DEVICE_INFO_TYPE_NAME_FULL

CSR_BT_SD_DEVICE_INFO_TYPE_MANUFACTURER
```

3.2.3 Device Status

The search results include a device status bitmask with a number of flags that identify various properties for a device. These properties are defined in csr_bt_sd_prim.h and can be:

Flag	Description
CSR_BT_SD_STATUS_TRUSTED	The device is trusted by the SC. No authorize indications will be sent from the security controller on connections from this device.
CSR_BT_SD_STATUS_PAIRED	The device has been paired, i.e. PIN keys have been exchanged for an authentication bond.
CSR_BT_SD_STATUS_REMOTE_NAME_UPDATED	The remote name has been updated (not necessarily changed) in the search compared to the name in the Device List. This indicates that the name was either included in an Extended Inquiry Response or was retrieved using a dedicated read remote name procedure.
CSR_BT_SD_STATUS_REMOTE_NAME_FROM_DEVICE_LIST	The remote name was found in Device List and has not been read from the remote device in this search.
CSR_BT_SD_STATUS_RADIO_LE	Peer is capable of using the Bluetooth 4.0 Low Energy radio type
CSR_BT_SD_STATUS_RADIO_BREDR	Peer has support for the classic BR/EDR Blueooth radio type
CSR_BT_SD_STATUS_PRIVATE_ADDR	The Low Energy capable peer is using a random (private) Bluetooth address

CSR Synergy Bluetooth 18.2.0 SD – Service Discovery API

Flag	Description
All other bits	These bits are reserved for future use, and must be ignored.

3.2.4 Search Modes

When searching for remote devices the SD can operate in two basic modes: Buffered search and standard search, which are illustrated in Figure 6.

Buffered Search (RSSI-sorting)

Internal SD state	Actions
Buffering devices	Inquiry for devices
Processing buffer	Read remote names – closest devices (highest RSSI) first
	Close search when names have been read for all devices found during buffering or restart search if configured to do so

Standard Search

Internal SD state	Actions
Device search	Inquiry until first unknown device is found
	Read name of remote device
	Inquiry until first unknown device is found
	Read name of remote device
	Etc…until cancelled from application or timeout

Figure 6: The two search modes of the SD

It should be noted that the two basic search modes can be configured in different ways as specified in Section 3.3.

Standard Search

The standard search procedure continually searches for remote devices and cancels search each time it finds an unknown device in order to read the name of the device. This implies that search results are sent to the application in random order and at random points in time depending on the detection of the remote devices.

Buffered Search (RSSI-sorting)

The SD module has the possibility of presenting devices to the application, found during a buffered search procedure, according to their RSSI (Received Signal Strength Indication) and thereby their distance from the device performing the search. The benefit is that devices located close to the user will be presented to the user before devices located further away from the user, thereby enabling the user to find the required device more quickly. This is based on the assumption that the user most often is interested in finding a device in the near proximity.

It should be noted that the RSSI level is not a direct or consistent indicator of the distance to the device, but in most circumstances it can give a usable indication nevertheless.

Presenting devices sorted by RSSI, requires the SD to search for a predefined duration (the *buffering devices* state) before requesting names of detected devices as opposite to requesting remote name momentarily after the detection of a single new device.

The duration of which to search for devices before reading names can be configured by *rssiBufferTime* in the CSR_BT_SD_SEARCH_REQ signal. It is recommended to set the value between 4000 and 6000 milliseconds for a normal usage scenario, as tests have shown that if a limited number of Bluetooth devices are in range, then all these devices will normally be found within this time. A value of *CSR_BT_SD_SEARCH_DISABLE_BUFFERING* will disable the RSSI sorting and the SD will operate in standard search mode.

When selecting a value for the RSSI buffer timer, it is important to note that this timer will define the minimum time it takes to get the first CSR_BT_SD_SEARCH_RESULT_IND in the default configuration.

If immediate search results are enabled these will be sent to the application in any state where the SD is performing inquiry.

3.3 Proximity Pairing Searching

When using the CSR patented feature Proximity Pairing, the possibility to set the Inquiry Transmit Power Level is often exploited.

The Inquiry Transmit Power Level is specified using the `CsrBtSdProximitySearchReqSend`-function. This function works exactly the same way as the `CsrBtSdSearchReqSend` –function, except that it takes an input parameter called `inquiryTxPowerLevel`, which specifies the Inquiry Transmit Power Level directly in dBm.

Refer to Sections 3.4.3 and 4.5 for details on the use of the `CsrBtSdProximitySearchReqSend`-function.

3.4 Search Configuration

The search procedure of the SD can be configured using a combination of parameters in a CSR_BT_SD_SEARCH_REQ, defines in csr_bt_usr_config.h and by using a CSR_BT_SD_SEARCH_CONFIG_REQ primitive.

3.4.1 Using CSR_BT_SD_SEARCH_CONFIG_REQ for Runtime Configuration

It is possible to configure the search settings in the SD at runtime by using the CSR_BT_SD_SEARCH_CONFIG_REQ signal. All parameters that this signal changes have a corresponding setting in csr_bt_usr_config.h, so this signal should only be used if the parameters should be changed during runtime. The following settings can be changed with the CSR_BT_SD_SEARCH_CONFIG_REQ signal:

Read Remote Name Timeout

During a search procedure the SD will read the names of unknown devices. The time-out for reading a remote name can be customised both in csr_bt_usr_config.h using CSR_BT_SD_READ_NAME_TIMEOUT_DEFAULT, and at runtime by means of the CSR_BT_SD_SEARCH_CONFIG_REQ primitive. A value of 5000 milliseconds is recommended.

Max Search Results

It is sometimes required to be able to limit the number of search results produced by a search procedure. This can be done in usr_config.h by setting CSR_BT_SD_SEARCH_MAX_RESULTS_DEFAULT. This value can be changed at runtime through the CSR_BT_SD_SEARCH_CONFIG_REQ. Setting this value to CSR_BT_UNLIMITED will result in all devices found during inquiry to be sent to the application layer.

Search Configuration

It is possible to modify the behaviour of the search procedure by setting a set of flags in a bitmask. Currently the following flags can be set (defined in csr_bt_sd_prim.h):

Flag	**Purpose**
CSR_BT_SD_SEARCH_USE_STANDARD	This corresponds to the negation of the rest of the flags, e.g. the standard behaviour.

Flag	Purpose
CSR_BT_SD_SEARCH_USE_PRECONFIGURED	Use the settings that have been previously configured. If this is set, the SD will ignore any settings in this field and instead use the settings from a previous CSR_BT_SD_SEARCH_CONFIG_REQ if this signal has been previously sent or from CSR_BT_SD_SEARCH_CONFIG_DEFAULT in csr_bt_usr_config.h.
CSR_BT_SD_SEARCH_FORCE_NAME_UPDATE	Force names to be updated during search. This will retrieve the remote name of a device even if this device is already present in the Device List.
CSR_BT_SD_SEARCH_SHOW_UNKNOWN_DEVICE_NAMES	If the name of the remote device could not be retrieved, the corresponding result indication is normally not sent to the application. If this flag is set, then result indications for these devices will be sent.
CSR_BT_SD_SEARCH_HIDE_PAIRED_DEVICES	If this is set, then any device that is already paired will not be shown in a CSR_BT_SD_SEARCH_RESULT_IND.
CSR_BT_SD_SEARCH_DO_NOT_CLEAR_FILTER	The SD keeps a search filter that will hide any devices that have previously been found. If this is set, then this filter will not be cleared between search requests. This can be advantageous in a scenario where the wanted device was not found in a search, and the user restarts the search.
CSR_BT_SD_SEARCH_CONTINUE_AFTER_RSSI_SORTING	Normally, the SD will stop an RSSI sorted search when all devices that were found during the initial search have been processed. If this flag is set, then the search will continue like if it was in standard search mode. This will only have any effect if a buffered search is performed.
CSR_BT_SD_SEARCH_ALLOW_UNSORTED_SEARCH_RESULTS	When using a buffered search, search results will by default be sent to the application sorted according to their RSSI level. By setting this flag search results will be sent to the application in unsorted order. This will only have any effect if a buffered search is performed. The benefit of specifying this flag is that search results potentially will be sent to the application faster.
CSR_BT_SD_SEARCH_ENABLE_IMM_RESULTS	Setting this flag will enable immediate search results to be sent to the application. Refer to section 4.7 for more details.
CSR_BT_SD_SEARCH_DISABLE_NAME_READING	This will disable dedicated name reading completely. Search results from devices supporting Extended Inquiry might still contain names, but the search will not be halted in order to retrieve remote names.
CSR_BT_SD_SEARCH_LOW_INQUIRY_PRIORITY_DURING_ACL	Setting this flag will lower the priority of the inquiry procedure if an ACL exist.
CSR_BT_SD_SEARCH_ENABLE_SCAN_DURING_INQUIRY	Setting this flag will allow discoverability during process of searcing for devices. Be aware that this might slow down the serach process and be more power comsuming.
CSR_BT_SD_SEARCH_DISABLE_BT_LE	Do not use the Bluetooth 4.0 Low Energy radio for scanning
CSR_BT_SD_SEARCH_DISABLE_BT_CLASSIC	Do not use the classic BR/EDR Bluetooth radio for inquiry

Flag	Purpose
CSR_BT_SD_SEARCH_ENABLE_LE_PASSIVE_SCAN	Setting this flag will change the scan for LE devices to be passive I.e. the SD will only receive the information which is contained in the AD packet advertised by the peer devices, and not asking for the additional scan response packet which is also obtained in the default active scan mode. The passive mode can be a little faster than active scan mode, when discovering devices and services, but since the AD packet can only contain 31 octes of data, the probability of getting an incomplete list of services or not obtaining the complete name is extended. If CSR_BT_SD_SEARCH_DISABLE_NAME_READING is not set, and the device name is not present in the AD packet, the SD will connect to the device to obtain the device name. This will extend the search time significantly. Please note that if multiple applications are performing a scan for LE devices concurrently, either via the SD or directly from GATT, active scan has highest priority.
All other bits in the pattern	These bits are reserved for future use, and must be set to 0.

Figure 7 illustrates the use of the CSR_BT_SD_SEARCH_CONFIG_REQ signal.

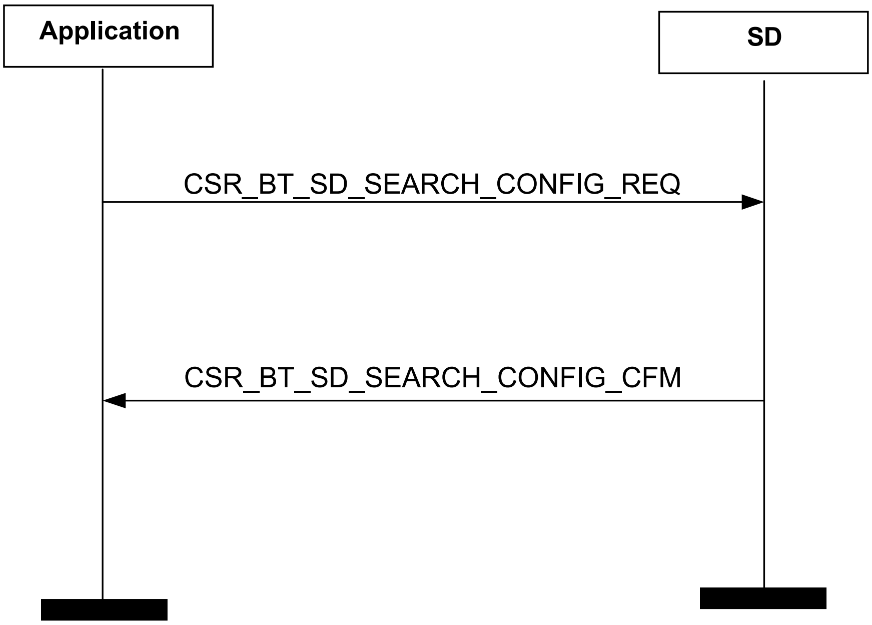


Figure 7: Sequence for the CSR_BT_SD_SEARCH_CONFIG_REQ

3.4.2 Setting the Class of Device

n order to provide full flexibility when specifying the Class of Device (CoD) for devices to search for, both a CoD and a CoD mask can be specified in a CSR_BT_SD_SEARCH_REQ (parameters *deviceClass* and *deviceClassMask*). The *deviceClassMask* specifies the significance of the bits in *deviceClass*, i.e. the *deviceClassMask* specifies which bits in the CoD that must match the bits of *deviceClass* and which bits that are allowed to differ.

An example with five different devices is found below:

	Major Service Class											Major Dev. Class					Minor Device Class							
Bit index	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Remote device 1 (mobile phone)	0	**1**	**1**	0	0	0	0	0	X	X	0	0	0	0	**1**	0	0	0	0	0	0	**1**	0	0
	Audio (21), Telephony (22)											Phone (9)					Cellular (2)							
Remote device 2 (printer)	0	0	0	0	0	**1**	0	0	X	X	0	0	0	**1**	**1**	0	**1**	0	0	0	0	0	0	0
	Rendering (18)											Imaging (9+10)					Printer (7)							
Remote device 3 (headset)	0	0	**1**	0	0	0	0	0	X	X	0	0	0	**1**	0	0	0	0	0	0	0	**1**	0	0
	Audio (21)											Audio/video (10)					Wearable headset (2)							
Remote device 4 (computer)	**1**	**1**	**1**	**1**	**1**	0	0	0	X	X	0	0	0	0	0	**1**	0	0	0	0	**1**	**1**	0	0
	Capturing (19), Object (20), Audio (21), Telephony (22), Information (23)											Computer (8)					Laptop (2+3)							
Remote device 5 (access point)	**1**	0	0	0	0	0	**1**	0	X	X	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0
	Networking (17), Information (23)											Access point (8+9)					Fully available (-)							

If the SD should detect a device with a Hands-Free profile, different solutions exists. All devices with this profile will have the 21st bit set in the Major Service Class. The Major Device Class can have different configurations as multiple sorts of devices can support the Hands-Free profile. In this example with five devices three of them potentially support the profile: A mobile phone, a headset and a computer. Normally a user will be interested in finding only the headset, which can be achieved using the following configuration of the *deviceClass* and *deviceClassMask* parameters in a CSR_BT_SD_SEARCH_REQ:

deviceClass	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0
	CSR_BT_AUDIO_MAJOR_SERVICE_MASK | CSR_BT_AV_MAJOR_DEVICE_MASK																							
deviceClassMask	0	0	**1**	0	0	0	0	0	0	0	0	**1**	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0
	CSR_BT_AUDIO_MAJOR_SERVICE_MASK | CSR_BT_SIGNIFICANT_MAJOR_DEVICE_MASK																							

This will result in ONLY the headset and not the computer or mobile phone to be found even though they support the profile of interest. If the SD should include all devices supporting the Hands-Free profile (and other profiles with the Audio bit set) in the search, the following configuration could be used:

deviceClass	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	CSR_BT_AUDIO_MAJOR_SERVICE_MASK																							
deviceClassMask	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	CSR_BT_AUDIO_MAJOR_SERVICE_MASK																							

One way of utilizing *deviceClassMask* is to search for a limited number of devices types in the initial search. If the device of interest is not found, the user can start a new search where no device classes should be excluded.

Generally it is NOT recommended to exclude devices based on the major device class but solely depend on the major service class, which implies that the Major Device Class bits in *deviceClassMask* should be set to 0.

A special use of the *deviceClassMask* is to set it to 0. In that case, all the found devices that have a class of device where some bits are common to the *deviceClass* will be accepted.

Another special case is when both the *deviceClassMask* and the *deviceClass* are set to 0. Then, all devices found will be accepted.

Note that if an application wants to search for devices that support either Major Service Class “Rendering”, “Audio” or both, it needs to either:

1. Send two CSR_BT_SD_SEARCH_REQs: One to search for devices supporting “Rendering” and one searching for devices supporting “Audio”.

Set the *deviceClassMask* to 0 and set the “Rendering” and “Audio” bits in the *deviceClass*. In this case, using the examples above, both the mobile phone, the headset and the computer will be found, because they all have the “Audio” bit set, and the printer device will be found as well, as it has the “Rendering” bit set. The access point device, however, would not be found as it has neither the “Audio” not the “Rendering” bits set.

3.4.3 Setting Inquiry Transmit Power Level

To specify the Inquiry Transmit Power Level the `CsrBtSdProximitySearchReqSend`-function is used. It takes an input parameter called `inquiryTxPowerLevel`, which specifies the Inquiry Transmit Power Level directly in dBm. The valid range is -70 → +20 dBm. Except for setting the Inquiry Transmit Power Level to a non-default value, the `CsrBtSdProximitySearchReqSend`-function works the same way as the `CsrBtSdSearchReqSend` –function, which are described elsewhere in this document.

An example of how the `CsrBtSdProximitySearchReqSend` –function is used, is found below:

```
CsrBtSdProximitySearchReqSend(APP_IFACEQUEUE,
        CSR_BT_SD_SEARCH_HIDE_PAIRED_DEVICES |
            CSR_BT_SD_SEARCH_FORCE_NAME_UPDATE,      /* Search configuration bitmask */
        5000,                                        /* RSSI buffer timeout [ms] */
        CSR_BT_INFINITE_TIME,                        /* Search timeout [ms] */
        CSR_BT_SD_RSSI_THRESHOLD_DONT_CARE,          /* RSSI threshold */
        CSR_BT_AUDIO_MAJOR_SERVICE_MASK |
            CSR_BT_AV_MAJOR_DEVICE_MASK,             /* Class of Device */
        CSR_BTAUDIO_MAJOR_SERVICE_MASK |
            CSR_BT_SIGNIFICANT_MAJOR_DEVICE_MASK,    /* Class of Device mask */
        CSR_BT_SD_ACCESS_CODE_GIAC,                  /* Inquiry access code */
        0,                                           /* Filter length */
        NULL,                                        /* Filter */
        -70);                                        /* Inquiry TX power level [dBm] */
```

To receive inquiry responses from devices in near proximity only, the `inquiryTxPowerLevel`-parameter must be set as low as possible, normally -70 for devices in very close proximity. If the actual transmit power of the BlueCore chip is -70 (or whatever number is used) depends on the PSKEY_LC_ENHANCED_POWER_TABLE. It is highly recommended that this power table contains an entry for Basic Rate Internal Amplifier setting = 0 to be able to set the inquiry range as low as possible. For further details on how to adjust the power table, please refer to CSR Application note CS-101532 (Enhanced Power Table Contruction).

If inquiry results from all devices within communication range are wanted, the `CsrBtSdSearchReqSend` function should be used, or the `inquiryTxPowerLevel`-parameter of the `CsrBtSdProximitySearchReqSend` function set to `CSR_BT_SD_DEFAULT_INQUIRY_TX_POWER_LEVEL`.

3.4.4 Example of a Typical Search

In the following is an example of the typical use of the search procedures of the SD. Consider a scenario where a user initiates a search for a headset.

Initial Search

In this scenario it is assumed that any paired devices is presented to the user before the actual search is started, so it is advantageous to suppress these devices during the search to avoid them from being displayed twice. This is done with the *CSR_BT_SD_SEARCH_HIDE_PAIRED_DEVICES* flag as previously described. To make sure that the names in the Device List are up-to-date the *CSR_BT_SD_SEARCH_FORCE_NAME_UPDATE* flag is also set in the first search:

```
CsrBtSdSearchReqSend(APP_IFACEQUEUE,
        CSR_BT_SD_SEARCH_HIDE_PAIRED_DEVICES |
            CSR_BT_SD_SEARCH_FORCE_NAME_UPDATE,      /* Search configuration bitmask */
        5000,                                        /* RSSI buffer timeout [ms] */
        CSR_BT_INFINITE_TIME,                        /* Search timeout [ms] */
        CSR_BT_SD_RSSI_THRESHOLD_DONT_CARE,          /* RSSI threshold */
        CSR_BT_AUDIO_MAJOR_SERVICE_MASK |
            CSR_BT_AV_MAJOR_DEVICE_MASK,             /* Class of Device */
        CSR_BTAUDIO_MAJOR_SERVICE_MASK |
            CSR_BT_SIGNIFICANT_MAJOR_DEVICE_MASK,    /* Class of Device mask */
        CSR_BT_SD_ACCESS_CODE_GIAC,                  /* Inquiry access code */
        0,                                           /* Filter length */
        NULL);                                       /* Filter */
```

In this example the *deviceClass* and *deviceClassMask* is specified to exclude devices of other classes than Audio/Video. As previously described the Major Device Mask should normally be set to 0, to make sure that all devices with matching Major Service Mask are found. The rest of the parameters in the request are described in section 4.5.

In most circumstances the device will be found using the above example. If this is not the case, the user will expectedly start a new search where one of the following settings could be used depending on why the initial search was closed.

Preceding Search (for result code CSR_BT_SD_RSSI_BUFFER_EMPTY)

If the initial search ended because all devices found during the buffering duration was processed (as described in section 3.2.4), it will most likely indicate that the majority of devices in the vicinity has been presented to the user. In this case the following preceding search can be used:

```
CsrBtSdSearchReqSend(APP_IFACEQUEUE,
        CSR_BT_SD_SEARCH_HIDE_PAIRED_DEVICES |
            CSR_BT_SD_SEARCH_SHOW_UNKNOWN_DEVICE_NAMES |
            CSR_BT_SD_SEARCH_DO_NOT_CLEAR_FILTER,        /* Search configuration bitmask */
        CSR_BT_SD_SEARCH_DISABLE_BUFFERING,              /* RSSI buffer timeout [ms] */
        CSR_BT_INFINITE_TIME,                            /* Search timeout [ms] */
        CSR_BT_SD_RSSI_THRESHOLD_DONT_CARE,              /* RSSI threshold */
        CSR_BT_AUDIO_MAJOR_SERVICE_MASK,                 /* Class of Device */
        CSR_BT_AUDIO_MAJOR_SERVICE_MASK,                 /* Class of Device mask */
        CSR_BT_SD_ACCESS_CODE_GIAC,                      /* Inquiry access code */
        0,                                               /* Filter length */
        NULL);                                           /* Filter */
```

Again paired devices are suppressed based on the assumptions that these were presented to the user initially. The *CSR_BT_SD_SEARCH_DO_NOT_CLEAR_FILTER* flag is set to avoid devices found in the previous search from being sent to the app again. If for some reason the headset does not reply to name requests, the *CSR_BT_SD_SEARCH_SHOW_UNKNOWN_DEVICE_NAMES* flag makes sure that the search result is sent to the application anyways. Since the majority of the devices is expected to be found, the benefit of using RSSI-sorting will be limited and is for this reason disabled by using *CSR_BT_SD_SEARCH_DISABLE_BUFFERING* as the parameter for the buffering duration.

Preceding Search (for result codes CSR_BT_SD_MAX_RESULTS or CSR_BT_SD_SEARCH_CANCELLED)

If the user cancels the initial search (result code *CSR_BT_SD_SEARCH_CANCELLED*) or the search closes with the result code *CSR_BT_SD_MAX_RESULTS* this could indicate that only a minority of the remote devices in the vicinity have been found. In this case, it would be advantageous to use RSSI-sorting like in the initial search, which is done in the following example:

```
CsrSdSearchReqSend(APP_IFACEQUEUE,
        CSR_BT_SD_SEARCH_HIDE_PAIRED_DEVICES |
            CSR_BT_SD_SEARCH_CONTINUE_AFTER_RSSI_SORTING |
            CSR_BT_SD_SEARCH_SHOW_UNKNOWN_DEVICE_NAMES |
            CSR_BT_SD_SEARCH_ALLOW_UNSORTED_SEARCH_RESULTS |
            CSR_BT_SD_SEARCH_DO_NOT_CLEAR_FILTER,            /* Search configuration bitmask */
        5000,                                                /* RSSI buffer timeout [ms] */
        CSR_BT_INFINITE_TIME,                                /* Search timeout [ms] */
        CSR_BT_SD_RSSI_THRESHOLD_DONT_CARE,                  /* RSSI threshold */
        CSR_BT_AUDIO_MAJOR_SERVICE_MASK,                     /* Class of Device */
        CSR_BT_AUDIO_MAJOR_SERVICE_MASK,                     /* Class of Device mask */
        CSR_BT_SD_ACCESS_CODE_GIAC,                          /* Inquiry access code */
        0,                                                   /* Filter length */
        NULL);                                               /* Filter */
```

The flags used are the same as in the previous example of a preceding search with the exception of *CSR_BT_SD_SEARCH_CONTINUE_AFTER_RSSI_SORTING*, which ensures that the search will not close before being cancelled from the application, the maximum number of devices is found or if the specified search time has elapsed. The final flag, *CSR_BT_SD_SEARCH_ALLOW_UNSORTED_SEARCH_RESULTS*, is set based on the fact that the benefit of enforcing sorting of search results according to the RSSI-level is highest when many unknown devices is present, which will rarely be the case in the second search.

3.5 Cancel Search for Remote Devices

This command cancels an active search. Be aware that this signal only cancels searches started by the application that sends this signal.

If an application sends this signal when it has no active searches, it is considered an error, therefore the signal is just ignored by the SD and hence no response is sent back to the application.

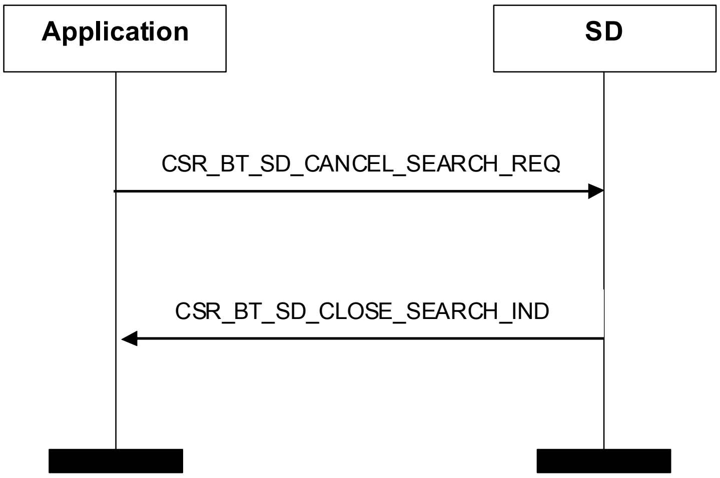


Figure 8: Cancel search sequence

3.6 Read SD Device List

This command allows an application to read all the information that the SD keeps in the Device List, which was described in section 3.1. The command can be configured to return different information from the Device List for example excluding of non-paired devices. Additionally the maximum amount of data to allocate in each CSR_BT_SD_READ_DEVICE_LIST_IND/CFM can be specified. CSR_BT_SD_READ_DEVICE_LIST_IND messages will be sent to the application until the last results can fit into a CSR_BT_SD_READ_DEVICE_LIST_CFM. This is described in section 4.12.

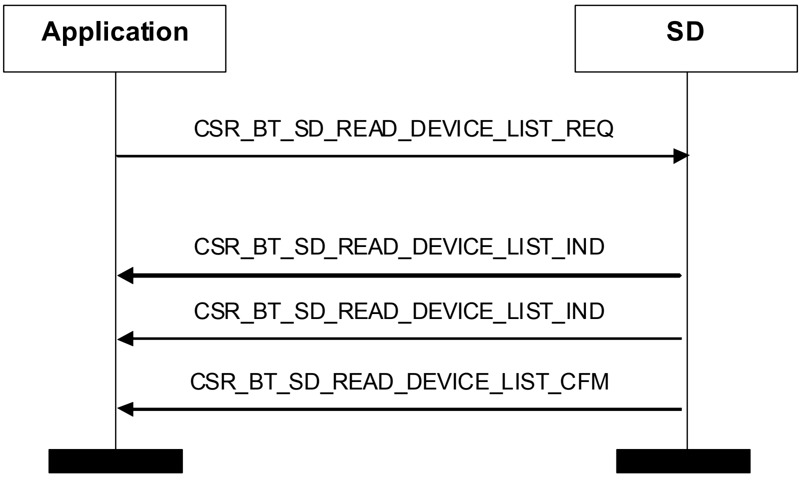

Application
SD
CSR_BT_SD_READ_DEVICE_LIST_REQ
CSR_BT_SD_READ_DEVICE_LIST_IND
CSR_BT_SD_READ_DEVICE_LIST_IND
CSR_BT_SD_READ_DEVICE_LIST_CFM

Figure 9: Read device list sequence

3.7 Read Information about a Device

This command allows an application to read all the information the SD holds about a given Bluetooth® device

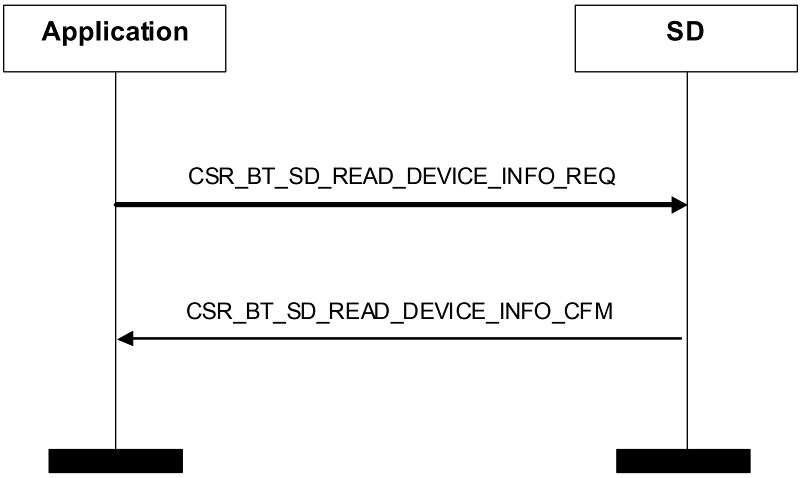

Application
SD
CSR_BT_SD_READ_DEVICE_INFO_REQ
CSR_BT_SD_READ_DEVICE_INFO_CFM

Figure 10: Read device info sequence

3.8 Read available Services

This command reads the available services of a remote Bluetooth device. Please remember that only one service search can be active per application at the time.

Note that only remote Bluetooth® device services, which matches services offered by the local device are returned in the confirm signal.

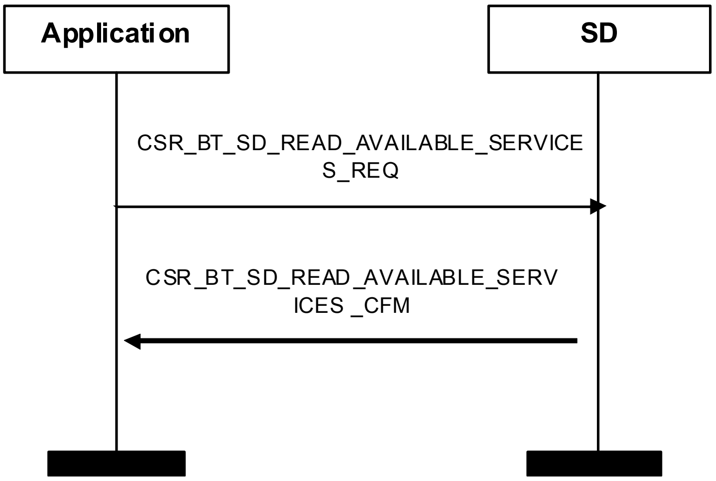

Application
SD
CSR_BT_SD_READ_AVAILABLE_SERVICES_REQ
CSR_BT_SD_READ_AVAILABLE_SERVICES _CFM

Figure 11: Read available services

3.9 Cancel Read available Services

This command allows an application to cancel an active read available services command. An application can only cancel a service search it has started itself. If the cancel read available services command is sent when the application has no active service searches, it is considered an error, therefore the signal is just ignored by the SD and hence no response is sent back to the application.

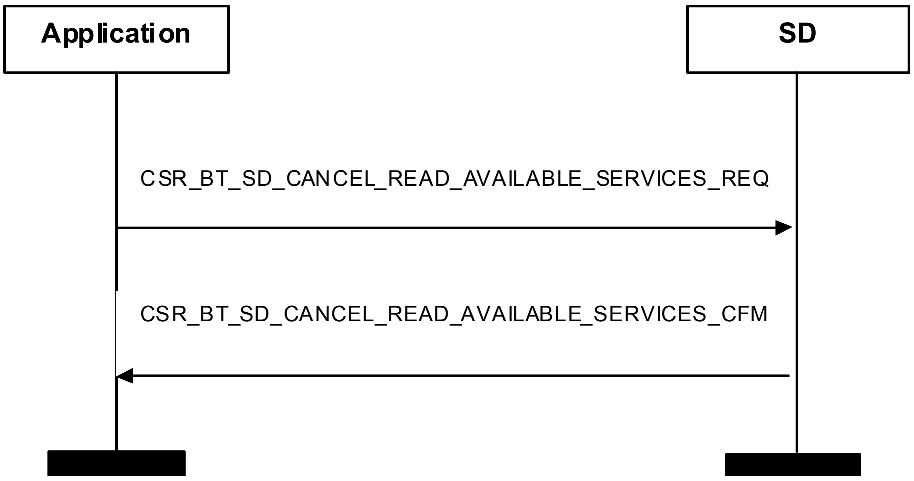

Application
SD
CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES_REQ
CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES_CFM

Figure 12: Cancel read available services

3.10 Cleaning up the SD

In order to minimize the memory consumption of the SD this primitive can be used for releasing different parts of the memory used by the SD. The most optimal place to utilize this functionality depends on the implementation, but when a user has performed a search for remote devices and selected a device in order to connect or initiate pairing, it can be assumed that another search is not immediately pending and therefore it would be beneficial to cleanup the SD.

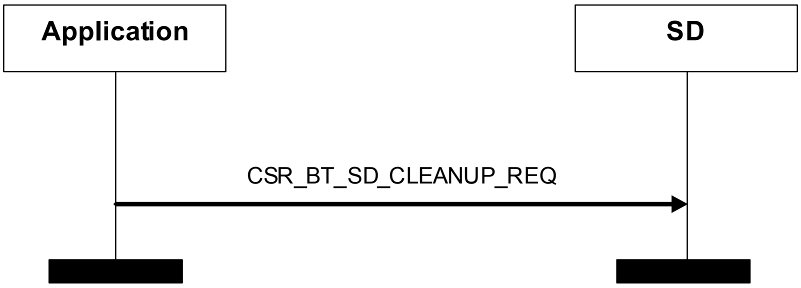


Figure 13: Cleanup

3.11 Typical Use Case of the SD

In order to give an idea of how to use the SD module a typical use case is depicted below.

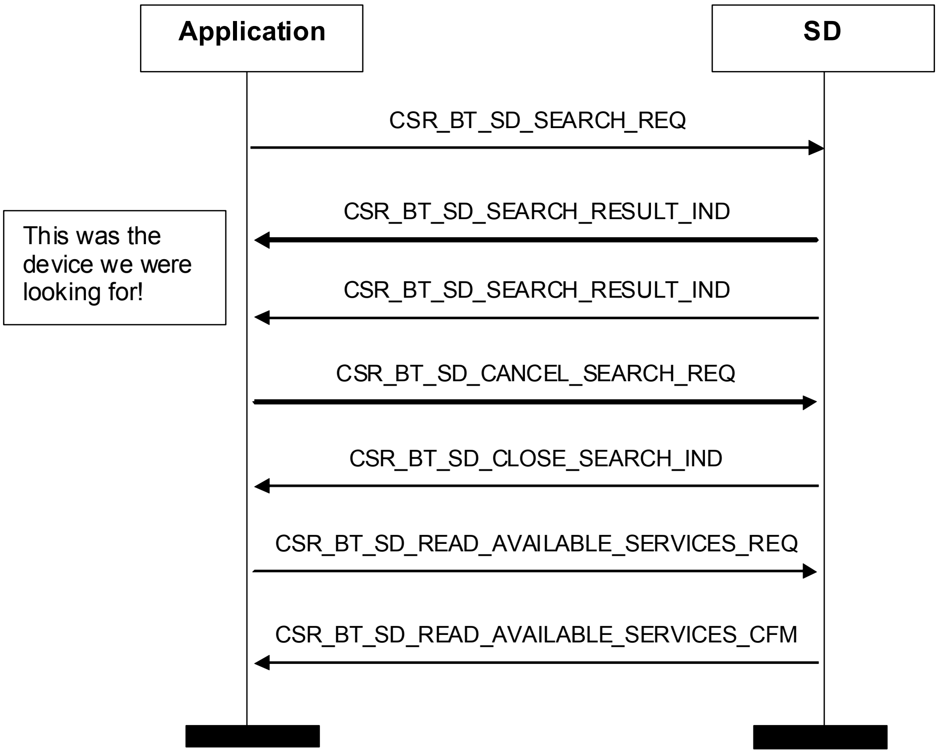

Application
SD
CSR_BT_SD_SEARCH_REQ
CSR_BT_SD_SEARCH_RESULT_IND
This was the device we were looking for!
CSR_BT_SD_SEARCH_RESULT_IND
CSR_BT_SD_CANCEL_SEARCH_REQ
CSR_BT_SD_CLOSE_SEARCH_IND
CSR_BT_SD_READ_AVAILABLE_SERVICES_REQ
CSR_BT_SD_READ_AVAILABLE_SERVICES_CFM

Figure 14: Typical use case of the SD module

The search is started and when the desired device is found the search is cancelled. Because the stored services for the remote device are not updated in a search unless extended inquiry results are used, the read available services command is issued to get a true updated list of which services the device of interest actually offers.

3.12 Synchronization with SCDB

In some systems the SCDB and/or the file system is not accessible during initialization of CSR Synergy BT, which may cause the synchronization between the SD Device List and the SCDB to fail. In these systems, it is necessary to send a CSR_BT_SD_SYNCHRONIZE_REQ to CSR Synergy BT.

When the synchronization is finished, the application will receive a CSR_BT_SD_SYNCHRONIZE_CFM primitive, which contains the total number of devices in the SCDB.

The synchronisation should only be performed immediately following a system boot and not during normal operation.

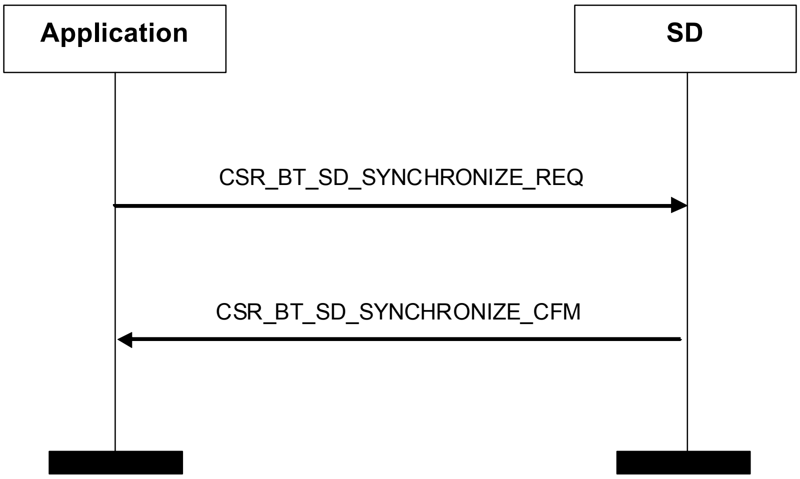


Figure 15: Synchronizing with SCDB

4 Service Discovery Primitives

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

4.1 List of All Primitives

Primitives	Reference
CSR_BT_SD_MEMORY_CONFIG_REQ	See section 4.2
CSR_BT_SD_SEARCH_CONFIG_REQ	See section 4.3
CSR_BT_SD_SEARCH_CONFIG_CFM	See section 4.3
CSR_BT_SD_READ_DEVICE_INFO_REQ	See section 4.4
CSR_BT_SD_READ_DEVICE_INFO_CFM	See section 4.4
CSR_BT_SD_SEARCH_REQ	See section 4.5
CSR_BT_SD_CANCEL_SEARCH_REQ	See section 4.6
CSR_BT_SD_IMMEDIATE_SEARCH_RESULT_IND	See section 4.7
CSR_BT_SD_SEARCH_RESULT_IND	See section 4.7
CSR_BT_SD_CLOSE_SEARCH_IND	See section 4.8
CSR_BT_SD_CLEANUP_REQ	See section 4.9
CSR_BT_SD_READ_AVAILABLE_SERVICES_REQ	See section 4.10
CSR_BT_SD_READ_AVAILABLE_SERVICES_CFM	See section 4.10
CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES_REQ	See section 4.11
CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES_CFM	See section 4.11
CSR_BT_SD_READ_DEVICE_LIST_REQ	See section 4.12
CSR_BT_SD_READ_DEVICE_LIST_IND	See section 4.12
CSR_BT_SD_READ_DEVICE_LIST_CFM	See section 4.12
CSR_BT_SD_SYNCHRONIZE_REQ	See section 4.13
CSR_BT_SD_SYNCHRONIZE_CFM	See section 4.13
CSR_BT_SD_READ_SERVICE_RECORD_REQ	See the **api-0136-di.pdf**
CSR_BT_SD_READ_SERVICE_RECORD_IND	See the **api-0136-di.pdf**
CSR_BT_SD_READ_SERVICE_RECORD_CFM	See the **api-0136-di.pdf**
CSR_BT_SD_CANCEL_READ_SERVICE_RECORD_REQ	See the **api-0136-di.pdf**
CSR_BT_SD_REGISTER_SERVICE_RECORD_REQ	See the **api-0136-di.pdf**
CSR_BT_SD_REGISTER_SERVICE_RECORD_CFM	See the **api-0136-di.pdf**
CSR_BT_SD_UNREGISTER_SERVICE_RECORD_REQ	See the **api-0136-di.pdf**
CSR_BT_SD_UNREGISTER_SERVICE_RECORD_CFM	See the **api-0136-di.pdf**
CSR_BT_SD_READ_SERVICES_REQ	See section 4.14
CSR_BT_SD_READ_SERVICES_CFM	See section 4.14
CSR_BT_SD_READ_SERVICES_CANCEL_REQ	See section 4.15
CSR_BT_SD_READ_SERVICES_CANCEL_CFM	See section 4.15

Table 1: List of all primitives

4.2 CSR_BT_SD_MEMORY_CONFIG_SIZE

Parameters **Primitives**	type	phandle	memoryConfig	deviceListMax	deviceListInfoMax	resultCode	resultSupplier
CSR_BT_SD_MEMORY_CONFIG_REQ	✓	✓	✓	✓	✓		
CSR_BT_SD_MEMORY_CONFIG_CFM	✓		✓	✓	✓	✓	✓

Table 2: CSR_BT_SD_MEMORY_CONFIG Primitives

Description

This signal is used for configuring the memory usage of the SD. This signal only needs to be used if corresponding values in csr_bt_usr_config.h must be changed during run-time. The SD sends a confirmation message to the originating task when the operation has been performed.

Parameters

type	Signal identity, CSR_BT_SD_MEMORY_CONFIG_REQ/CFM.
phandle	The identity of the calling process.
memoryConfig	Bitmask specifying configuration flags – this value is reserved and shall be set to CSR_BT_SD_MEMORY_CONFIG_USE_STANDARD.
deviceListMax	Specifies the total maximum number of entries in the Device List – this parameter must be set to CSR_BT_UNLIMITED or at least the same size as deviceListMax.
deviceListInfoMax	Specifies the maximum number of entries with extended info in the Device List. Must be set to at least 1.
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

The function:

```
    void CsrBtSdMemoryConfigReqSend(        CsrQid    phandle,
                                            CsrUint32    memoryConfig,
                                            CsrUint32 deviceListMax,
                                            CsrUint32 deviceListInfoMax)
```

defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_MEMORY_CONFIG_REQ primitive to SD.

4.3 CSR_BT_SD_SEARCH_CONFIG

Parameters Primitives	type	phandle	searchConfig	readNameTimeout	maxSearchResults	resultCode	resultSupplier
CSR_BT_SD_SEARCH_CONFIG_REQ	✓	✓	✓	✓	✓		
CSR_BT_SD_SEARCH_CONFIG_CFM	✓		✓	✓	✓	✓	✓

Table 3: CSR_BT_SD_SEARCH_CONFIG Primitives

Description

This signal provides runtime configuration of various parameters controlling search behaviour. It should only be used if corresponding parameters in csr_bt_usr_config.h should be changed during runtime. The SD will send a confirmation message to the originating task when the operation has been performed

Parameters

type Signal identity, CSR_BT_SD_SEARCH_CONFIG_REQ/CFM.

phandle The identity of the calling process.

searchConfig Bitmask containing a set of behaviour defining flags. For details refer to section 3.3.

readNameTimeout The timeout in milliseconds for reading remote names through the SD. This is explained in section 3.3.

maxSearchResults The maximum number of results to return in a device search. For details, refer to section 3.3.

resultCode The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.

resultSupplier This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

The function:

```
void CsrBtSdSearchConfigReqSend(CsrQid phandle,

                                   CsrUint32 searchConfig,

                                   CsrUint32 readNameTimeout,

                                   CsrUint32 maxSearchResults);
```

defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_SEARCH_CONFIG_REQ primitive to SD.

4.4 CSR_BT_SD_READ_DEVICE_INFO

Primitives \ Parameters	type	phandle	deviceAddr	deviceClass	resultCode	resultSupplier	*info	infoLen	deviceStatus	addressType	context	serviceChangedHandle
CSR_BT_SD_READ_DEVICE_INFO_REQ	✓	✓	✓							✓	✓	
CSR_BT_SD_READ_DEVICE_INFO_CFM	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 4: CSR_BT_SD_READ_DEVICE_INFO Primitives

Description

This signal allows an application to request the SD for all known information about a specific device.

Parameters

type Signal identity, CSR_BT_SD_READ_DEVICE_PROPERTIES_REQ/CFM.

phandle The identity of the calling process. The response is returned to phandle.

deviceAddr The Bluetooth address of the device of interest.

resultCode The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.

resultSupplier This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

deviceClass The Class of Device for the remote device.

*info A byte stream containing information about the device. Refer to section 3.2.1 for details.

infoLen The length in bytes of the info field.

deviceStatus A bitmask containing the status of the device. Refer to section 3.2.3 for details.

context Application-specific context. The value in the request is returned in the confirm.

addressType Peer device address type. For public and standard Bluetooth, this is always CSR_BT_ADDR_PUBLIC. Low Energy devices with privacy enabled may have this set to CSR_BT_ADDR_PRIVATE.

serviceChangedHandle For internal use only.

The function:

```
void CsrBtSdReadDeviceInfoReqSend(          CsrQid    phandle,
                                            deviceAddr_t deviceAddr)
```

defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_READ_DEVICE_INFO_REQ primitive to SD.

4.5 CSR_BT_SD_SEARCH

Parameters Primitives	type	phandle	searchConfig	rssiBufferTime	totalSearchTime	rssiThreshold	deviceClass	deviceClassMask	inquiryAccessCode	filterLen	*filter	leRssiThreshold
CSR_BT_SD_SEARCH_REQ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5: CSR_BT_SD_SEARCH Primitives

Description

Start Bluetooth device discovery of devices in the vicinity. Only devices that are made discoverable will be found during this process. Note that you cannot be seen by other devices, while running the discovery process.

Parameters

type Signal identity, CSR_BT_SD_SEARCH_REQ.

phandle The identity of the calling process. The response is returned to phandle.

searchConfig The search configuration used for this search. Refer to section 3.4 for details.

rssiBufferTime The time in milliseconds used for uninterrupted inquiry to fill the RSSI buffer before searching for names. A value of CSR_BT_SD_SEARCH_DISABLE_BUFFERING will disable the use of the RSSI buffer. Refer to section 3.2.4 for details.

totalSearchTime Defines the maximum number of milliseconds the discovery procedure will continue if not terminated due to the number of responses or an empty RSSI buffer. The maximum timeout time allowed is defined by the CSR_BT_MAX_TIME in the file “profiles.h”. A value of CSR_BT_INFINITE_TIME will disable the timeout.

rssiThreshold Defines a RSSI threshold value that a search result must meet in order to result in a CSR_BT_SD_SEARCH_IND. The value CSR_BT_SD_RSSI_THRESHOLD_DONT_CARE will result in all results to be sent to the application.

deviceClass A device class that must be matched in order to generate a search result. Significant bits in this mask are defined by deviceClassMask. Refer to section 3.4.2 for an example.

deviceClassMask Defines the significant bits of deviceClass. This means that bits in deviceClass corresponding to bits that are set in deviceClassMask must all match the corresponding bits in the Class of Device of the remote device in order to cause a result indication for this device. Refer to section 3.4.2 for an example.

inquiryAccessCode Defines the Inquiry Access Code to use for the search. Normally this is set to CSR_BT_SD_ACCESS_CODE_GIAC, but all valid codes are supported.

filterLen This parameter has been added for future use. Set to 0 for now.

filter This parameter has been added for future use. Set to NULL for now.

inquiryTxPowerLevel Specifies the Inquiry Transmit Power Level. Valid range is -70 to +20 dBm. Refer to Section 3.4.3 for details leRssiThreshold As for ‘rssiThreshold’, but value is used for the low energy radio advertising packets.

The function:

```
void CsrBtSdSearchReqSend( CsrQid phandle,
                          CsrUint32 searchConfig,
                          CsrUint32 rssiBufferTime,
                          CsrUint32 totalSearchTime,
                          CsrInt8 rssiThreshold,
                          classOfDevice_t deviceClass,
                          classOfDevice_t deviceClassMask,
                          CsrUint24 inquiryAccessCode,
                          CsrUint32 filterLen,
                          CsrUint8 *filter);
```

is defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_SEARCH_REQ primitive to SD.

The function:

```
void CsrBtSdProximitySearchReqSend( CsrQid phandle,
                                    CsrUint32 searchConfig,
                                    CsrUint32 rssiBufferTime,
                                    CsrUint32 totalSearchTime,
                                    CsrInt8 rssiThreshold,
                                    classOfDevice_t deviceClass,
                                    classOfDevice_t deviceClassMask,
                                    CsrUint24 inquiryAccessCode,
                                    CsrUint32 filterLen,
                                    CsrUint8 *filter,
                                    CsrInt8 inquiryTxPowerLevel);
```

is defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_SEARCH_REQ primitive to SD and makes it possible to specify the Inquiry Transmit Power Level.

Two extended functions also exist; `CsrBtSdSearchReqSendEx()` and `CsrBtSdProximitySearchReqSendEx()`. They take the same parameters, plus an extra parameter: `CsrInt8 leRssiThreshold`.

4.6 CSR_BT_SD_CANCEL_SEARCH

Parameters / Primitives	type	phandle
CSR_BT_SD_CANCEL_SEARCH_REQ	✓	✓

Table 6: CSR_BT_SD_CANCEL_SEARCH Primitives

Description

This signal is used for cancelling an active search request. Please be aware that a state event error will happen, if this signal is sent when no search has been started in advance.

Parameters

type	Signal identity, CSR_BT_SD_CANCEL_SEARCH_REQ.
phandle	The identity of the calling process. The response is returned to phandle.

The function:

```
        void CsrBtSdCancelSearchReqSend (  CsrQid    phandle)
```

defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_CANCEL_SEARCH_REQ primitive to SD.

4.7 CSR_BT_SD_SEARCH_RESULT / CSR_BT_SD_IMMEDIATE_SEARCH_RESULT

Parameters **Primitives**	type	deviceAddr	deviceClass	rssi	infoLen	*info	deviceStatus
CSR_BT_SD_SEARCH_RESULT_IND	✓	✓	✓	✓	✓	✓	✓
CSR_BT_SD_IMMEDIATE_SEARCH_RESULT_IND	✓	✓	✓	✓	✓	✓	✓

Table 7: CSR_BT_SD_SEARCH_RESULT/CSR_BT_SD_IMMEDIATE_SEARCH_RESULT Primitives

Description

If immediate search results have been enabled a CSR_BT_SD_IMMEDIATE_SEARCH_RESULT_IND will be sent to the application momentarily when a new device is discovered. Name and/or services will only be available in this primitive if the remote device supports Extended Inquiry or if the information is already present in the Device List. Immediate search results will never be sorted according to the RSSI level

A CSR_BT_SD_SEARCH_RESULT_IND will be sent to the application every time a device matching the search criteria from the CSR_BT_SD_SEARCH_REQ is found. If RSSI sorting is enabled these results will not be sent before the buffering period has completed.

Please note that the result indications are always returned to the process handle that was included in the CSR_BT_SD_SEARCH_REQ.

Parameters

type — Signal identity, CSR_BT_SD_SEARCH_RESULT_IND or CSR_BT_SD_IMMEDIATE_SEARCH_RESULT_IND

deviceAddr — The Bluetooth address of the remote device.

deviceClass — The Class of Device of the remote device

rssi — The RSSI level of the remote device. This is measured in dBm and has a range of -127 to 20.

infoLen — The length of the info structure.

*info — A byte stream containing extra information about the device. See section 3.2.1 for details.

deviceStatus — A bitmask containing status of the device. See section 3.2.3 for details.

4.8 CSR_BT_SD_CLOSE_SEARCH

Primitives \ Parameters	type	resultCode	resultSupplier
CSR_BT_SD_CLOSE_SEARCH_IND	✓	✓	✓

Table 8: CSR_BT_SD_CLOSE_SEARCH Primitives

Description

This signal is sent to the application from the SD, when a search procedure has completed. The reason is indicated in the result parameter as described below.

Parameters

type — Signal identity, CSR_BT_SD_CLOSE_SEARCH_IND.

resultCode — The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.

resultSupplier — This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

4.9 CSR_BT_SD_CLEANUP

Parameters Primitives	type	phandle	cleanupMode	resultCode	resultSupplier
CSR_BT_SD_CLEANUP_REQ	✓	✓	✓		
CSR_BT_SD_CLEANUP_CFM	✓		✓	✓	✓

Table 9: CSR_BT_SD_CLEANUP_REQ Primitive

Description

This signal is sent to the application from the SD, when a search procedure has completed. The reason is indicated in the result parameter as described below.

Parameters

type — Signal identity, CSR_BT_SD_CLEANUP_REQ/CFM.

phandle — The identity of the calling process.

cleanupMode — This parameter must be set to one of the following values:

CSR_BT_SD_CLEANUP_EVERYTHING: The entire Device List (except paired devices), all state information about earlier searches will be removed
CSR_BT_SD_CLEANUP_INSTANCE: All entries in the Device List and state information specific to the application sending the cleanup primitive will be removed
CSR_BT_SD_CLEANUP_DEVICE_LIST: The entire Device List (except paired devices) will be released

resultCode — The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.

resultSupplier — This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

4.10 CSR_BT_SD_READ_AVAILABLE_SERVICES

Parameters Primitives	type	phandle	deviceAddr	serviceConfig	filterLen	*filter	resultCode	resultSupplier	deviceClass	infoLen	*info	deviceStatus
CSR_BT_SD_READ_AVAILABLE_SERVICES_REQ	✓	✓	✓	✓	✓	✓						
CSR_BT_SD_READ_AVAILABLE_SERVICES_CFM	✓		✓				✓	✓	✓	✓	✓	✓

Table 10: CSR_BT_SD_READ_AVAILABLE_SERVICES Primitives

Description

This signal is used for reading services offered by a remote Bluetooth® device.

Please note that the service search only searches for services at the remote Bluetooth® device, which matches services offered by the local device.

Parameters

type — Signal identity, CSR_BT_SD_READ_AVAILABLE_SERVICES_REQ/CFM.

phandle — The identity of the calling process. The response is returned to phandle.

deviceAddr — The Bluetooth® device address.

serviceConfig — A bitmask for modifying behaviour of the search. The following values defined in csr_bt_sd_prim.h are currently valid:

CSR_BT_SD_SERVICE_USE_STANDARD: Use the standard search behaviour of the SD.

CSR_BT_SD_SERVICE_NO_NAME_UPDATE: Do not update the name of the remote device during search.

All other bits: These bits are reserved for future use, and must be set to 0.

filterLen — This parameter has been added for future use. Set to 0 for now.

*filter — This parameter has been added for future use. Set to NULL for now.

resultCode — The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.

resultSupplier — This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

deviceClass — The Class of Device of the remote device

infoLen — The length of the info structure.

*info	A byte stream containing extra information about the device. See section 3.2.1 for details.
deviceStatus	A bitmask containing status of the device. See section 3.2.3 for details.

The function:

```
        void CsrBtSdReadAvailableServicesReqSend( CsrQid    phandle,
                                                 deviceAddr_t deviceAddr,
                                                 CsrUint32 serviceConfig,
                                                 CsrUint32 filterLen,
                                                 CsrUint8  *filterLen)
```

defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_READ_AVAILABLE_SERVICES_REQ primitive to SD.

4.11 CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES

Parameters / **Primitives**	type	phandle
CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES_REQ	✓	✓
CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES_CFM	✓	

Table 11: CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES Primitives

Description

This signal is used for cancelling an active service search.

Please note that sending this signal when no service search is active will result in a state event error.

Parameters

type	Signal identity, CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES_REQ/CFM.
phandle	The identity of the calling process. The response is returned to phandle.

The function:

```
void CsrBtSdCancelReadAvailableServicesReqSend( CsrQid    phandle)
```

defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_CANCEL_READ_AVAILABLE_SERVICES_REQ primitive to SD.

4.12 CSR_BT_SD_READ_DEVICE_LIST

Parameters / Primitives	type	phandle	maxNumOfBytesInEachResult	deviceListConfig	deviceInfoCount	*deviceInfo	totalNumOfDevices
CSR_BT_SD_READ_DEVICE_LIST_REQ	✓	✓	✓	✓			
CSR_BT_SD_READ_DEVICE_LIST_IND	✓				✓	✓	
CSR_BT_SD_READ_DEVICE_LIST_CFM	✓				✓	✓	✓

Table 12: CSR_BT_SD_READ_DEVICE_LIST Primitives

Description

This signal allows another process to read all information in the Device List. If the complete list of devices is larger than *maxNumOfBytesInResult* the SD sends Indication(s) until the rest of the list is smaller than *maxNumOfBytesInResult* and thus can fit within the confirm signal.

Parameters

type — Signal identity, CSR_BT_SD_READ_DEVICE_LIST_REQ/IND/CFM.

phandle — The identity of the calling process. The response is returned to phandle.

maxNumOfBytesInEachResult — The maximum number of bytes that is allowed in a message to the calling process.

deviceListConfig — A bitmask defining restrictions on which device type to include in the result. The following bits have been defined in csr_bt_sd_prim.h:

CSR_BT_SD_DEVICE_USE_STANDARD: Use the standard settings for reading the Device List – this corresponds to the negation of the following flags.

CSR_BT_SD_DEVICE_EXCLUDE_NON_PAIRED_DEVICES: Do not include non-paired devices.

CSR_BT_SD_DEVICE_EXCLUDE_PAIRED_DEVICES: Do not include paired devices.

All other bits: These bits are reserved for future use, and must be set to 0.

deviceInfoCount — Specifies the number of devices contained in the allocated pointer *deviceInfo*.

*deviceInfo — Pointer to structure of the *CsrBtsdDeviceInfoType* – defined in csr_bt_sd_prim.h. The pointer is a list covering all information the SD holds about devices requested. Please be aware that it is the calling process' responsibility to CsrPfree this pointer in order to avoid memory leak.

totalNumOfDevices — Specifies the total number of devices returned.

The function:

```
        void CsrBtSdReadDeviceListReqSend ( CsrQid phandle,

                                                CsrUint32 maxNumOfBytesInEachResult,

                                                CsrUint32 deviceListConfig)
```

4.13 CSR_BT_SD_SYNCHRONIZE

Parameters Primitives	type	phandle	totalNumOfDevices
CSR_BT_SD_READ_DEVICE_LIST_REQ	✓	✓	
CSR_BT_SD_READ_DEVICE_LIST_CFM	✓		✓

Table 13: SD_SYNCHRONIZE Primitives

Description

In some systems the SCDB and/or the file system is not accessible during initialization of CSR Synergy BT, which may cause the synchronization between the SD Device List and the SCDB to fail. In these systems it is necessary to send a CSR_BT_SD_SYNCHRONIZE_REQ to CSR Synergy BT.

Parameters

type	Signal identity, CSR_BT_SD_READ_DEVICE_LIST_REQ/IND/CFM.
phandle	The identity of the calling process. The response is returned to phandle.
totalNumOfDevices	Specifies the total number of devices returned.

The function:

```
        void CsrBtSdSynchronizeReqSend ( CsrQid    phandle)

                                                defined in csr_bt_sd_lib_h, builds
                                                and sends the
                                                CSR_BT_SD_SYNCHRONIZE_REQ primitive
                                                to SD.
```

4.14 CSR_BT_SD_READ_SERVICES

Parameters **Primitives**	type	phandle	deviceAddr	serviceConfig	filterLen	*filter	resultCode	resultSupplier	deviceClass	listCount	*list	deviceStatus
CSR_BT_SD_READ_SERVICES_REQ	✓	✓	✓	✓	✓	✓						
CSR_BT_SD_READ_SERVICES_CFM	✓		✓				✓	✓	✓	✓	✓	✓

Table 14: CSR_BT_SD_READ_SERVICES Primitives

Description

This signal is used for reading services offered by a remote Bluetooth® device. This API will return *all* services supported on the peer with no regard to what the local device supports.

Parameters

type

Signal identity, CSR_BT_SD_READ_SERVICES_REQ/CFM.

phandle

The identity of the calling process. The response is returned to phandle.

deviceAddr

The Bluetooth® device address.

serviceConfig

A bitmask for modifying behaviour of the search. The following values defined in csr_bt_sd_prim.h are currently valid:

CSR_BT_SD_SERVICE_USE_STANDARD: Use the standard search behaviour of the SD.

CSR_BT_SD_SERVICE_NO_NAME_UPDATE: Do not update the name of the remote device during search.

All other bits: These bits are reserved for future use, and must be set to 0.

filterLen

This parameter has been added for future use. Set to 0 for now.

*filter

This parameter has been added for future use. Set to NULL for now.

resultCode

The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.

resultSupplier

This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

deviceClass

The Class of Device of the remote device

listCount

Number of CsrBtUuid elements in *list.

*info

List of service UUIDs supported by the peer.

deviceStatus A bitmask containing status of the device. See section 3.2.3 for details.

The function:

```
        void CsrBtSdReadServicesReqSend(CsrQid phandle,
                                        deviceAddr_t deviceAddr,
                                        CsrUint32 serviceConfig,
                                        CsrUint32 filterLen,
                                        CsrUint8 *filterLen)
```

defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_READ_SERVICES_REQ primitive to SD.

4.15 CSR_BT_SD_ READ_SERVICES_CANCEL

Parameters Primitives	type	phandle
CSR_BT_SD_READ_SERVICES_CANCEL_REQ	✓	✓
CSR_BT_SD_READ_SERVICES_CANCEL_CFM	✓	

Table 15: CSR_BT_SD_READ_SERVICES_CANCEL Primitives

Description

This signal is used for cancelling an active service search.

Please note that sending this signal when no service search is active will result in a state event error.

Parameters

type	Signal identity, CSR_BT_SD_READ_SERVICES_CANCEL_REQ/CFM.
phandle	The identity of the calling process. The response is returned to phandle.

The function:

```
void CsrBtSdReadServicesCancelReqSend(CsrQid    phandle)
```

defined in csr_bt_sd_lib.h, builds and sends the CSR_BT_SD_READ_SERVICES_CANCEL_REQ primitive to SD.

5 Document References

Document	Reference
Bluetooth Core Specification v.1.1, v1.2 and v2.0, section N/A	[BT]
Bluetooth Security Architecture, Version 1.0, section N/A	[SEC]
The whitepaper: Bluetooth User Interface Flow Diagrams for Bluetooth Secure Simple Pairing. Revision v1.0.	[UIF]
CSR Synergy Bluetooth, Porting guide, gu-0102-porting.pdf	[PORT]

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
UniFi™	Group term for CSR's range of chips designed to meet IEEE 802.11 standards
SD	Service Discovery
SC	Security Controller
SIG	Special Interest Group

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.