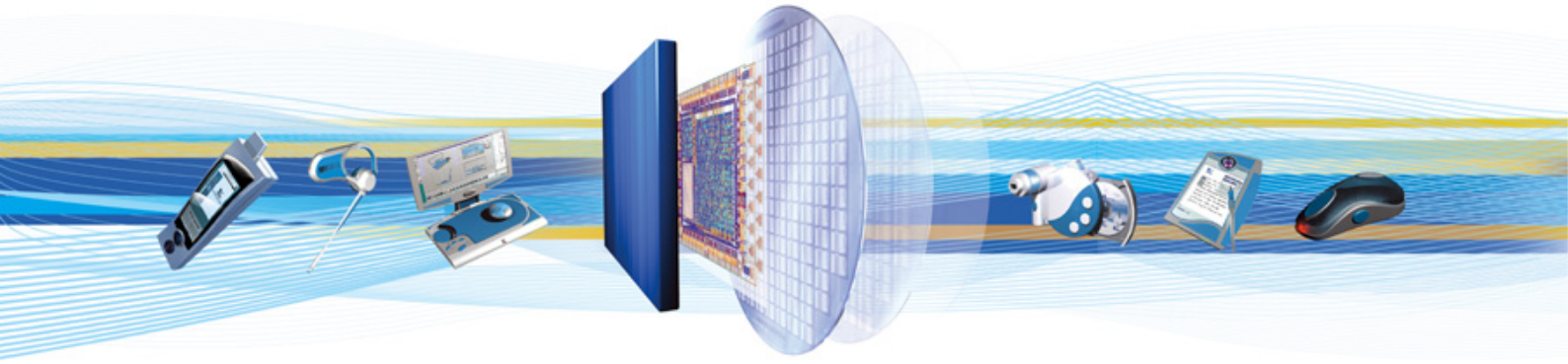
CSR Synergy Bluetooth 18.2.0

Porting Guide

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

1 Introduction 3
1.1 General Information 3
1.2 Audience 3
1.3 Reading Information 3
2 Porting Model 4
3 Persistent Memory Access 6
3.1 Database Structure 6
3.2 CsrBtscDbRead 7
3.3 CsrBtscDbWrite 7
3.4 CsrBtscDbRemove 7
3.5 CsrBtscDbReadFirst 7
3.6 CsrBtscDbReadNext 8
3.7 CSR Synergy Bluetooth Reference Implementation 9
4 SCO Audio Interface 10
4.1.1 Audio over PCM 10
4.1.2 Audio over BCSP (UART), H4DS (UART) or USB 10
5 Document References 12

List of Figures

Figure 1 CSR Synergy Concept 3
Figure 2: Porting model 4
Figure 3: HCI SCO data packet 10

1 Introduction

1.1 General Information

CSR Synergy Bluetooth is developed to work with CSR's family of BlueCore ICs and fits into the CSR Synergy Concept shown in Figure 1, together with other CSR Synergy Technologies.

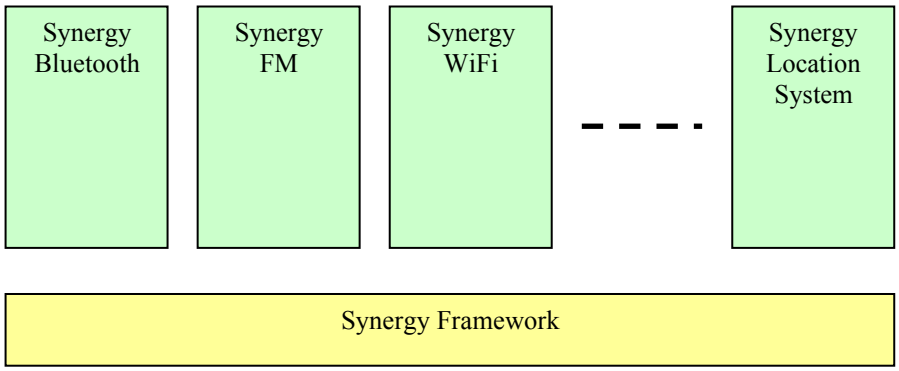


Figure 1 CSR Synergy Concept

CSR Synergy Bluetooth is intended for embedded products having a host processor for running the CSR Synergy Bluetooth and the Bluetooth® application. CSR Synergy Bluetooth together with the BlueCore IC is a complete Bluetooth® system solution from RF to profiles. CSR Synergy Bluetooth includes much of the Bluetooth® intelligence and gives the user a simple API. This makes it possible to develop a Bluetooth® product without in-depth Bluetooth® knowledge.

1.2 Audience

This porting guide gives an overview of what tasks is involved in porting CSR Synergy Bluetooth to a platform defined by the end user. This document is a supplement to the gu-0101-users_guide.pdf and must be read in conjunction with this.

The information contained in this porting guide is intended for system architects and system designers designing a system where CSR Synergy Bluetooth must be included as a system component. Besides having in-depth knowledge of C-programming and real-time programming, the following competences are necessary for a successful porting of CSR Synergy Bluetooth:

- Target OS/kernel and how it is used by current applications
- Knowledge of the ported scheduler that is part of the CSR Synergy Framework

1.3 Reading Information

The focus of this document is to describe how CSR Synergy Bluetooth is ported and not how the CSR Synergy Framework is ported. For description of the porting of the CSR Synergy Framework please see the documentation in the CSR Synergy Framework.

This document is divided into chapters for the different areas that need consideration during porting. For detailed information about the various profile APIs, please refer to the API documentation for the relevant profile.

2 Porting Model

By combining the CSR Synergy Framework and CSR Synergy Bluetooth, everything required for easy development of Bluetooth® applications is available.

A central point regarding CSR Synergy Bluetooth is that it is designed to be portable. CSR Synergy Bluetooth is therefore designed with minimal dependencies on external systems and this again implies that CSR Synergy Bluetooth executes with a very limited set of external functions.

Figure 2 illustrates the main components in a system with CSR Synergy framework and CSR Synergy Bluetooth integrated and the external dependencies of CSR Synergy Bluetooth. Please note that only a subset of the supplied Profile Managers is included in the figure.

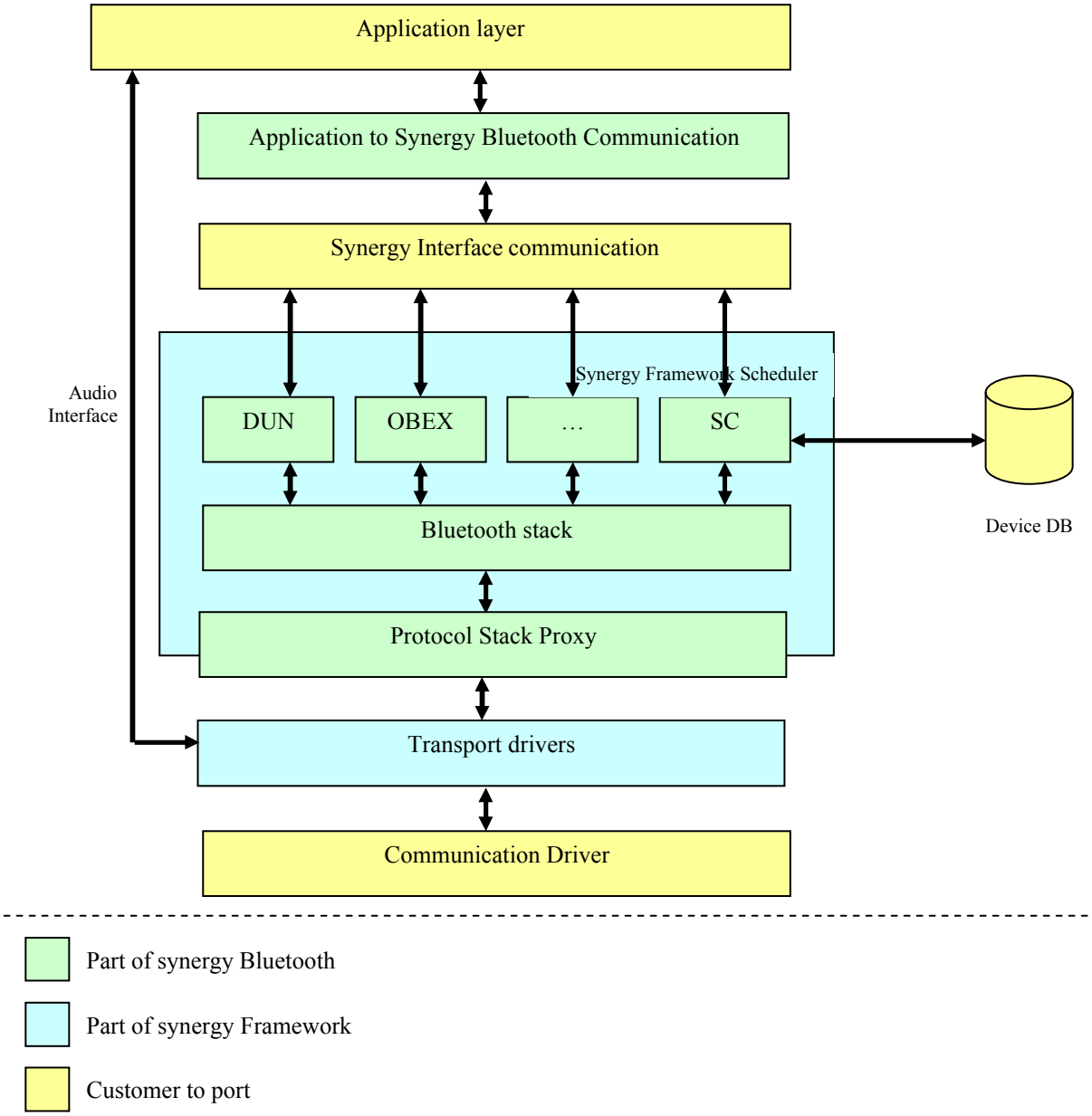


Figure 2: Porting model

The figure illustrates how the CSR Synergy Bluetooth components execute within a Scheduler or a Scheduler API. Apart from the Scheduler API, arrows represent the external interfaces. The number of external interfaces is kept to a minimum in order to ease integration into other subsystems. The external interfaces, which need to be ported, can be divided into:

1. **Scheduler API**, which is part of the CSR Synergy Framework and described in [API].
2. **Lower layer interface** for host to BlueCore communication, which is also part of the CSR Synergy Framework.
3. **Upper layer interface** symbolised by the CSR Synergy Communication Interface box.
4. **Persistent Memory Access** for storing of link keys of bonded devices symbolised by the Device DB
5. **Audio interface**

The three first interfaces (Scheduler API, Upper Level interface and Lower level interface) is part of the porting of the CSR Synergy Framework and will not be described here, but is described in the CSR Synergy Framework.

The remaining interfaces (Persistent Memory Access and Audio interface) will be described in more details in the following sections.

3 Persistent Memory Access

CSR Synergy Bluetooth includes information management for bonded devices, which is used by the Security Controller (SC). This information is things like: link key, remote name, remote class of device, remote services. In order to preserve this information when devices are powered off, CSR Synergy Bluetooth must have access to persistent memory where this information can be stored and maintained. All information are indexed by a Bluetooth address and the persistent memory access is modelled as a database access where the Bluetooth address is used as the index key.

As the actual implementation of the persistent memory is platform dependent an API has been defined that the security controller (SC) within CSR Synergy Bluetooth rely on and which shall be implemented in order to get CSR Synergy Bluetooth working.

The API defines functions for reading, writing, removing etc. entries in the persistent database.

The degree of persistent depend on the implementation, this means that if an implementation holds the information in memory these information will be lost when the power is switched off.

In the next section, the structures and functions needed to be ported are described in more details.

3.1 Database Structure

The database must define a structure with at least the following fields:

```
typedef struct
{
    BD_ADDR_T          deviceAddr;
    CsrBool             linkkeyValid;
    CsrUint8            linkkeyLen;
    deviceLinkkey_t     Linkkey;
    classOfDevice_t     classOfDevice;
    deviceName_t        remoteName;
    CsrUint32               knownServices11_00_31;
    CsrUint32               knownServices11_32_63;
    CsrUint32               knownServices12_00_31;
    CsrUint32               knownServices13_00_31;
    CsrBool                authorised;
    CsrUint8            linkkeyType;
}  CsrBtscDbDeviceRecord_t;
```

Note: additional fields are required for AMP and Bluetooth low energy.

The structure is defined in the csr_bt_sc_db.h file. Additional fields may be added, but the listed fields may not be removed.

3.2 CsrBtscDbRead

This function is used for reading from the database and has the following prototype:

```
CsrBool CsrBtscDbRead(deviceAddr_t *deviceAddr,
                      CsrBtscDbDeviceRecord_t *theRecord);
```

where `deviceAddr` is the Bluetooth address of the device and `theRecord` is a pointer to an already allocated memory block of at least `sizeof(CsrBtscDbDeviceRecord_t)`.

If a device could be found that matches the `deviceAddr` `TRUE` is return else `FALSE` is returned.

The function will be called on the following scenarios:

- A new link key has been calculated, either because the local or remote device requested it, or in order to compare it to an already existing link-key (triggered by low-level HCI stack)
- The encryption mode changes (triggered by CSR_BT_SC_ENCRYPTION_REQ)
- When the trust-level changes (triggered by CSR_BT_SC_SET_TRUST_LEVEL_REQ)
- When the SD or SC reads device properties (triggered by CSR_BT_SD_READ_DEVICE_PROPERTIES_CFM)
- Device services etc. needs to be updated (triggered by CSR_BT_SC_UPDATE_DEVICE_REQ)

3.3 CsrBtscDbWrite

This function is used for writing an entry into the database and has the following prototype:

```
void CsrBtscDbWrite(deviceAddr_t *deviceAddr,
                    CsrBtscDbDeviceRecord_t *theRecord);
```

where `deviceAddr` is the Bluetooth address of the device and `theRecord` is a pointer to the CsrBt`scDbDeviceRecord` that is to be written.

The function will be called when:

- New link keys have been calculated, either because the local or remote device requested it, or in order to compare it to an already existing link-key (triggered by low-level HCI stack)
- The device services etc. needs to be updated (triggered by CSR_BT_SC_UPDATE_DEVICE_REQ)
- Trust-level changes (triggered by CSR_BT_SC_SET_TRUST_LEVEL_REQ)

3.4 CsrBtscDbRemove

This function is used for removing an entry from the database and has the following prototype:

```
void CsrBtscDbRemove(deviceAddr_t *deviceAddr);
```

where `deviceAddr` is a pointer to the Bluetooth address of the device that is to be removed from the database.

The function will be called when the HCI stack has deleted the device from its local cache.

3.5 CsrBtscDbReadFirst

This function is used for getting the first entry from the database and has the following prototype:

```
CsrBool CsrBtscDbReadFirst(CsrBtscDbDeviceRecord_t *theRecord);
```

where `theRecord` is a pointer to an already allocated memory block of at least `sizeof(CsrBtscDbDeviceRecord_t)`.

If the database is empty `FALSE` is returned else `TRUE` is returned.

This function is called when the SD or SC needs to read the first entry in the database, triggered by a CSR_BT_SC_READ_DEVICE_RECORD_REQ.

3.6 CsrBtscDbReadNext

The function is used for getting the next entry in the database and has the following prototype:

```
CsrBool CsrBtscDbReadNext(CsrBtscDbDeviceRecord_t *theRecord);
```

where `theRecord` is a pointer to an already allocated memory block of at least `sizeof(CsrBtscDbDeviceRecord_t)`.

The function should return TRUE when an entry is read and FALSE when no more entries are present.

This function is called when the SD or SC needs to read the next entry in the database, triggered by a CSR_BT_SC_READ_DEVICE_RECORD_REQ.

3.7 CSR Synergy Bluetooth Reference Implementation

CSR Synergy Bluetooth provides a number of reference implementations for the security database interface, namely

- A in-memory implementation
- A *filesystem* implementation.

The in-memory implementation (*simple_mem)* provides the most basic, non-persistent implementation possible. It is able to hold a single entry in memory, so only the link-key to the most recently paired device is remembered. If the application is closed and reopened, the database will be empty. The code can be found under *example_drivers/csr_bt_sc_db/simple_mem/*.

The other implementation is a persistent file-based database. The implementation is cross-platform as CSR Synergy Bluetooth provides wrappers for a Posix-based filesystem API (fclose, fopen, etc.). The ScDb implementation then uses this API to provide the database interface. There are no built-in limitations of the file-based implementation, so it will (in theory) be possible to store an unlimited number of link-keys in the database. As the database is file-based, the link-keys will not be lost when an application is restarted.

4 SCO Audio Interface

SCO data can either be routed over the PCM port or over the HCI interface, i.e. either BSCP (UART), H4DS (UART) or USB.

The normal way is to run the SCO audio on the PCM port but it is possible to route it over the HCI interface also.

4.1.1 Audio over PCM

In order to route SCO data over the PCM port the PSKey: `MAP_SCO_PCM` shall be set to TRUE.

4.1.2 Audio over BCSP (UART), H4DS (UART) or USB

When `MAP_SCO_PCM` is set to `FALSE`, it is possible to set-up whether SCO is routed over the HCI interface or is mapped to PCM slot 1,2 or 3. The control of this is handled by the application through an interface in the individual profiles. For a description of this please see the audio documentation of the different profiles that makes audio possible.

If SCO is to be routed over the HCI interface a SCO service function must be registered when the application receives an audio confirm/indication signal with result code CSR_BT_SUCCESS and the SCO handle included.

The SCO service is registered by calling:

```
CsrBool CsrRegisterScoHandle(CsrUin16 theScoHandle, ScoHandlerFuncType
theFunctionPtr)
```

which returns TRUE if the call is successful. The `theScoHandle` is the identity of the SCO handle, which is returned as a parameter in the audio confirm/indication signal. The `theFunctionPtr` must be the name of the function, which the received HCI SCO data packets are sent to. This function must be defined as illustrated below:

```
void theFunction(char * theData)
```

where `theData` is the HCI SCO data packet, see Figure 3. For more information about HCI SCO data packet refer to [BT] Part E section 5.4.3.

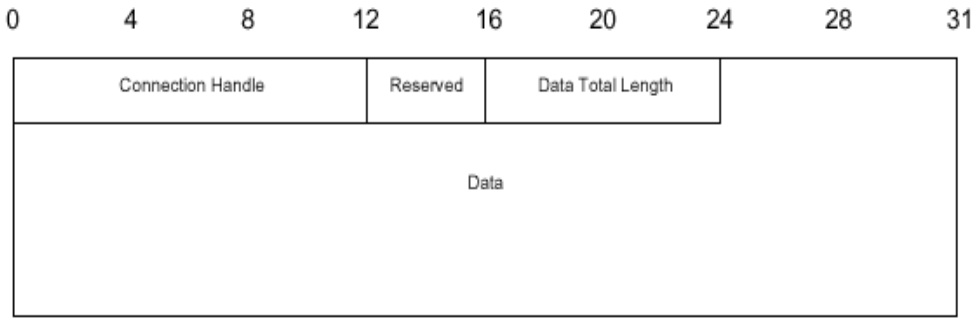


Figure 3: HCI SCO data packet

If the SCO service is registered with success, SCO data can be sent to the peer device by calling the function:

```
void CsrSendScoData(char * theData)
```

The parameter `theData` must be a HCI SCO data packet, where the *Connection Handle* again is the identity of the SCO handle. *Data Total Length* is the length of SCO data giving in bytes, and *Data* is the SCO data. The SCO data format must be the same format as the incoming HCI SCO data packet.

The functions `CsrRegisterScoHandle` and Csr`SendScoData` are defined in *csr_sco_audio.h*.

In csr_bt_usr_config.h it is possible to set the voice parameter, which controls all the various settings for SCO connections. These settings apply to all SCO connections, and cannot be set individually. Note that the input part of the voice parameter is normally overridden by PSKEY_PCM_FORMAT, so only the air coding is used. This behaviour can be disabled by setting the PS key to 0xFFFF.

NOTE: when audio is transferred as a 16 bit data stream, CSR recommends a minimum of 460.8 kb/s. For 2 or 3 simultaneous SCO links, CSR recommends a minimum of 921.6 kb/s. For 8 bit data streams, the baud rate requirements can be halved.

5 Document References

Document	Reference
Bluetooth® Core Specification, Version no. 2.0, 2.1 and 3.0	[BT]

All documents are not part of the CSR Synergy Bluetooth delivery, but can be found on the CSR website: www.csr.com.

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

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.