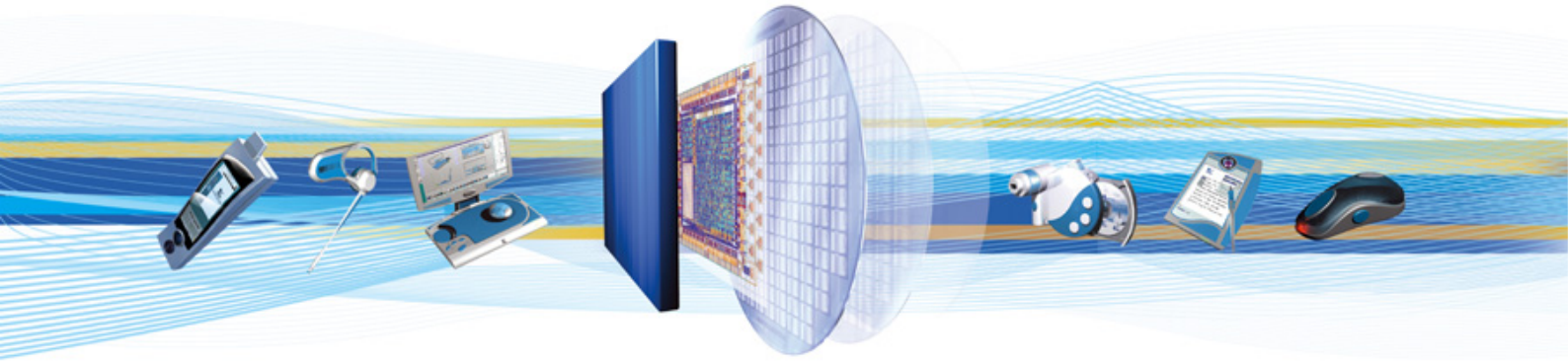
CSR Synergy Framework 3.1.0

Log

API Description

August 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 Tables

1 Introduction

1.1 Introduction and Scope

This document describes the logging services provided in CSR Synergy, how to use them, and how to use custom log formatting plugins e.g. to interface with platform specific interfaces.

The purpose of this document is to make the reader able to use the logging services as part of their application development.

2 Description

2.1 Introduction

The primary way of performing debugging and optimisation of CSR Synergy is through the CSR Synergy logging services. The logging services provide a generic way to log certain events and conditions that occur during the execution of the CSR Synergy host software to aid in debugging, integration, and optimisation. It is a generic framework that be used with the standard CSR Synergy log formatters as well as custom. The resulting log data can be written to any kind of media.

Events that can be logged can roughly be categorized as follows:

- Task events
- Environment events
- Generic text logging

Task events are events that happen in the context of a given task, i.e. that the event happens during the execution of or at the request of a particular task. Examples of such events are that the task is being initialized, sends a message to another task, schedules a timer, or that a timer triggers.

Environment events are events that happen in the context of the system as a whole or at least not in the context of a particular task. Examples of environment events are the registration of a background interrupt, request to schedule a background interrupt, or when data is sent or received when communicating with a chip, e.g. a CSR BlueCore™.

Generic text logging is ``printf()-style'' logging with severity levels (debug, info, warning, error, critical).

2.2 Concepts and terminology

Before proceeding some basic terminology needs to be established:

- Log formatter
- Log transport
- Log instance

A log formatter is a component that receives events and translates them into a well-defined byte stream that can be parsed later.

A log transport is a component that receives formatted log data and writes it to the desired medium. Examples of media are UART (to move log data off a target e.g. for live analysis or external storage), disk, flash, RAM-disk, display, etc. The log transport interface is described in [LOGTRANSPORT]. A log transport is of type `void *`.

A log instance is an instance of a logging service translating events to a particular log format. A log instance must be registered in the logging framework to be active (i.e. be notified of events). A log instance is of type `CsrLog *`.

2.3 Log formatters

The CSR Synergy Framework provides four generic log formatters:

- PCAP logging
- BTsnoop logging
- Frontline FTS live logging
- Cleartext logging

All of these are based on the generic log transport interface described in [LOGTRANSPORT].

2.3.1 PCAP logging

PCAP logging provides logging of all scheduler events and in general works very well for task communication debugging. PCAP logging can be read with WireShark using the dissectors provided by CSR Synergy.

2.3.2 BTsnoop logging

BTsnoop logging performs logging of Bluetooth HCI data packets it BTsnoop format where it can be read e.g. by WireShark and Frontline FTS.

2.3.3 FTS live logging

FTS live logging performs logging of Bluetooth HCI communication to be used with live traffic analysis in Frontline FTS.

2.3.4 Cleartext logging

Cleartext logging provides translation from the generic text logging and enables the log data to be transported over the generic log transport interface. The cleartext log formatter is intended to be used with stream-based interfaces such as a terminal connected to a UART or a display and other interfaces (e.g. simple text log files) where the log data is read in its raw form by humans.

2.4 Log transports

The log transport interface is documented elsewhere but this section contains some general information.

First of all, it is worth noting that while the log transport interface is a generic interface, certain log formatters make little sense without being used with the proper type of log transport. One example of this is the FTS live logging in which log data must be delivered to FTS over a special pipe. It is the responsibility of the user to ensure that a given log formatter is correctly paired with an appropriate log transport.

A particular instance of a log transport is identified by a void pointer used as a handle. The logging services do not use this handle for anything other than identifying to the generic log transport interface which *log transport* instance a particular *log formatter* instance wishes to pass its log data to. This also means that it is possible for multiple log formatting instances to share a given log transport if desired.

Log formatters are also not concerned with configuring and enabling log transports – this is the responsibility of the application. The CSR Synergy provided log formatters assume that the log transport is up and running at the time they are given a log transport handle. This also means that CSR Synergy doesn't define log transport initialization interfaces – these are considered both platform and transport type specific. This can be seen in the CSR Synergy Framework in that all initialization routines for the log transports are defined in the header file inc/platform/csr_logtransport_init.h.

It is very important to understand that a log formatter doesn't necessarily have to use the log transport interface. The log transport interface is simply an integration tool used for the CSR Synergy provided log formatters. A custom log formatter is free e.g. to write log data directly to files or display if desired, or to pass log data to a log framework provided by the platform OS.

3 Log Formatter Configuration

In this section, it is demonstrated how to set up the various log formatters provided as part of the CSR Synergy Framework.

3.1 Log instance control

3.1.1 CsrLogFormatInstRegister

Prototype

```
#include ”csr_log_formats.h”

CsrBool CsrLogFormatInstRegister(CsrLog *l);
```

Description

This function registers a log instance in the logging framework which causes it to be activated and events being passed to it.

Parameters

Type	Argument	Description
`CsrLog *`	`l`	Log instance to register

Table 1: Arguments to CsrLogFormatInstRegister

3.1.2 CsrLogUnregister

Prototype

```
#include ”csr_log_formats.h”

void CsrLogUnregister(CsrLog *l);
```

Description

This function unregisters a log instance in the logging framework which causes it to be deactivated and events no longer being passed to it.

Parameters

Type	Argument	Description
`CsrLog *`	`l`	Log instance to unregister

Table 2: Arguments to CsrLogUnregister

3.2 BTsnoop logging

3.2.1 CsrLogBtsnoopCreate

Prototype

```
#include ”csr_log_btsnoop.h”

CsrLog * CsrLogBtsnoopCreate (void *ltHdl);
```

Description

This function creates a BTsnoop logging instance that uses the log transport identified by the log transport handle given as a parameter.

Parameters

Type	Argument	Description
`void *`	`ltHdl`	Log transport handle

Table 3: Arguments to CsrLogBtsnoopCreate

Example

The following example sets up a log transport handle for a file and creates a PCAP log formatter instance that uses it.

```
void *ltBtsnoopFile;


ltBtsnoopFile = CsrLogTransportFileOpen("btsnoop.log");

if (ltBtsnoopFile)

{

    CsrLog *logBtsnoopFile;

    logBtsnoopFile = CsrLogBtsnoopCreate(ltBtsnoopFile);

    CsrLogFormatInstRegister(logBtsnoopFile);

}
```

The `CsrLogTransportFileOpen()` call can be replaced by a call to any function that sets up a log transport.

3.2.2 CsrLogBtsnoopDestroy

Prototype

```
#include ”csr_log_btsnoop.h”

void CsrLogBtsnoopDestroy(CsrLog *l);
```

Description

This function is used for de-allocating a BTsnoop logging instance when it is no longer needed.

Parameters

Type	Argument	Description
`CsrLog *`	`l`	BTsnoop log instance

Table 4: Arguments to CsrLogBtsnoopDestroy

3.3 Cleartext logging

3.3.1 CsrLogCleartextCreate

Prototype

```
#include ”csr_log_cleartext.h”
```

```
CsrLog *CsrLogCleartextCreate(void *ltHdl, const CsrCharString *format);
```

Description

This function creates a cleartext logging instance that uses the log transport identified by the log transport handle given as a parameter. The format is a character string that is parsed to determine what information is output. The output format can be selected by concatenating a string (can be done by the preprocessor) consisting of the `CSR_LOG_CLEARTEXT_TEMPLATE` symbols defined in `csr_log_cleartext.h`.

Parameters

Type	Argument	Description
`void *`	`ltHdl`	Log transport handle
`const CsrCharString *`	`format`	Output format string

Table 5: Arguments to CsrLogCleartextCreate

Example

The following example sets up a log transport handle for a file and creates a cleartext log formatter instance that uses it. The default output format (configured in `config/csr_usr_config.h`) is used.

```
void *ltCTFile;


ltCTFile = CsrLogTransportFileOpen("log.txt");

if (ltCTFile)

{

    CsrLog *logCTFile;


    logCTFile = CsrLogCleartextCreate(ltCTFile,
        CSR_LOG_CLEARTEXT_FORMAT);

    CsrLogFormatInstRegister(logCTFile);

}
```

The `CsrLogTransportFileOpen()` call can be replaced by a call to any function that sets up a log transport.

3.3.2 CsrLogCleartextDestroy

Prototype

```
#include ”csr_log_cleartext.h”

void CsrLogCleartextDestroy(CsrLog *l);
```

Description

This function is used for de-allocatiing a cleartext logging instance when it is no longer needed.

Parameters

Type	Argument	Description
CsrLog *	l	Cleartext log instance

Table 6: Arguments to CsrLogCleartextDestroy

3.4 FTS logging

3.4.1 CsrLogFtsCreate

Prototype

```
#include ”csr_log_fts.h”

CsrLog *CsrLogFtsCreate(void *ltHdl);
```

Description

This function creates an FTS logging instance that uses the log transport identified by the log transport handle given as a parameter.

Parameters

Type	Argument	Description
void *	ltHdl	Log transport handle

Table 7: Arguments to CsrLogFtsCreate

Example

The following example sets up a log transport handle for an FTS pipe and creates an FTS log formatter instance that uses it.

```
void *ltFtsPipe;


ltFtsPipe = CsrLogTransportFtsPipeOpen(“c:\\path\\to\\fts\\”);
if (ltFtsPipe)
{
    CsrLog *logFts;


    logFts = CsrLogFtsCreate(ltFtsPipe);
    CsrLogFormatInstRegister(logFts);
}
```

3.4.2 CsrLogFtsDestroy

Prototype

```
#include ”csr_log_fts.h”

void CsrLogFtsDestroy(CsrLog *l);
```

Description

This function is used for deallocatiing a FTS logging instance when it is no longer needed.

Parameters

Type	Argument	Description
`CsrLog *`	`l`	FTS log instance

Table 8: Arguments to CsrLogFtsDestroy

3.5 PCAP logging

3.5.1 CsrLogPcapCreate

Prototype

```
#include ”csr_log_pcap.h”

CsrLog *CsrLogPcapCreate(void *ltHdl);
```

Description

This function creates a PCAP logging instance that uses the log transport identified by the log transport handle given as a parameter.

Parameters

Type	Argument	Description
`void *`	`ltHdl`	Log transport handle

Table 9: Arguments to CsrLogPcapCreate

Example

The following example sets up a log transport handle for a file and creates a PCAP log formatter instance that uses it.

```
void *ltPcapFile;


ltPcapFile = CsrLogTransportFileOpen("pcap.cap");

if (ltPcapFile)

{

    CsrLog *logPcapFile;


    logPcapFile = CsrLogPcapCreate(ltPcapFile);

    CsrLogFormatInstRegister(logPcapFile);

}
```

The `CsrLogTransportFileOpen()` call can be replaced by a call to e.g. `CsrLogTransportWSPipeOpen(NULL, NULL)` that creates a pipe to WireShark for live log analysis, or it can be replaced by any other mechanism that sets up a log transport.

3.5.2 CsrLogPcapDestroy

Prototype

```
#include ”csr_log_pcap.h”

void CsrLogPcapDestroy(CsrLog *l);
```

Description

This function is used for deallocating a PCAP logging instance when it is no longer needed.

Parameters

Type	Argument	Description
`CsrLog *`	`l`	PCAP log instance

Table 10: Arguments to CsrLogPcapDestroy

4 Custom Log Formatters

This section describes how to implement custom log formatters that can be used with CSR Synergy.

4.1 The CsrLog Structure

A log formatter is defined by a `CsrLog` structure which is defined in `gsp/inc/csr_log_formats.h`. Fundamentally, this is just a struct of function pointers that are used for handling the various events supported by the log framework. The table below lists all the non-deprecated callbacks that may be used in new implementations.

Callback name	Event
`lregplatform`	Registers low level platform information
`lregtech`	Used for registering technologies
`lbci`	BlueCore™ Channel Interface
`ltrans`	BlueCore™ transport protocol
`ltextregister`	Text logging entity registration
`ltextprint`	Text logging
`ltextbufprint`	Text buffer print logging
`lsave`	A message has been pushed onto a message queue
`lpop`	A message has been popped from a message queue
`lactivate`	A task handler function starts
`ldeactivate`	A task handler function returns
`lputmsg`	A message has been put by a task
`lgetmsg`	A message has been received by a task
`ltimedeventin`	A timer has been started
`ltimedeventfire`	A timer has triggered and the timer callback starts
`ltimedeventdone`	A timer callback returns
`ltimedeventcancel`	A timer has been cancelled
`bgintreg`	A background interrupt has been registered
`bgintunreg`	A background interrupt has been unregistered
`bgintset`	A background interrupt has been scheduled
`bgintservicestart`	A background interrupt handler function has started
`bgintservicedone`	A background interrupt handler function has returned

For a description on the API for these callbacks, please consult the `gsp/inc/csr_log_formats.h` header file.

5 Document References

Ref	Title
[LOGTRANSPORT]	Log Transport API: api-0008-log_transport

Terms and Definitions

API	Application Programming Interface
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
MSC	Message Sequence Chart
SW	Software

Document History

Revision	Date	History
1	2010-08-19	1st draft.
2	OCT 2010	Ready for release 2.2.0
3	DEC 2010	Ready for release 3.0.0
4	Aug 2011	Ready for release 3.1.0

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.

No statements or representations in this document are to be construed as advertising, marketing, or offering for sale in the United States imported covered products subject to the Cease and Desist Order issued by the U.S. International Trade Commission in its Investigation No. 337-TA-602. Such products include SiRFstarIII™ chips that operate with SiRF software that supports SiRFInstantFix™, and/or SiRFLoc® servers, or contains SyncFreeNav functionality.

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.