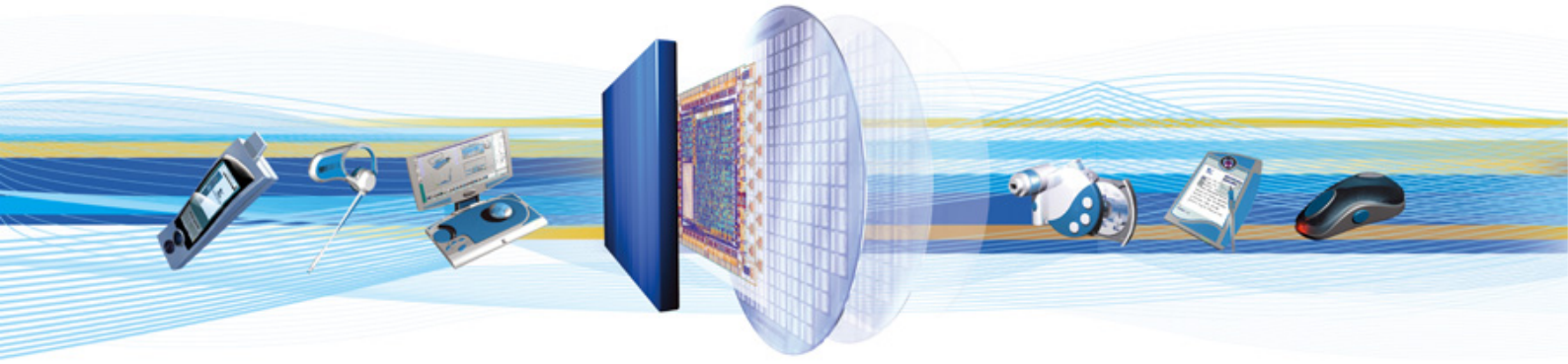
CSR Synergy Framework 3.1.0

Utilities

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 utility functions that a BSP must provide in order to be compliant with the CSR Synergy Framework specification. These are collectively referred to as the Synergy Utilities API and must be exposed to both the CSR Synergy Framework itself as well as any Synergy technologies that are used with the framework.

The function prototypes which are required for complying with the requirements and functionality described in this document are found in `csr_util.h`.

2 Utilities API

2.1 Introduction

The Synergy Utilities API is a collection of frequently used helper function that are used throughout Synergy.

The CSR Synergy Framework provides implementations of this API with a set of BSPs -- one for Windows-x86, one for Linux-x86 and one for Nucleus-ARM. These BSPs implement fully functioning and compliant implementations of the complete CSR Synergy Framework Utilities API. The implementations on these platforms may serve as a good starting point for other implementations for similar platforms. However, for every implementation it should be considered that the functions API are widely and frequently used, and thus it is important that implementations are efficient.

2.2 API functionality

The functionality provided by the Synergy Utilities API can generally be divided into the following categories:

- Arithmetic number conversion
- Memory manipulation
- Character string manipulation
- Random number generation
- Miscellaneous

In the following, the individual functions are presented with an overview of their prototype, a description of their arguments and return type, as well as an explanation of the required functionality and semantics.

2.3 Arithmetic number conversion

2.3.1 CsrStrToInt

Prototype

```
#include "csr_util.h"

CsrUint32 CsrStrToInt(const CsrCharString *s);
```

Description

This function is used for converting an ASCIIZ string containing an 32-bit unsigned integer in base 10 on the form of multiple of the ASCII characters “0” to “9”.

Parameters

Type	Argument	Description
`const CsrCharString *`	`s`	A pointer to the ASCIIZ encoded string that contains the integer value. May not be `NULL`.

Table 1: Arguments to CsrStrToInt

Returns

The converted 32-bit unsigned integer.

2.3.2 CsrHexStrToUint8

Prototype

```
#include "csr_util.h"

CsrBool CsrHexStrToUint8(const CsrCharString *string,
    CsrUint8 *returnValue);
```

Description

This function is used for converting an ASCIIZ string containing an 8-bit unsigned integer in base 16 on the standard hexadecimal form "0x##" or "##" where "#" denotes any of the ASCII characters "0" to "9" and "a" to "f" in lower or upper case.

Upon successful conversion, the 8-bit value pointed to by `returnValue` is changed to the value of the string. If a run-time error occurs during conversion (such as encountering illegal characters), the contents of the value pointed at by `returnValue` are undefined.

Parameters

Type	Argument	Description
`const CsrCharString *`	`string`	A pointer to the hexadecimal ASCIIZ encoded string that contains the integer value. May not be `NULL`.
`CsrUint8 *`	`returnValue`	A pointer to the 8-bit value in which the converted value will be stored. May not be `NULL`.

Table 2: Arguments to CsrHexStrToUint8

Returns

TRUE, if the conversion completes without error, otherwise FALSE.

2.3.3 CsrHexStrToUint16

Prototype

```
#include "csr_util.h"

CsrBool CsrHexStrToUint16(const CsrCharString *string,
    CsrUint16 *returnValue);
```

Description

This function is used for converting an ASCIIZ string containing an 16-bit unsigned integer in base 16 on the standard hexadecimal form "0x####" or "####" where "#" denotes any of the ASCII characters "0" to "9" and "a" to "f" in lower or upper case.

Upon successful conversion, the 16-bit value pointed to by `returnValue` is changed to the value of the string. If a run-time error occurs during conversion (such as encountering illegal characters), the contents of the value pointed at by `returnValue` are undefined.

Parameters

Type	Argument	Description
`const CsrCharString *`	`string`	A pointer to the hexadecimal ASCIIZ encoded string that contains the integer value. May not be `NULL`.
`CsrUint16 *`	`returnValue`	A pointer to the 16-bit value in which the converted value will be stored. May not be `NULL`.

Table 3: Arguments to CsrHexStrToUint16

Returns

TRUE, if the conversion completes without error, otherwise FALSE.

2.3.4 CsrHexStrToUint32

Prototype

```
#include "csr_util.h"

CsrBool CsrHexStrToUint32(const CsrCharString *string,
    CsrUint32 *returnValue);
```

Description

This function is used for converting an ASCIIZ string containing an 32-bit unsigned integer in base 16 on the standard hexadecimal form "0x########" or "########" where "#" denotes any of the ASCII characters "0" to "9" and "a" to "f" in lower or upper case.

Upon successful conversion, the 32-bit value pointed to by `returnValue` is changed to the value of the string. If a run-time error occurs during conversion (such as encountering illegal characters), the contents of the value pointed at by `returnValue` are undefined.

Parameters

Type	Argument	Description
`const CsrCharString *`	`string`	A pointer to the hexadecimal ASCIIZ encoded string that contains the integer value. May not be `NULL`.
`CsrUint32 *`	`returnValue`	A pointer to the 32-bit value in which the converted value will be stored. May not be `NULL`.

Table 4: Arguments to CsrHexStrToUint32

Returns

TRUE, if the conversion completes without error, otherwise FALSE.

2.3.5 CsrIntToBase10

Prototype

```
#include "csr_util.h"

void CsrIntToBase10(CsrInt32 number, CsrCharString *str);
```

Description

This function is used for converting a signed 32-bit integer to an ASCIIZ representation of the same value in base 10.

Parameters

Type	Argument	Description
`CsrInt32`	`number`	The 32-bit integer value to convert.
`CsrCharString *`	`str`	A character array large enough to hold a signed 32-bit integer in base 10 plus sign byte and zero termination.

Table 5: Arguments to CsrHexStrToUint32

Returns

Does not return a value.

2.3.6 CsrUInt16ToHex

Prototype

```
#include "csr_util.h"

void CsrUInt16ToHex(CsrUint16 number, CsrCharString *str);
```

Description

This function is used for converting an unsigned 16-bit integer to an ASCIIZ representation of the same value in base 16.

Parameters

Type	Argument	Description
CsrUint16	number	The 16-bit integer value to convert.
CsrCharString *	str	A character array at least five bytes long.

Table 6: Arguments to CsrUInt16ToHex

Returns

Does not return a value.

2.3.7 CsrUInt32ToHex

Prototype

```
#include "csr_util.h"

void CsrUInt32ToHex(CsrUint32 number, CsrCharString *str);
```

Description

This function is used for converting an unsigned 32-bit integer to an ASCIIZ representation of the same value in base 16.

Parameters

Type	Argument	Description
CsrUint32	number	The 32-bit integer value to convert.
CsrCharString *	str	A character array at least five bytes long.

Table 7: Arguments to CsrUInt32ToHex

Returns

Does not return a value.

2.3.8 CsrPow

Prototype

```
#include "csr_util.h"
```

```
CsrUint32 CsrPow(CsrUint32 base, CsrUint32 exponent);
```

Description

This function calculates the value of the parameter `base` to the power of the parameter `exponent`.

Parameters

Type	Argument	Description
`CsrUint32`	`base`	The base value.
`CsrUint32`	`exponent`	The exponent value.

Table 8: Arguments to CsrPow

Returns

The value of `base` to the exponent `exponent`.

2.4 Memory manipulation

2.4.1 CsrMemCpy

Prototype

```
#include "csr_util.h"

void *CsrMemCpy(void *dst, void *src, CsrSize len);
```

Description

The CsrMemCpy() function copies len bytes from buffer src to buffer dst, incrementing the addresses for every byte copied. The two memory regions may not overlap. If the source and destination memory regions overlap, the contents of the destination memory region will not be identical to the source memory region prior to the copy. To handle copies between overlapping memory regions, use CsrMemMove() instead.

Parameters

Type	Argument	Description
`void *`	`src`	The source address pointing to the data to be copied.
`void *`	`dst`	The starting destination address where the data are copied to.
`CsrSize`	`len`	The number of bytes to copy.

Table 9: Arguments to CsrMemCpy

Returns

The original value of `dst`.

2.4.2 CsrMemMove

Prototype

```
#include "csr_util.h"

void *CsrMemMove(void *dst, void *src, CsrSize len);
```

Description

The CsrMemMove() function copies `len` bytes from buffer `src` to buffer `dst`, incrementing the addresses for every byte copied. The two memory regions may overlap – the copy is done in a non-destructive manner.

Parameters

Type	Argument	Description
`void *`	`src`	The source address pointing to the data to be copied.
`void *`	`dst`	The starting destination address where the data are copied to.
`CsrSize`	`len`	The number of bytes to copy.

Table 10: Arguments to CsrMemMove

Returns

The original value of `dst`.

2.4.3 CsrMemSet

Prototype

```
#include "csr_util.h"

void *CsrMemSet(void *dst, int ch, CsrSize len);
```

Description

The CsrMemSet() function sets the contents of the memory region `len` bytes long starting at `dst` to `ch` converted to an unsigned char.

Parameters

Type	Argument	Description
`void *`	`dst`	The source address pointing to the data to be copied.
`int`	`ch`	The value (converted to an unsigned char) that the memory region is set to.
`CsrSize`	`len`	The number of bytes to copy.

Table 11: Arguments to CsrMemSet

Returns

The original value of `dst`.

2.4.4 CsrMemCmp

Prototype

```
#include "csr_util.h"

int CsrMemCmp(void *a, void *b, CsrSize len);
```

Description

The CsrMemCmp() function compares the `len` bytes long memory regions `a` and `b` and determine if their contents are identical.

Parameters

Type	Argument	Description
void *	a	The first address of the first memory region.
void *	b	The first address of the second memory region.
CsrSize	len	The length of the two memory regions.

Table 12: Arguments to CsrMemCmp

Returns

If the two memory regions have identical contents or are of zero length, the function returns 0. Otherwise, the difference of the first two bytes, treated as unsigned char values, is returned.

2.4.5 CsrMemDup

Prototype

```
#include "csr_util.h"

void *CsrMemDup(const void *src, CsrSize count);
```

Description

The CsrMemDup() function duplicates a memory region `count` bytes long starting at `src` into an equally long memory region that shall be deallocated with CsrPfree() when no longer needed

Parameters

Type	Argument	Description
void *	src	The first address of the first memory region.
CsrSize	count	The length of the memory region.

Table 13: Arguments to CsrMemDup

Returns

A pointer to a memory region with contents identical to `src`. When no longer needed, this pointer must be deallocated with CsrPfree() to avoid memory leaks.

2.5 Character String Manipulation

2.5.1 CsrStrLen

Prototype

```
#include "csr_util.h"

CsrSize CsrStrLen(const CsrCharString *s);
```

Description

The CsrStrLen() function determines the number of characters in an ASCIIZ string excluding the zero termination.

Parameters

Type	Argument	Description
const CsrCharString *	s	A pointer to the string to obtain the length of.

Table 14: Arguments to CsrStrLen

Returns

The number of characters leading up to the terminating zero.

2.5.2 CsrStrCmp

Prototype

```
#include "csr_util.h"

int CsrStrCmp(const CsrCharString *s1, const CsrCharString *s2);
```

Description

The CsrStrCmp() function compares two ASCIIZ strings lexicographically.

Parameters

Type	Argument	Description
`const CsrCharString *`	`s1`	A pointer to the first string to compare.
`const CsrCharString *`	`s2`	A pointer to the second string to compare.

Table 15: Arguments to CsrStrCmp

Returns

If the two strings are of equal length (possibly zero) and have identical contents, the return value is 0. Otherwise the return value is positive or negative, depending on whether `s1` or `s2` is larger than the other. The comparison is done for each byte as unsigned characters.

2.5.3 CsrStrNCmp

Prototype

```
#include "csr_util.h"

int CsrStrNCmp(const CsrCharString *s1,
    const CsrCharString *s2,
    CsrSize len);
```

Description

The CsrStrNCmp() function compares at most `len` characters from two ASCIIZ strings lexicographically.

Parameters

Type	Argument	Description
`const CsrCharString *`	`s1`	A pointer to the first string to compare.
`const CsrCharString *`	`s2`	A pointer to the second string to compare.
`CsrSize`	`len`	The maximum number of characters to compare.

Table 16: Arguments to CsrStrNCmp

Returns

If the two strings are of equal length (possibly zero) and up to the first len characters are identical, the return value is 0. Otherwise the return value is positive or negative, depending on whether `s1` or `s2` is larger than the other. The comparison is done for each byte as unsigned characters.

2.5.4 CsrStrNICmp

Prototype

```
#include "csr_util.h"

int CsrStrNICmp(const CsrCharString *s1,
    const CsrCharString *s2,
    CsrSize len);
```

Description

The CsrStrNICmp() function compares at most `len` characters from two ASCIIZ strings lexicographically ignoring whether letters are upper or lower case.

Parameters

Type	Argument	Description
`const CsrCharString *`	`s1`	A pointer to the first string to compare.
`const CsrCharString *`	`s2`	A pointer to the second string to compare.
`CsrSize`	`len`	The maximum number of characters to compare.

Table 17: Arguments to CsrStrNICmp

Returns

If the two strings are of equal length (possibly zero) and up to the first len characters are identical, the return value is 0. Otherwise the return value is positive or negative, depending on whether `s1` or `s2` is larger than the other. The comparison is done for each byte as unsigned characters.

2.5.5 CsrStrChr

Prototype

```
#include "csr_util.h"

CsrCharString *CsrStrChr(const CsrCharString *s, int ch);
```

Description

The CsrStrChr() function locates the first occurrence of the character `ch` in the ASCIIZ string `s`.

Parameters

Type	Argument	Description
`const CsrCharString *`	`s`	A pointer to the first string to compare.
`CsrCharString`	`ch`	The character to search for.

Table 18: Arguments to CsrStrChr

Returns

If `ch` is not found, the return value is `NULL`, otherwise a pointer to the character is returned.

2.5.6 CsrStrStr

Prototype

```
#include "csr_util.h"

CsrCharString *CsrStrStr(const CsrCharString *s,
    const CsrCharString *ss);
```

Description

The CsrStrStr() function locates the first occurrence of the ASCIIZ substring `ss` within the ASCIIZ string `s`.

Parameters

Type	Argument	Description
`const CsrCharString *`	`src`	A pointer to the string to be copied.
`CsrCharString *`	`dst`	A pointer to the character array to copy to the string to.

Table 19: Arguments to CsrStrStr

Returns

If `ss` is the empty string, `s` is returned. If `ss` is found nowhere within `s`, `s` is returned. Otherwise, a pointer to the beginning of the first occurrence of `ss` within `s` is returned.

2.5.7 CsrStrNCpy

Prototype

```
#include "csr_util.h"

CsrCharString *CsrStrNCpy(CsrCharString *dst,
    const CsrCharString *src,
    CsrSize len);
```

Description

The CsrStrNCpy() function copies at most `len` characters from the ASCIIZ string `src` to the character array `dst`. If the length of `src` is longer than or equal to `len`, `dst` is **not** zero-terminated.

Note: This function does not guarantee zero-termination of the destination character array.

Parameters

Type	Argument	Description
`const CsrCharString *`	`src`	A pointer to the string to be copied.
`CsrCharString *`	`dst`	A pointer to the character array to copy to the string to.
`CsrSize`	`len`	The maximum number of characters to copy.

Table 20: Arguments to CsrNCpy

Returns

The original value of `dst`.

2.5.8 CsrStrNCpyZero

Prototype

```
#include "csr_util.h"

CsrCharString *CsrStrNCpyZero(CsrCharString *dst,
    const CsrCharString *src,
    CsrSize count);
```

Description

The CsrStrNCpyZero() function copies at most `len` characters from the ASCIIZ string `src` to the character array `dst` and zero terminates the `dst` array.

Parameters

Type	Argument	Description
`const CsrCharString *`	`src`	A pointer to the string to be copied.
`CsrCharString *`	`dst`	A pointer to the character array to copy to the string to.
`CsrSize`	`len`	The maximum number of characters to copy.

Table 21: Arguments to CsrNCpyZero

Returns

The original value of `dst`.

2.5.9 CsrStrCat

Prototype

```
#include "csr_util.h"

CsrCharString *CsrStrCat(CsrCharString *dst, const CsrCharString *append);
```

Description

The CsrStrCat() function appends the ASCIIZ string from `append` to the character array `dst`. It is up to the caller to ensure that `dst` is large enough to hold the contents of `append` including zero termination.

Parameters

Type	Argument	Description
`const CsrCharString *`	`append`	A pointer to the string to be copied.
`CsrCharString *`	`dst`	A pointer to the character array to copy to the string to.

Table 22: Arguments to CsrStrCat

Returns

The original value of `dst`.

2.5.10 CsrStrNCat

Prototype

```
#include "csr_util.h"

CsrCharString *CsrStrNCat(CsrCharString *dst,
    const CsrCharString *append,
    CsrSize len);
```

Description

The CsrStrNCat() function appends at most `len` characters from the ASCIIZ string `append` to the character array `dst`. If the length of `src` is longer than or equal to `len`, `dst` is **not** zero-terminated. It is up to the caller to ensure that `dst` is large enough to hold `append`.

Note: This function does not guarantee zero-termination of the destination character array.

Parameters

Type	Argument	Description
`const CsrCharString *`	`src`	A pointer to the string to be copied.
`CsrCharString *`	`dst`	A pointer to the character array to copy to the string to.
`CsrSize`	`len`	The maximum number of characters to copy.

Table 23: Arguments to CsrNCat

Returns

The original value of `dst`.

2.5.11 CsrStrDup

Prototype

```
#include "csr_util.h"

CsrCharString *CsrStrDup(const CsrCharString *src);
```

Description

The CsrStrDup() function provides a copy of the supplied ASCIIZ string `src`.

Parameters

Type	Argument	Description
`const CsrCharString *`	`src`	A pointer to the string to be copied.

Table 24: Arguments to CsrStrDup

Returns

A pointer to an ASCIIZ string duplicate of src. When the contents are no longer needed, this pointer shall be deallocated using CsrPfree() to avoid memory leaks.

2.5.12 CsrVsnprintf

Prototype

```
#include "csr_util.h"

int CsrVsnprintf(CsrCharString *dst, CsrSize count, const CsrCharString *format,
va_list argptr);
```

Description

The CsrVsnprintf() function provides formatted output conversion based on the supplied ASCIIZ string `format` and the optional list of parameters and stores the resulting ASCIIZ string in the `dst` character array.

NOTE: It is assumed that the implementation of this function is thread safe.

Parameters

Type	Argument	Description
`CsrCharString *`	`dst`	The destination character array for the converted output.
`CsrSize`	`count`	The maximum number of characters to write
`const CsrCharString *`	`format`	An ASCIIZ string describing the output and format conversion of any optional arguments. The format string is a standard C printf() format string as defined in the ANSI C standard. But it is guaranteed that the caller of this function, does not use floating point arguments like %f in this format string.
`Va_list`	`argptr`	Pointer to the list of arguments.

Table 25: Arguments to CsrVsnprintf

Returns

If an output or conversion error occurs, -1 is returned, otherwise the number of characters (excluding the zero termination byte) output is printed.

2.6 Random number generation

2.6.1 CsrRandom

Prototype

```
#include "csr_util.h"

CsrUint32 CsrRandom(void);
```

Description

The CsrRandom() function is used for obtaining a pseudo-random number.

NOTE: It is assumed that the platform has seeded the random number generator prior a call to the CsrRandom() function in case this is needed on the platform. Also, it is assumed that the function implementation is made thread safe, but it is guaranteed that the function is not called from interrupt context.

Parameters

None.

Returns

A 32 bit unsigned pseudo-random number.

2.7 Miscellaneous

2.7.1 CsrBitCountSparse

Prototype

```
#include "csr_util.h"

CsrUint8 CsrBitCountSparse(CsrUint32 n);
```

Description

The CsrBitCountSparse() function counts the number of bits that are set in the 32-bit unsigned integer `n`. This function is optimised for numbers where it expected that the majority of the bits are cleared, but it will return the correct number of bits set in any case.

Parameters

Type	Argument	Description
`CsrUint32`	`n`	An unsigned 32-bit integer expected to have fewer bits set than cleared.

Table 26: Arguments to CsrBitCountSparse

Returns

The number of bits set.

2.7.2 CsrBitCountDense

Prototype

```
#include "csr_util.h"

CsrUint8 CsrBitCountDense(CsrUint32 n);
```

Description

The CsrBitCountDense() function counts the number of bits that are set in the 32-bit unsigned integer `n`. This function is optimised for numbers where it expected that the majority of the bits are set, but it will return the correct number of bits set in any case.

Parameters

Type	Argument	Description
`CsrUint32`	`n`	An unsigned 32-bit integer expected to have fewer bits set than set.

Table 27: Arguments to CsrBitCountDense

Returns

The number of bits set.

2.7.3 CsrGetBaseName

Prototype

```
#include "csr_util.h"

const CsrCharString *CsrGetBaseName(const CsrCharString *file);
```

Description

The CsrGetBaseName() function strips away any directory path information of a relative or absolute path, returning a pointer to the actual file system object (e.g. file or directory) name that a file system path string is pointing to.

Parameters

Type	Argument	Description
`const CsrCharString *`	`file`	An ASCIIZ string containing a relative or absolute directory path to a file system object of any type (e.g. file, directory, socket, etc.).

Table 28: Arguments to CoalSparseBitCount

Returns

A pointer offset into the provided directory path buffer at the offset where the name of the referenced object begins.

2.7.4 CsrIsSpace

Prototype

```
#include "csr_util.h"

CsrBool CsrIsSpace(CsrUint8 ch);
```

Description

The CoalIsSpace() function determines if the provided 8-bit character `ch` is a whitespace character.

Parameters

Type	Argument	Description
`CsrUint8`	`ch`	An 8-bit character that is checked whether it is a whitespace character.

Table 29: Arguments to CsrIsSpace

Returns

If the character `ch` is a whitespace character the function returns TRUE, otherwise it returns FALSE.

2.7.5 CsrOffsetOf

Prototype

```
#include "csr_util.h"

#define CsrOffsetOf(st, m)   ((CsrSize)&((st *)0)->m)
```

Description

The CsrOffsetOf() macro returns the offset of a struct member into a struct.

Parameters

Type	Argument	Description
`CsrUint8`	`ch`	An 8-bit character that is checked whether it is a whitespace character.

Table 30: Arguments to CsrIsSpace

Returns

The struct member offset in bytes.

3 Document References

Ref	Title

Terms and Definitions

[SYN-FRW-COAL-API]	CSR Synergy Framework COAL API. Doc. Api-0001-coal
BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
CSR	Cambridge Silicon Radio

Document History

Revision	Date	History
1	01 JUL 09	Ready for release 1.1.0
2	30 NOV 09	Ready for release 2.0.0
3	20 APR 10	Ready for release 2.1.0
4	OCT 10	Ready for release 2.2.0
5	DEC 10	Ready for release 3.0.0
6	Aug 11	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.