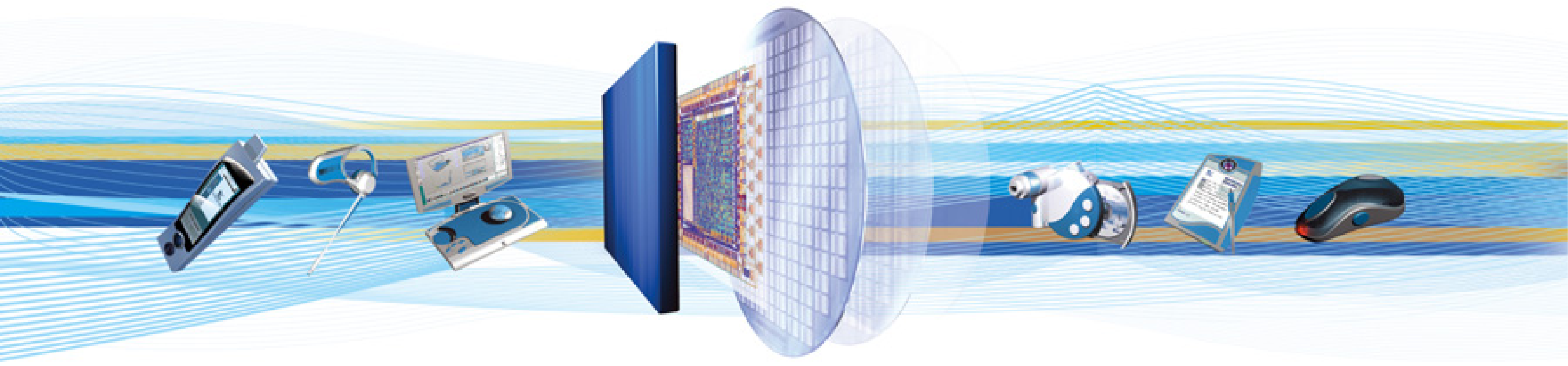
CSR Synergy®

CSR Synergy Framework 3.1.0

Low-Level File System

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 a low-level function based file system API which a BSP can choose to implement instead of porting the message based file system API (FSAL). The CSR Synergy Framework is delivered with a generic task implementation of the FSAL API, and this generic version is implemented on top of this low-level function based file system API. If a platform does not support an asynchronous file system API, it can choose to implement support for this low-level file system interface as opposed to the FSAL API.

The function prototypes which are required for complying with the requirements and functionality described in this document are split between two files the file handling functions are found in `csr_file.h` and the directory handling functions are found in `csr_dir.h`

2 File Handling Interface

2.1 CsrFileOpen

Prototype

```
#include "csr_file.h"

CsrResult CsrFileOpen(CsrFileHandle **handle, const CsrUtf8String *fileName,
CsrFileOpenFlags flags, CsrFilePerms perms);
```

Description

This function is used for opening a file.

Parameters

Type	Argument	Description
`CsrFileHandle **`	`handle`	A double pointer which the low-level file system can use to store a filehandle structure. This parameter will be used in all future operations on this file.
`Const CsrUtf8String *`	`fileName`	The path to the file which should be opened. Examples of valid file names are: "foo[.extention]": A file specified in this way should be opened relative to the current working directory. The [.extention] part is not mandatory. ./[directory1/][directory2/]foo[.extention]: A file specified in this way should be opened in the specified directory path but still relative to the current working directory. The [.extention] part is not mandatory. The number of [directory/] in the path can be any number but the low-level file system is allowed to impose a maximum length in bytes to any path (including the path of the current working directory). It is recommended not to impose this limit to be less than 255 bytes. /[directory1/][directory2/]foo[.extention]: A file specified in this way should be opened in the specified directory path but from the root of the filesystem. The [.extention] part is not mandatory. The number of [directory/] in the path can be any number but the low-level file system is allowed to impose a maximum length in bytes to any path (including the path of the current working directory). It is recommended not to impose this limit to be less than 255 bytes.

`CsrFileOpenFlags`	`flags`	A bit pattern specifying the type of operations allowed on the file.
		
		Possible flags are:
		
		`CSR_FILE_OPEN_FLAGS_CREATE`
		
		`CSR_FILE_OPEN_FLAGS_READ_ONLY`
		
		`CSR_FILE_OPEN_FLAGS_WRITE_ONLY`
		
		`CSR_FILE_OPEN_FLAGS_READ_WRITE`
		
		`CSR_FILE_OPEN_FLAGS_APPEND`
		
		`CSR_FILE_OPEN_FLAGS_TRUNCATE`
		
		`CSR_FILE_OPEN_FLAGS_EXCL`

`CsrFilePerms`	`perms`	A bit pattern specifying the permissions which should apply for a new file. This parameter should only be evaluated in case the `CSR_FILE_OPEN_FLAGS_CREATE` is set in the `flags` parameter.
		
		Possible values are:
		
		`CSR_FILE_PERMS_USER_READ`
		
		`CSR_FILE_PERMS_USER_WRITE`
		
		`CSR_FILE_PERMS_USER_EXECUTE`
		
		`CSR_FILE_PERMS_GROUP_READ`
		
		`CSR_FILE_PERMS_GROUP_WRITE`
		
		`CSR_FILE_PERMS_GROUP_EXECUTE`
		
		`CSR_FILE_PERMS_OTHERS_READ`
		
		`CSR_FILE_PERMS_OTHERS_WRITE`
		
		`CSR_FILE_PERMS_OTHERS_EXECUTE`

Table 1: Arguments to CsrFileOpen

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

2.2 CsrFileClose

Prototype

```
#include "csr_file.h"

CsrResult CsrFileClose(CsrFileHandle *handle);
```

Description

This function is used for closing an open file handle.

Parameters

Type	Argument	Description
`CsrFileHandle *`	`handle`	The file handle to close.

Table 2: Arguments to CsrFileClose

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

2.3 CsrFileWrite

Prototype

```
#include "csr_file.h"

CsrResult CsrFileWrite(const void * buffer, CsrSize bytesToWrite, CsrFileHandle
*handle, CsrSize *written);
```

Description

This function is used for writing content of a buffer to an open file.

Parameters

Type	Argument	Description
`const void *`	`buffer`	A pointer to the data to be written in the file. May not be `NULL`.
`CsrSize`	`bytesToWrite`	The number of bytes to be written.
`CsrFileHandle *`	`handle`	The file handle to write to.
`CsrSize *`	`written`	The low-level file system must write the actual number of bytes written to the file in this parameter

Table 3: Arguments to CsrFileWrite

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

2.4 CsrFileRead

Prototype

```
#include "csr_file.h"

CsrResult CsrFileRead(void * buffer, CsrSize bytesToRead, CsrFileHandle *handle,
CsrSize *bytesRead);
```

Description

This function is used for reading data from an open file.

Parameters

Type	Argument	Description
`void *`	`buffer`	A pointer to the buffer in which the data should be read in to. May not be `NULL`.
`CsrSize`	`bytesToRead`	The maximum number of bytes to read from the file.
`CsrFileHandle *`	`handle`	The file handle to read from.
`CsrSize *`	`bytesRead`	The low-level file system must write the actual number of bytes read from the file in this parameter.

Table 4: Arguments to CsrFileRead

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

2.5 CsrFileSeek

Prototype

```
#include "csr_file.h"

CsrResult CsrFileSeek(CsrFileHandle *handle, CsrInt32 offset, CsrInt32
relativeOffset);
```

Description

This function is used for seeking to a given position in a file.

Parameters

Type	Argument	Description
`CsrFileHandle *`	`handle`	The file handle to seek in.
`CsrInt32`	`offset`	Specifies the number of bytes to move the file pointer relative to the origin specified by the `relativeOffset` parameter. This value is allowed to be negative.

`CsrInt32`	`relativeOffset`	The origin in the file from where offset of bytes should be counted.
		
		Possible values are:
		`CSR_SEEK_SET`
		`CSR_SEEK_CUR`
		`CSR_SEEK_END`

Table 5: Arguments to CsrFileSeek

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:
`CSR_FILE_RESULT_FAILURE`
`CSR_FILE_RESULT_EOF`
`CSR_FILE_RESULT_READ_ONLY`
`CSR_FILE_RESULT_NOT_EXIST`
`CSR_FILE_RESULT_NOT_ALLOWED`
`CSR_FILE_RESULT_ALREAD_EXISTS`
`CSR_FILE_RESULT_NO_SPACE`

2.6 CsrFileFlush

Prototype

```
#include "csr_file.h"

CsrResult CsrFileFlush(CsrFileHandle *handle);
```

Description

This function is used for flushing a file directly to the disk.

Parameters

Type	Argument	Description
`CsrFileHandle *`	`handle`	The file handle to flush.

Table 6: Arguments to CsrFileFlush

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:
`CSR_FILE_RESULT_FAILURE`
`CSR_FILE_RESULT_EOF`
`CSR_FILE_RESULT_READ_ONLY`
`CSR_FILE_RESULT_NOT_EXIST`
`CSR_FILE_RESULT_NOT_ALLOWED`
`CSR_FILE_RESULT_ALREAD_EXISTS`
`CSR_FILE_RESULT_NO_SPACE`

2.7 CsrFileTell

Prototype

```
#include "csr_file.h"

CsrResult CsrFileTell(CsrFileHandle *handle, CsrUint32 *position);
```

Description

This function is used for telling the current position of the file pointer relative to the beginning of a file.

Parameters

Type	Argument	Description
`CsrFileHandle *`	`handle`	The file handle.
`CsrUint32 *`	`position`	The low-level file system must write the current position of the file pointer to this pointer.

Table 7: Arguments to CsrFileTell

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

2.8 CsrFileRemove

Prototype

```
#include "csr_file.h"

CsrResult CsrFileRemove(const CsrUtf8String * filename);
```

Description

This function is used for removing a file from the file system.

Parameters

Type	Argument	Description
`Const CsrUtf8String *`	`fileName`	The path to the file that should be removed. The same clauses applies to this parameter as described for the `fileName` parameter in 2.1.

Table 8: Arguments to CsrFileRemove

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
```

```
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

2.9 CsrFileSetEndOfFile

Prototype

```
#include "csr_file.h"

CsrResult CsrFileSetEndOfFile(CsrFileHandle *handle);
```

Description

This function is used for marking the current position of the file pointer as the end of the file.

Parameters

Type	Argument	Description
`CsrFileHandle *`	`handle`	The file handle.

Table 9: Arguments to CsrSetEndOfFile

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

2.10 CsrFileSetPerms

Prototype

```
#include "csr_file.h"

CsrResult CsrFileSetPerms(const CsrUtf8String * name, CsrFilePerms perms);
```

Description

This function is used for setting the permissions for a file or a directory.

Parameters

Type	Argument	Description
`Const CsrUtf8String *`	`name`	The path to the file or directory which permissions must be changed. The same clauses applies to this parameter as described for the `fileName` parameter in 2.1.

CsrFilePerms	perms	A bit pattern specifying the permissions which should apply for the file or directory specified by `name`. Possible values are: CSR_FILE_PERMS_USER_READ CSR_FILE_PERMS_USER_WRITE CSR_FILE_PERMS_USER_EXECUTE CSR_FILE_PERMS_GROUP_READ CSR_FILE_PERMS_GROUP_WRITE CSR_FILE_PERMS_GROUP_EXECUTE CSR_FILE_PERMS_OTHERS_READ CSR_FILE_PERMS_OTHERS_WRITE CSR_FILE_PERMS_OTHERS_EXECUTE

Table 10: Arguments to CsrFileSetPerns

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

2.11 CsrFileRename

Prototype

```
#include "csr_file.h"

CsrResult CsrFileRename(const CsrUtf8String *oldName, const CsrUtf8String
*newName);
```

Description

This function is used for renaming a file or directory.

Parameters

Type	Argument	Description
const CsrUtf8String *	oldName	The old path to the file or directory which must be renamed. The same clauses applies to this parameter as described for the `fileName` parameter in 2.1.
const CsrUtf8String *	newName	The new path to the file or directory. The same clauses applies to this parameter as described for the `fileName` parameter in 2.1.

Table 11: Arguments to CsrFileRename

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_FILE_RESULT_FAILURE
CSR_FILE_RESULT_EOF
CSR_FILE_RESULT_READ_ONLY
CSR_FILE_RESULT_NOT_EXIST
CSR_FILE_RESULT_NOT_ALLOWED
CSR_FILE_RESULT_ALREAD_EXISTS
CSR_FILE_RESULT_NO_SPACE
```

3 Directory Handling Interface

3.1 CsrDirGetCurrentWorkingDir

Prototype

```
#include "csr_dir.h"

CsrResult CsrDirGetCurrentWorkingDir(CsrUtf8String **dirName);
```

Description

This function is used for obtaining the full path to the current working directory.

Parameters

Type	Argument	Description
`CsrUtf8String **`	`name`	The low-level file system must allocate a buffer large enough to contain the full path to the current working directory using CsrPmemAlloc(), and it must write the name, encoded as UTF8, of the current working directory to this buffer and assign the buffer to this double pointer. If the path contains path separator they must be written a '/' (forward slashes). It is the responsibility of the low-level file system to convert these forward slashes back a forth where ever appropriate.

Table 12: Arguments to CsrDirGetCurrentWorkingDir

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:
```
CSR_DIR_RESULT_FAILURE
CSR_DIR_RESULT_NOT_EXIST
CSR_DIR_RESULT_ALREADY_EXIST
CSR_DIR_RESULT_NOT_EMPTY
CSR_DIR_RESULT_INVALID_PATH
```

3.2 CsrDirStat

Prototype

```
#include "csr_dir.h"

CsrResult CsrDirStat(const CsrUtf8String * path, CsrDirEntryStat * fileStat);
```

Description

This function is used for obtaining status information regarding a file or directory.

Parameters

Type	Argument	Description
`const CsrUtf8String *`	`path`	The path to the file or directory. The same clauses applies to this parameter as described for the `fileName` parameter in 2.1.

`CsrDirEntryStat *`	`fileStat`	The status information of the specified file or directory.
		
		The information is filled into the following `CsrDirEntryStat` struct
		
		typedef struct
		`{`
		`    CsrSize        size;    /* 0 if not file */`
		`    CsrDirMode     mode;    /* mode */`
		`    CsrDirTm       time;    /* last modified */`
		`} CsrDirEntryStat;`
		
		Where the the time parameter is specified by the `CsrDirTm` struct below.
		
		`typedef struct`
		`{`
		`  CsrTime tm_sec;   /* Seconds: 0-59  */`
		`  CsrTime tm_min;   /* Minutes: 0-59  */`
		`  CsrTime tm_hour;  /* Hours since midnight: 0-23 */`
		`  CsrTime tm_mday;  /* Day of the month: 1-31 */`
		`  CsrTime tm_mon;   /* Months since january: 0-11 */`
		`  CsrTime tm_year;  /* Years since 1900 */`
		`  CsrTime tm_wday;  /* Days since Sunday (0-6) */`
		`  CsrTime tm_yday;  /* Days since Jan. 1: 0-365 */`
		`  CsrTime tm_isdst; /* +1 Daylight Savings Time, 0 No DST, 0xFFFF don't know    */`
		`  CsrBool utcTime;  /* TRUE=UTC, FALSE=local time */`
		`} CsrDirTm;`

Table 13: Arguments to CsrDirStat

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_DIR_RESULT_FAILURE
CSR_DIR_RESULT_NOT_EXIST
CSR_DIR_RESULT_ALREADY_EXIST
CSR_DIR_RESULT_NOT_EMPTY
CSR_DIR_RESULT_INVALID_PATH
```

3.3 CsrDirMake

Prototype

```
#include "csr_dir.h"

CsrResult CsrDirMake(const CsrUtf8String * dirName);
```

Description

This function is used for creating a new directory.

Parameters

Type	Argument	Description
`const CsrUtf8String *`	`dirName`	The path to the new directory. The same clauses applies to this parameter as described for the `fileName` parameter in 2.1. Except of course the file name part.

Table 14: Arguments to CsrDirMake

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_DIR_RESULT_FAILURE
CSR_DIR_RESULT_NOT_EXIST
CSR_DIR_RESULT_ALREADY_EXIST
CSR_DIR_RESULT_NOT_EMPTY
CSR_DIR_RESULT_INVALID_PATH
```

3.4 CsrDirRemove

Prototype

```
#include "csr_dir.h"

CsrResult CsrDirRemove(const CsrUtf8String * dirName);
```

Description

This function is used for removing a new directory.

NB: if the directory is not empty this operation should fail.

Parameters

Type	Argument	Description
`const CsrUtf8String *`	`dirName`	The path to the new directory. The same clauses applies to this parameter as described for the `fileName` parameter in 2.1. Except of course the file name part.

Table 15: Arguments to CsrDirRemove

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_DIR_RESULT_FAILURE
CSR_DIR_RESULT_NOT_EXIST
CSR_DIR_RESULT_ALREADY_EXIST
CSR_DIR_RESULT_NOT_EMPTY
CSR_DIR_RESULT_INVALID_PATH
```

3.5 CsrDirChange

Prototype

```
#include "csr_dir.h"
```

```
CsrResult CsrDirChange(const CsrUtf8String * dirName);
```

Description

This function is used for changing the current working directory.

Parameters

Type	Argument	Description
`const CsrUtf8String *`	`dirName`	The path to the new directory. The same clauses applies to this parameter as described for the `fileName` parameter in 2.1. Except of course the file name part.

Table 16: Arguments to CsrDirChange

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_DIR_RESULT_FAILURE
CSR_DIR_RESULT_NOT_EXIST
CSR_DIR_RESULT_ALREADY_EXIST
CSR_DIR_RESULT_NOT_EMPTY
CSR_DIR_RESULT_INVALID_PATH
```

3.6 CsrDirFindFirst

Prototype

```
#include "csr_dir.h"

CsrDirHandle *CsrDirFindFirst(const CsrUtf8String * searchPattern, CsrDirFindStat *
dirStat);
```

Description

This function is used for obtaining information about the first instance of a file or directory name that matches the searchPattern in a directory.

Please note that if this operation succeeds it is the responsibility of the application to close the search again with `CsrDirFindClose()`.

Parameters

Type	Argument	Description
`const CsrUtf8String *`	`searchPattern`	The search pattern to match. The search pattern are allowed to contain wildcards and paths like demonstrated in the clauses regarding the `fileName` parameter in 2.1.

`CsrDirFindStat *`	`dirStat`	The information of the first file or directory that matches searchPattern.
		
		The information is filled into the following `CsrFsalDirEntry` struct
		
		typedef struct
		```
		{
		    CsrUtf8String  *name;
		    CsrSize        size;    /* 0 if not file */
		    CsrDirMode     mode;    /* mode */
		    CsrDirTm       time;    /* last modified */
		} CsrDirFindStat;
		```
		Where the the time parameter is specified by the `CsrDirTm` struct below.
		```
		typedef struct
		{
		  CsrTime tm_sec;   /* Seconds: 0-59  */
		  CsrTime tm_min;   /* Minutes: 0-59  */
		  CsrTime tm_hour;  /* Hours since midnight: 0-23 */
		  CsrTime tm_mday;  /* Day of the month: 1-31 */
		  CsrTime tm_mon;   /* Months since january: 0-11 */
		  CsrTime tm_year;  /* Years since 1900 */
		  CsrTime tm_wday;  /* Days since Sunday (0-6) */
		  CsrTime tm_yday;  /* Days since Jan. 1: 0-365 */
		  CsrTime tm_isdst; /* +1 Daylight Savings Time, 0
		No DST, 0xFFFF don't know    */
		  CsrBool utcTime;  /* TRUE=UTC, FALSE=local time */
		} CsrDirTm;
		```
		The possible values for `CsrDirMode` parameter are:
		```
		CSR_DIR_MODE_DIRECTORY
		CSR_DIR_MODE_REGULAR_FILE
		CSR_DIR_MODE_USER_READ_PERMISSION
		CSR_DIR_MODE_USER_WRITE_PERMISSION
		CSR_DIR_MODE_USER_EXECUTE
		CSR_DIR_MODE_GROUP_READ_PERMISSION
		CSR_DIR_MODE_GROUP_WRITE_PERMISSION
		CSR_DIR_MODE_GROUP_EXECUTE
		CSR_DIR_MODE_OTHERS_READ_PERMISSION
		CSR_DIR_MODE_OTHERS_WRITE_PERMISSION
		CSR_DIR_MODE_OTHERS_EXECUTE
		```

Table 17: Arguments to CsrDirFindFirst

Returns

The dirHandle pointer to use in future searches. If the no entries matching `searchPattern` is found this function returns `NULL`.

3.7 CsrDirFindNext

Prototype

```
#include "csr_dir.h"

CsrResult CsrDirFindNext(CsrDirHandle *handle, CsrDirFindStat * dirStat);
```

Description

This function is used for obtaining the next entry in a directory which matches `searchPattern` from `CsrDirFindFirst()`.

Parameters

Type	Argument	Description
`CsrDirHandle *`	`dirHandle`	The handle obtained with `CsrDirFindFirst()`

`CsrDirFindStat *`	`dirStat`	The information of the first file or directory that matches searchPattern.
		
		The information is filled into the following `CsrFsalDirEntry` struct
		
		typedef struct
		`{`
		`    CsrUtf8String  *name;`
		`    CsrSize        size;    /* 0 if not file */`
		`    CsrDirMode     mode;    /* mode */`
		`    CsrDirTm       time;    /* last modified */`
		`} CsrDirFindStat;`
		
		Where the the time parameter is specified by the `CsrDirTm` struct below.
		
		`typedef struct`
		`{`
		`  CsrTime tm_sec;   /* Seconds: 0-59  */`
		`  CsrTime tm_min;   /* Minutes: 0-59  */`
		`  CsrTime tm_hour;  /* Hours since midnight: 0-23 */`
		`  CsrTime tm_mday;  /* Day of the month: 1-31 */`
		`  CsrTime tm_mon;   /* Months since january: 0-11 */`
		`  CsrTime tm_year;  /* Years since 1900 */`
		`  CsrTime tm_wday;  /* Days since Sunday (0-6) */`
		`  CsrTime tm_yday;  /* Days since Jan. 1: 0-365 */`
		`  CsrTime tm_isdst; /* +1 Daylight Savings Time, 0`
		`No DST, 0xFFFF don't know    */`
		`  CsrBool utcTime;  /* TRUE=UTC, FALSE=local time */`
		`} CsrDirTm;`
		
		The possible values for `CsrDirMode` parameter are:
		
		`CSR_DIR_MODE_DIRECTORY`
		`CSR_DIR_MODE_REGULAR_FILE`
		`CSR_DIR_MODE_USER_READ_PERMISSION`
		`CSR_DIR_MODE_USER_WRITE_PERMISSION`
		`CSR_DIR_MODE_USER_EXECUTE`
		`CSR_DIR_MODE_GROUP_READ_PERMISSION`
		`CSR_DIR_MODE_GROUP_WRITE_PERMISSION`
		`CSR_DIR_MODE_GROUP_EXECUTE`
		`CSR_DIR_MODE_OTHERS_READ_PERMISSION`
		`CSR_DIR_MODE_OTHERS_WRITE_PERMISSION`
		`CSR_DIR_MODE_OTHERS_EXECUTE`

Table 18: Arguments to CsrDirFindNext

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_DIR_RESULT_FAILURE
CSR_DIR_RESULT_NOT_EXIST
CSR_DIR_RESULT_ALREADY_EXIST
CSR_DIR_RESULT_NOT_EMPTY
CSR_DIR_RESULT_INVALID_PATH
```

3.8 CsrDirFindClose

Prototype

```
#include "csr_dir.h"

CsrResult CsrDirFindClose(CsrDirHandle *handle);
```

Description

This function is used for closing an ongoing search started with `CsrDirFindFirst()`.

Parameters

Type	Argument	Description
`CsrDirHandle *`	`dirHandle`	The handle obtained with `CsrDirFindFirst()`

Table 19: Arguments to CsrDirFindClose

Returns

The result of the operation.

If successful this should be set to `CSR_RESULT_SUCCESS` if it fails the possible result codes are:

```
CSR_DIR_RESULT_FAILURE
CSR_DIR_RESULT_NOT_EXIST
CSR_DIR_RESULT_ALREADY_EXIST
CSR_DIR_RESULT_NOT_EMPTY
CSR_DIR_RESULT_INVALID_PATH
```

4 Document References

Terms and Definitions

CSR	Cambridge Silicon Radio

Document History

Revision	Date	History
1	02 DEC 09	Ready for release 2.0.1
2	20 APR 10	Ready for release 2.1.0
3	OCT 10	Ready for release 2.2.0
4	DEC 10	Ready for release 3.0.0
5	Aug 11	Ready for release 3.1.0

TradeMarks, Patents and Licences

Life Support Policy and Use in Safety-critical Compliance

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.