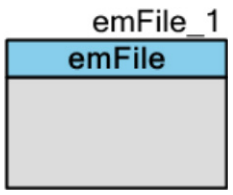
File System Library (emFile)

1.20

Features



- Up to four Secure Digital (SD) cards in SPI mode
- FAT12/16 or FAT32 format
- Optional integration with an Operating System (OS)
- Optional Long File Name (LFN) handling

General Description

The emFile component provides an interface to SD cards formatted with a FAT file system. The SD card specification includes multiple hardware interface options for communication with an SD card. This component uses the SPI interface method for communication. Up to four independent SPI interfaces can be used for communication with one SD card each. Both FAT12/16 and FAT32 file system formats are supported. This component provides the physical interface to the SD card and works with the emFile library licensed from SEGGER Microcontroller to provide a library of functions to manipulate a FAT file system.

When to Use emFile

Use the emFile component to access SD cards formatted using the FAT12/16 or FAT32 file system formats.

Getting Started

Installing the emFile Library

The emFile file system implementation consists of two parts. The first part is the emFile component which is shipped with PSoC Creator. The second part is the emFile file system library licensed from SEGGER Microcontroller. The library is delivered as a zip file that can be downloaded from the Cypress website emFile component page.

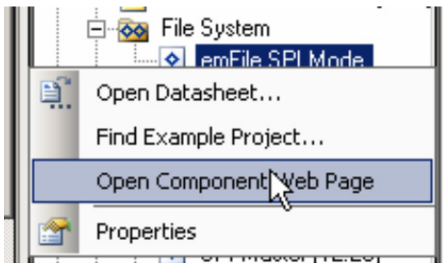
To install the emFile library:

1. Open PSoC Creator and go to the Component Catalog window. Search for the emFile component. It is located under the **Cypress** tab at **Communications > File System > emFile SPI Mode**.

2. Right-click on the emFile component.

   The menu shown in Figure 1 will appear.

   Figure 1. emFile Open Component Web Page

3. Click on Open Component Web Page.

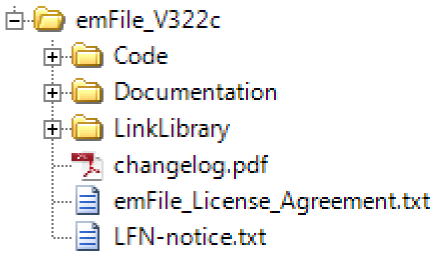
   It will direct you to the component landing page where the zip file with latest file system libraries is located.

4. Download the zip file and extract it to a folder of your choice, preserving the directory structure of the zip file. Make sure the files are not read-only after extracting.

emFile Library Directory Organization

The directory structure of the unzipped emFile library will be similar to that shown in Figure 2.

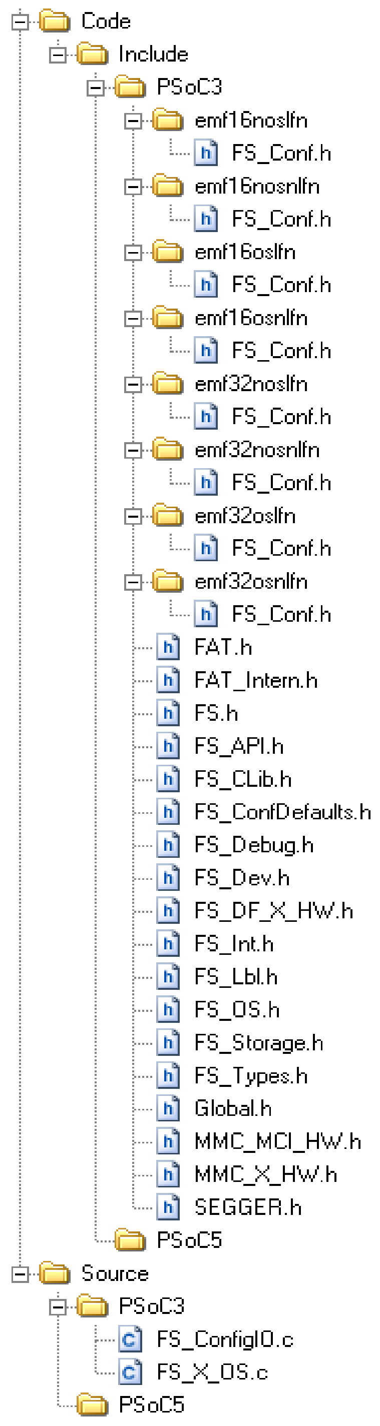
Figure 2. emFile Directory Organization



The Code directory contains the portion of the library that is provided in source code format. This directory has two subdirectories: Include and Source. The Include directory provides header files for the library. The Source directory provides the executable portion of the library that is delivered in source format.

Figure 3 shows files in the Code directory. This figure shows the directory organization of files for PSoC 3. The PSoC 5 directory contains files for PSoC 5 family devices, such as the PSoC 5LP. The PSoC 5 directory is similar to the PSoC 3 file organization.

Figure 3. emFile “Code” Directory File Listing



```
Code
  Include
    PSoC3
      emf16noslfn
        FS_Conf.h
      emf16nosnlfn
        FS_Conf.h
      emf16oslfn
        FS_Conf.h
      emf16osnlfn
        FS_Conf.h
      emf32noslfn
        FS_Conf.h
      emf32nosnlfn
        FS_Conf.h
      emf32oslfn
        FS_Conf.h
      emf32osnlfn
        FS_Conf.h
      FAT.h
      FAT_Intern.h
      FS.h
      FS_API.h
      FS_CLib.h
      FS_ConfDefaults.h
      FS_Debug.h
      FS_Dev.h
      FS_DF_X_HW.h
      FS_Int.h
      FS_Lbl.h
      FS_OS.h
      FS_Storage.h
      FS_Types.h
      Global.h
      MMC_MCI_HW.h
      MMC_X_HW.h
      SEGGER.h
    PSoC5
Source
  PSoC3
    FS_ConfigIO.c
    FS_X_OS.c
  PSoC5
```

The Include section is divided into header files that are always applicable at the top level of the directory and header files in subdirectories that are used depending on the options chosen for the particular LinkLibrary. The names of the subdirectories are specified as emf<options>. All of the options are described in Table 1.

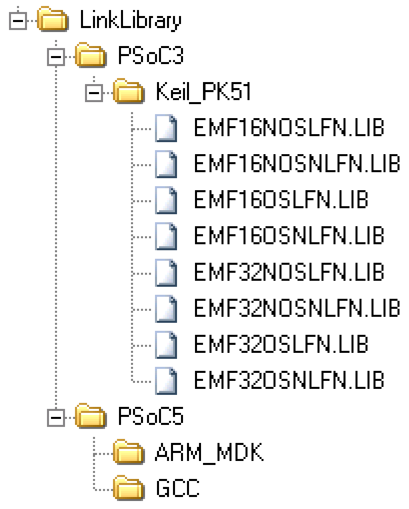
Table 1. Options

Option	Description
16	FAT 12/16 format
32	FAT 32 format
os	Operating System support
nos	No Operating System support
lfn	Long File Name support
nlfn	No Long File Name support

The Documentation section contains the User Guide for the SEGGER Microcontroller emFile file system library.

The LinkLibrary section is divided into PSoC 3 and PSoC 5 directories. Each directory contains a subdirectory for each of the supported tool chains. Within those directories are object libraries that provide the implementation of the file system for the specific options chosen. The library is added to a project using the Build Settings options described later in this datasheet. The expanded PSoC 3/Keil PK51 section contains the object libraries shown in Figure 4.

Figure 4. LinkLibrary Structure



The naming convention for the LinkLibrary is: <prefix>emf<options>.<extension>. The options are the same options used for the include file directory names. The prefix and extension are specific to the tool chain used. See Table 2.

Table 2. Naming Convention

Toolchain	Prefix	Extension
Keil PK51		lib
GCC	lib	a
ARM_MDK		lib

The files in the unzipped emFile directory are *changelog.pdf*, *emFile_License_Agreement.txt*, and *LFN-notice.txt*. The *changelog.pdf* file contains the history on changes that were made in the library. The *emFile_License_Agreement.txt* file is an End User License Agreement that contains the terms of use of the emFile library. The *LFN-notice.txt* file is a notice about the use of long file names.

Selecting the File System (FS) Library

The main aspect that should be considered when selecting the library is the storage capacity of the SD card. The FS_FormatSD() function analyzes the capacity of the card and performs formatting for either FAT 16 or FAT 32. When the SD size is less than or equal to 2 gigabytes the card will be formatted for FAT 16 and if the size of the card is greater than 2 gigabytes the card will be formatted for FAT 32. Therefore for SD card with capacity of 2 gigabytes or less a FAT 16 library should be used and FAT 32 library should be used in other case.

Secondly, it should be considered if the Long File Name (LFN) support is required. If the “no LFN” (or Short File Name) library is used, then file names that do not conform to the “8.3 filename” can’t be created. The “8.3 filename” means that there is a maximum of eight characters that can be used for the actual file name and a maximum of three bytes that can be used for file extension. If the SD card contains the files with long file names and if it will be used with the “no LFN” library then the library will generate the short filename for it. Fox example, a file named “LongNameFile.txt” will have a short file name of “LONGNA~1.TXT”. The “LFN” library allows creating files with file names of maximum 256 bytes.

Note that there are legal issues concerned with usage of LFN. Please read *LFN-notice.txt* for more information.

Next it should be decided if an external embedded OS will be used. If so, then the OS will be controlling the code flow. The specific OS uses a set of tasks that have different priorities and those tasks interrupt each other. Because of the multitasking nature of the OS, there may be collisions while the OS tries to perform simultaneous tasks to file system resources.

To avoid this, there are several API functions that must be implemented. Without those functions, you will not be able to build the project. These API functions implement locking of file system resources. The locking will be performed each time when any file system API function starts its operation and it will unlock the file system resources when the file system API function completes the operation. A detailed description of Locking/Unlocking functions is contained in Chapter 8 “OS integration” of the *emFile User Guide*.

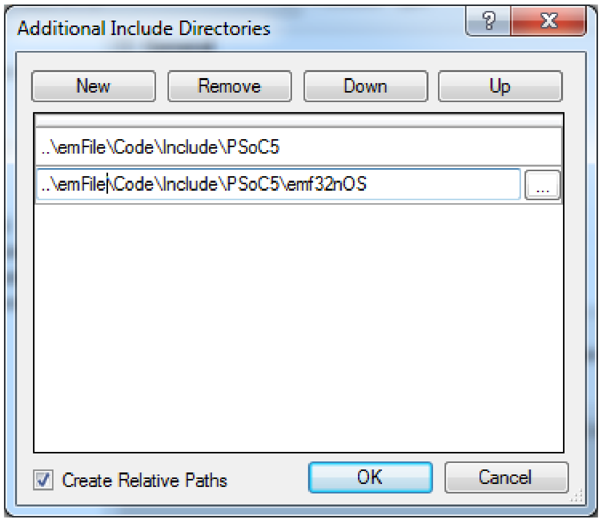
The “no OS” library doesn’t require any additional API functions to be implemented as it uses the forever loop code flow without multitasking.

Creating an emFile Project for a PSoC 5LP Application using GCC toolchain

When using the GCC toolchain, you must specify the directory where the library file is located. The steps to create the emFile project for PSoC 5LP application using GCC toolchain are as follows:

1. Decide which library you need. This decision is based on whether you need FAT12/16 or FAT32, whether your application uses an OS, and whether you need long file name support. This example uses emf32nosnlfn.lib (FAT32, no OS, and no long file name support).
2. Select the necessary include file directory. Go to **Project > Build Settings > ARM GCC 4.8.4 > Compiler > General**. Click the “**…**” button in the **Additional Include Directories** property field. The **Additional Include Directories** dialog will display. Click the **New** button and select an include directory for PSoC 5 and an include directory for the specific options you want. When complete, the dialog should appear as shown in Figure 5.

   Figure 5. Adding include Directories

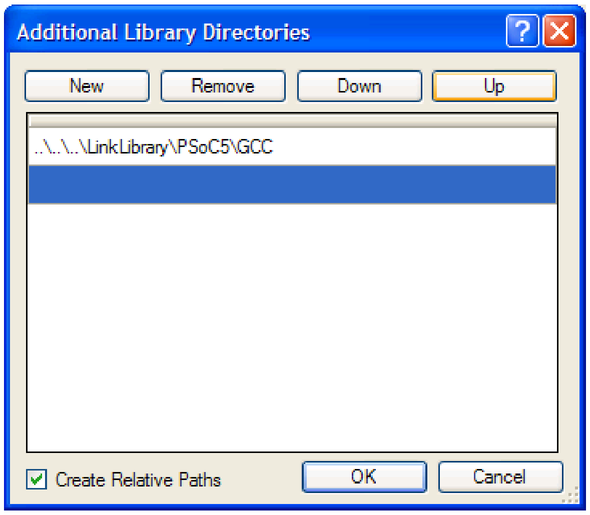
   ```
   ..\emFile\Code\Include\PSoC5
   ..\emFile\Code\Include\PSoC5\emf32nOS
   ```

   Note If want to run the project in both “Debug” and “Release” configurations you need to add the Include Directories for both configurations.
3. Go to **Project > Build Settings > ARM GCC 4.8.4 > Linker > General > Additional Library Directories**. Click the “…” button in the Additional Library Directories property field.

The Additional Library Directories dialog will display.

4. Click the **New** button and select the library directory for the GCC library. When complete, the dialog should appear as shown in Figure 6.

Figure 6. Adding Library Directory

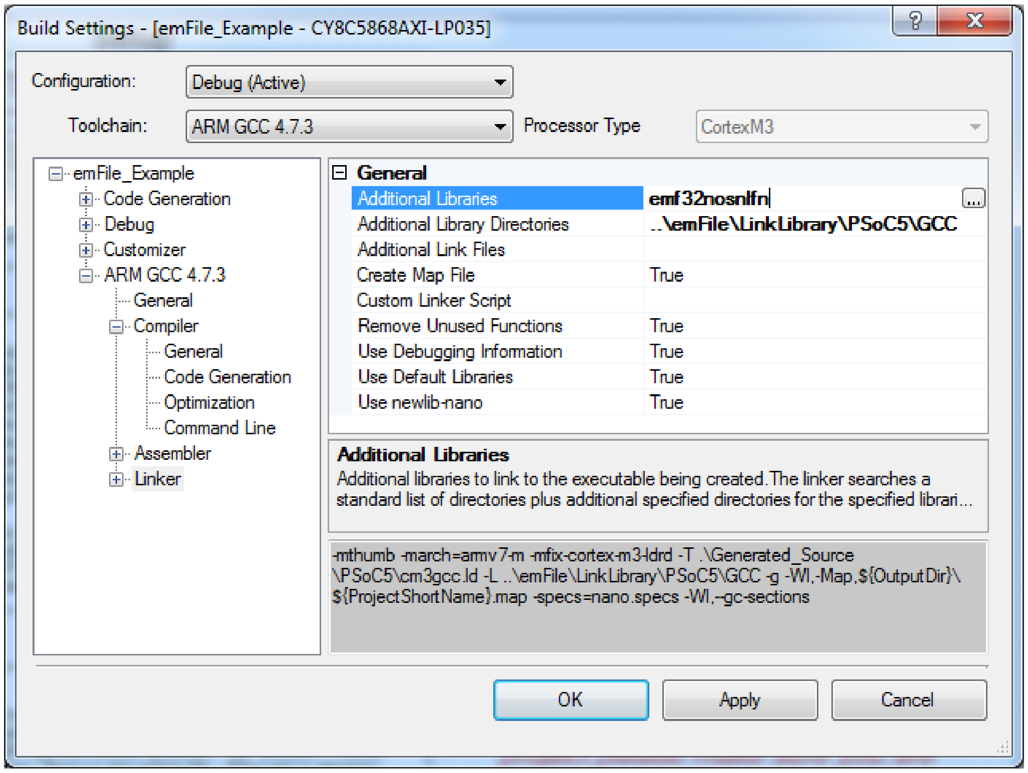


..\..\..\LinkLibrary\PSoC5\GCC

☑ Create Relative Paths

5. Specify the library in the library directory.
   a. Go to **Project > Build Settings > ARM GCC 4.8.4 > Linker > General > Additional Libraries**.
   b. Type in the library name without prefix "lib" and suffix ".a" (for example, "libemf32nosnlfn.a" would be "emf32nosnlfn") see Figure 7. This is because GCC compiler will automatically add the “lib” prefix to the library name.

Figure 7. Specifying Library Name



Configuration: Debug (Active)
Toolchain: ARM GCC 4.7.3　Processor Type: CortexM3

```
emFile_Example
  Code Generation
  Debug
  Customizer
  ARM GCC 4.7.3
    General
    Compiler
      General
      Code Generation
      Optimization
      Command Line
    Assembler
    Linker
```

General

Additional Libraries	emf32nosnlfn
Additional Library Directories	..\emFile\LinkLibrary\PSoC5\GCC
Additional Link Files	
Create Map File	True
Custom Linker Script	
Remove Unused Functions	True
Use Debugging Information	True
Use Default Libraries	True
Use newlib-nano	True

Additional Libraries
Additional libraries to link to the executable being created. The linker searches a standard list of directories plus additional specified directories for the specified librari...

```
-mthumb -march=armv7-m -mfix-cortex-m3-ldrd -T .\Generated_Source
\PSoC5\cm3gcc.ld -L ..\emFile\LinkLibrary\PSoC5\GCC -g -Wl,-Map,${OutputDir}\
${ProjectShortName}.map -specs=nano.specs -Wl,--gc-sections
```

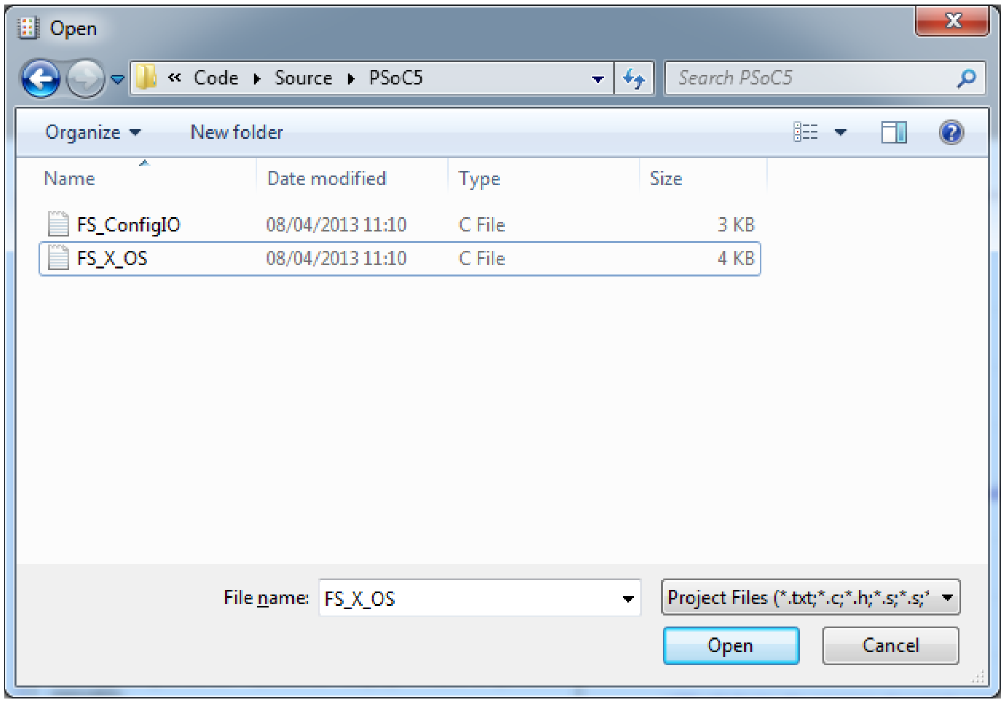
Note If want to run the project in both “Debug” and “Release” configurations you need to add the Link library for both configurations.

6. If usage of OS or logging feature is desired then *FS_ConfigIO.c* or *FS_X_OS.c* files should be added to the project. Details on OS integration and logging feature usage can be found in the emFile User Guide in Chapters 8 and 9 respectively.

   The mentioned files can be added directly into your project from the Code/Source/PSoC5 directory or copied first to your project directory and added from there. The files will usually be edited, so you need to decide whether to change the original files or a copy that is specific to the project.

   The files are added to the project using **Project > Existing Item**. The dialog shown in Figure 8 will open to allow you to select the files.

Figure 8. Adding PSoC 5 Source Files



Name	Date modified	Type	Size
FS_ConfigIO	08/04/2013 11:10	C File	3 KB
FS_X_OS	08/04/2013 11:10	C File	4 KB

File name: FS_X_OS

Project Files (*.txt;*.c;*.h;*.s;*.s;*

5. Add <FS.h> header file in your main.c function.

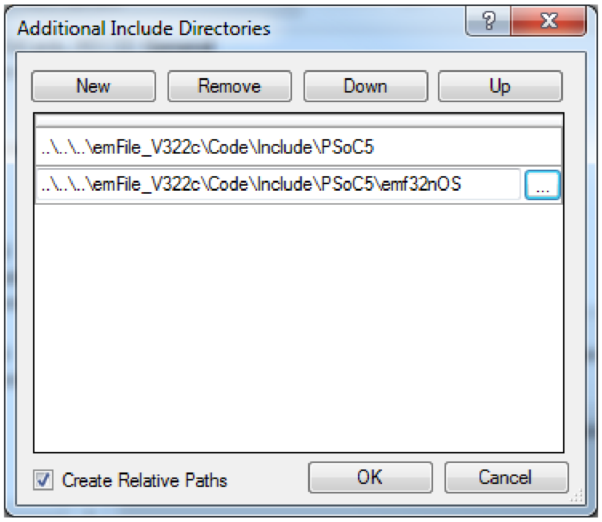
The PSoC 5LP emFile library is now included in your project.

Creating an emFile Project for a PSoC 5LP Application using MDK Toolchain

There are also libraries for the ARM MDK compiler in the *.zip file. The project creation steps for ARM MDK libraries are the same as the steps as using the GCC Toolchain, except for the location of the files to include.

1. Decide which library you need. This decision is based on whether you need FAT12/16 or FAT32, whether your application uses an OS, and whether you need long file name support. This example uses emf32nosnlfn.lib (FAT32, no OS, and no long file name support).
2. Select the necessary include file directory. Go to **Project > Build Settings > ARM MDK Generic > Compiler > General**. Click the "**…**" button in the **Additional Include Directories** property field. The **Additional Include Directories** dialog will display. Click the **New** button and select an include directory for PSoC 5 and an include directory for the specific options you want. When complete, the dialog should appear as shown in Figure 9.

Figure 9. Adding Include Directories

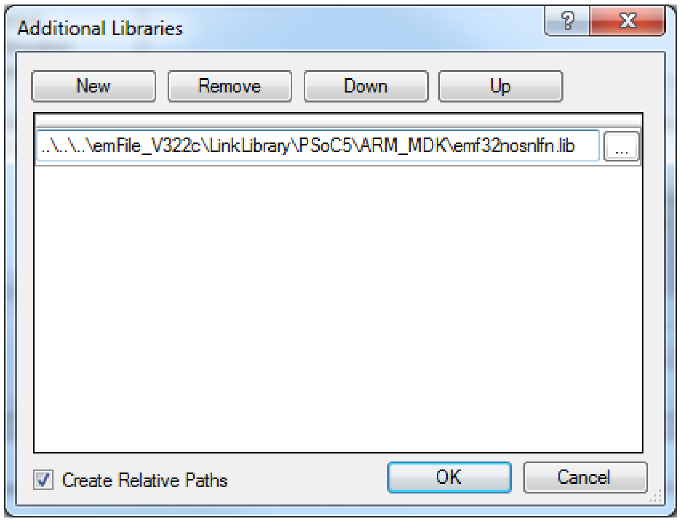


..\..\..\emFile_V322c\Code\Include\PSoC5
..\..\..\emFile_V322c\Code\Include\PSoC5\emf32nOS

Note that if want to run the project in both "Debug" and "Release" configurations you need to add the Include Directories for both configurations.

3. Select the Link library file you need. Go to **Project > Build Settings > ARM MDK Generic > Linker > General**. Click the "**…**" button in the **Additional Libraries** property field. The **Additional Link Files** dialog will display. Click the **New** button and select the library file based on the specific options you want. When complete, the dialog should appear as shown in Figure 12.

Figure 10. Adding a Link Library



..\..\..\emFile_V322c\LinkLibrary\PSoC5\ARM_MDK\emf32nosnlfn.lib

Note If want to run the project in both "Debug" and "Release" configurations you need to add the Link library for both configurations.

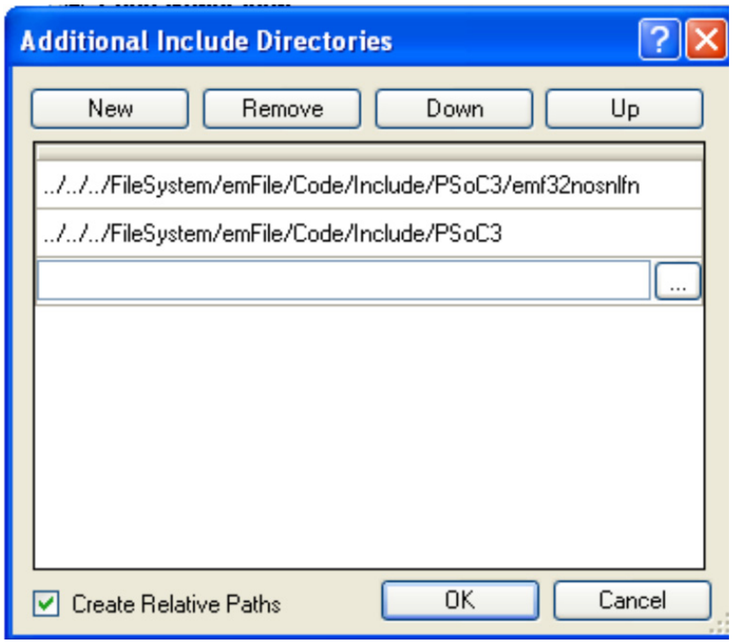
6. Add the source file FS_ConfigIO.c into the project if you want to use debug logging features of the file system library. This step is the same as step #6 in “Creating an emFile Project for a PSoC 5LP Application using GCC toolchain”
7. Add <FS.h> header file in your main.c function.

Creating an emFile Project for a PSoC 3 Application

To use the emFile library in a PSoC 3 project:

1. Decide which library you need. This decision is based on whether you need FAT12/16 or FAT32, whether your application uses an OS, and whether you need long file name support. This example uses emf32nosnlfn.lib (FAT32, no OS, and no long file name support).
2. Select the necessary include file directory. Go to **Project > Build Settings > DP8051 Keil 9.51 > Compiler > General**. Click the “**…**” button in the **Additional Include Directories** property field. The **Additional Include Directories** dialog will display. Click the **New** button and select an include directory for PSoC 3 and an include directory for the specific options you want. When complete, the dialog should appear as shown in Figure 11.

Figure 11. Adding Include Directories

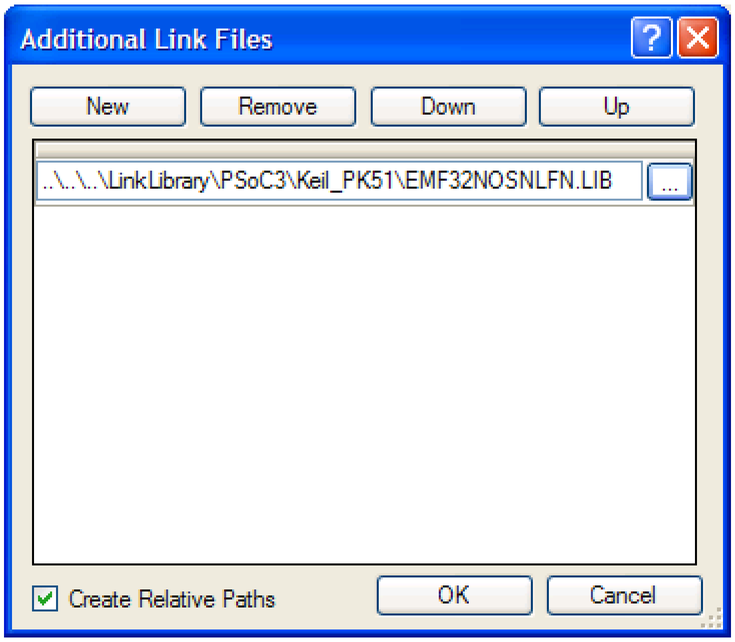


../../../FileSystem/emFile/Code/Include/PSoC3/emf32nosnlfn
../../../FileSystem/emFile/Code/Include/PSoC3

Note that if want to run the project in both “Debug” and “Release” configurations you need to add the Include Directories for both configurations.

4. Select the Link library file you need. Go to **Project > Build Settings > DP8051 Keil 9.51 > Linker > General**. Click the "**…**" button in the **Additional Link Files** property field. The **Additional Link Files** dialog will display. Click the **New** button and select the library file based on the specific options you want. When complete, the dialog should appear as shown in Figure 12.

Figure 12. Adding a Link Library



..\..\..\LinkLibrary\PSoC3\Keil_PK51\EMF32NOSNLFN.LIB

☑ Create Relative Paths

Note If want to run the project in both "Debug" and "Release" configurations you need to add the Link library for both configurations.

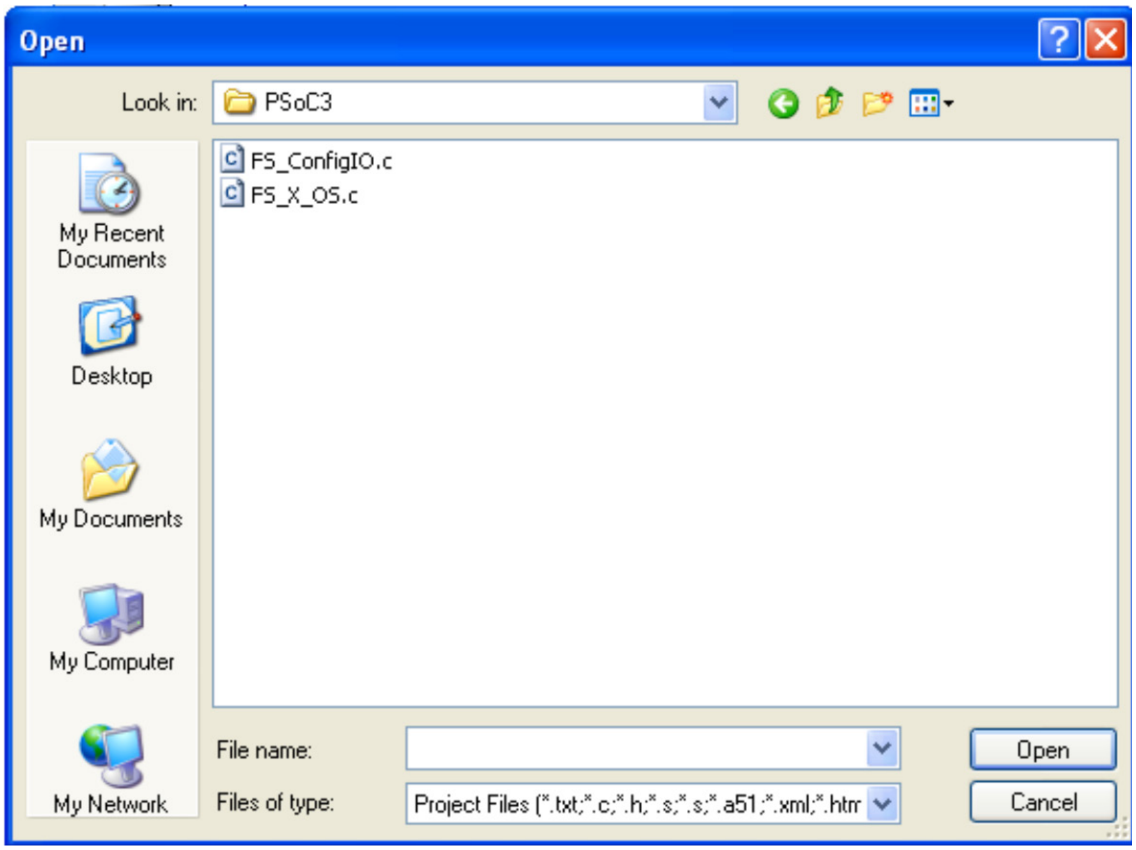
5. Add the source file *FS_ConfigIO.c* into the project if you want to use debug logging features of the file system library. For detailed information on debugging, refer to Chapter 9 of the emFile User guide.

   If you are using an OS, also add *FS_X_OS.c.* For more details on OS integration, refer to Chapter 8 of emFile User guide.

   These files can be added directly into your project from the Code/Source/PSoC3 directory or copied first to your project directory and added from there. The files will usually be edited, so you need to decide whether to change the original files or a copy that is specific to the project.

   The files are added to the project using ***Project > Existing Item***. The dialog shown in Figure 13 will open to allow you to select the files.

Figure 13. Adding PSoC 3 Source Files



FS_ConfigIO.c
FS_X_OS.c

File name:

Files of type: Project Files (*.txt;*.c;*.h;*.s;*.s;*.a51;*.xml;*.htm

6. Add <FS.h> header file in your main.c function.

The PSoC 3 emFile library is now included in your project.

Input/Output Connections

This section describes the various input and output connections for the emFile component. An asterisk (*) in the list of I/Os means that the I/O may be excluded from the component under the conditions listed in the description of that I/O.

There are no visible connections for the component on the symbol. All connections shown are for pin connections that are included internal to the component. These pins will all appear in the Design-Wide Resources Pin Editor. They must be assigned to the appropriate physical pins using the Pin Editor. For each of the connections there are four pins named with indexes 0 to 3. These represent the four independent SD cards that the component can support. One to four of these pins will be present in the design, depending on the number of SD cards selected. The following is a description of these pins.

SPI0_WP – SPI3_WP*

Optional input pins based on **SD Card [0-3] Write Protection** options, which configure the write protect for SD cards [0-3]. These pins are present if these options are checked. The default state for these pins is not present, which indicates that the SD cards are not write protected.

mosi0 – mosi3

Master Out Slave In pin of the SPI Master. This pin should be connected to the DI/CMD pin of the SD card.

miso0 – miso3

Master In slave Out pin of the SPI Master. This pin should be connected to the DO/DAT0 pin of the SD card.

sclk0 – sclk3

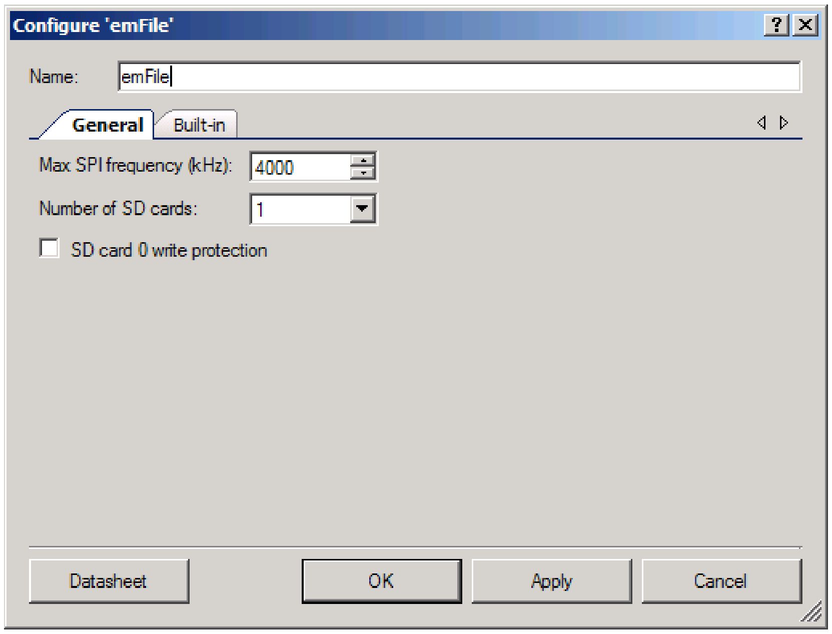
Serial Clock pin from the SPI Master. This pin should be connected to the clock pin of the SD card.

SPI0_CS – SPI3_CS

Card Select output pin. This pin should be connected to the nCS pin of the SD card.

Component Parameters

Drag an emFile onto your design and double-click it to open the Configure dialog.



Name: emFile

Max SPI frequency (kHz): 4000

Number of SD cards: 1

☐ SD card 0 write protection

The emFile provides the following parameters.

Max SPI frequency

Defines the maximum frequency at which the SPI Master components will clock an SD card. The value is specified in kHz. Possible values are in the range 400 to 25,000 kHz, the default setting is **4000**.

The range of frequencies is based on the specification for SD cards. Note that there are limitations for maximum frequency for the SPI Master (v2.20) component that is used in emFile. Refer to the SPI Master v2.20 datasheet "Timing Characteristics" for detailed information about the maximum frequency that this component can support.

Number of SD cards

Defines the number of SD cards in the emFile system. The value can be set between 1 and 4. The default setting is **1**.

SD card *n* write protection

Defines the write protect enable for each of *n* SD cards. It is disabled by default.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name “emFile_1” to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is “emFile.”

The emFile library routines automatically initialize and enable the emFile component. The only APIs provided for the emFile component are to support transitioning between power modes.

This section does not include a description of file system APIs of the emFile library because these APIs are described in the *emFile User Guide*, Chapter 4.

Note To use long file names (LFN) support on PSoC 5LP devices, you must call FS_FAT_SupportLFN(). For PSoC 3 devices, this feature is enabled by default.

Functions

Function	Description
emFile_Sleep()	Prepares emFile to enter sleep mode.
emFile_Wakeup()	Restores emFile after coming out of sleep mode.
emFile_SaveConfig()	Saves the SPI Master configuration used by the HW driver.
emFile_RestoreConfig()	Restores the SPI Master configuration used by the HW Driver.

void emFile_Sleep(void)

Description:	Prepares emFile to go to sleep.
Parameters:	None
Return Value:	None
Side Effects:	None

void emFile_Wakeup(void)

Description:	Restores emFile after coming out of sleep mode.
Parameters:	None
Return Value:	None
Side Effects:	Calling the emFile_Wakeup() function without first calling the emFile_Sleep() or emFile_SaveConfig() function may produce unexpected behavior.

void emFile_SaveConfig(void)

Description:	Saves the SPI Master configuration used by the HW driver. This function is called by emFile_Sleep().
Parameters:	None
Return Value:	None
Side Effects:	None

void emFile_RestoreConfig(void)

Description:	Restores the SPI Master configuration used by the HW Driver. This function is called by emFile_Wakeup().
Parameters:	None
Return Value:	None
Side Effects:	Calling this function without first calling the emFile_Sleep() or emFile_SaveConfig() function may produce unexpected behavior.

Sample Firmware Source Code

PSoC Creator provides many example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator components

- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

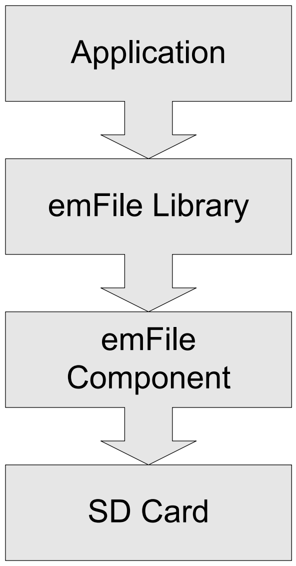
The emFile component has not been verified for MISRA-C:2004 coding guidelines compliance.

Functional Description

File System Library Structure

The emFile file system implementation consists of two parts: the emFile component and the emFile file system library licensed from SEGGER Microcontroller. A file system application will use the APIs provided in the emFile library. That library will use the emFile component to provide the physical interface to the SD card using a SPI interface. The structure of the emFile file system is shown in Figure 14.

Figure 14. emFile Structure



Placement

The emFile component is placed throughout the UDB array and all placement information is provided to the API through the *cyfitter.h* file.

Performance and Resource Usage

The performance of emFile depends on a set of parameters (CPU, compiler, optimization, size of payload data) and it is limited by the maximum speed of the SPI Master component. The following table contains read/write speed values depending on a variety of factors that affect emFile component performance. The measurements were taken in "Release" mode with optimization level set to "Speed."

			Performance (KBps) [1]							
			1 Byte		256 Bytes		1 KB		8 KB	
Device	CPU Speed	SPI Clock	Write	Read	Write	Read	Write	Read	Write	Read
PSoC 3	24 MHz	12 MHz	0.02	0.09	4.46	20.00	15.38	47.17	22.04	47.34
PSoC 3	48 MHz	12 MHz	0.03	0.18	8.15	38.20	28.33	87.72	41.67	93.02
PSoC 3	24 MHz	8 MHz	0.02	0.09	4.46	19.69	15.38	47.17	22.04	47.33
PSoC 3	48 MHz	8 MHz	0.03	0.17	7.83	36.05	27.10	84.03	39.70	87.91
PSoC 3	24 MHz	4 MHz	0.02	0.10	4.27	19.40	14.70	44.64	16.70	44.94
PSoC 3	48 MHz	4 MHz	0.03	0.15	6.98	32.40	23.98	74.07	35.08	77.67
PSoC 5 LP	24 MHz	12 MHz	0.05	0.53	11.13	134.74	45.71	276.76	182.86	330.32
PSoC 5 LP	48 MHz	12 MHz	0.05	0.83	15.24	196.92	58.85	426.67	262.56	553.51
PSoC 5 LP	24 MHz	8 MHz	0.05	0.53	11.13	134.74	45.31	269.47	191.40	330.32
PSoC 5 LP	48 MHz	8 MHz	0.05	0.66	12.55	170.67	51.72	353.10	230.11	422.27
PSoC 5 LP	24 MHz	4 MHz	0.04	0.36	8.83	91.43	32.82	196.92	109.23	215.58
PSoC 5 LP	48 MHz	4 MHz	0.04	0.50	11.13	116.36	41.29	238.14	160.63	276.76

1 The numbers were measured with File Buffer and Cache features off.

Memory Usage

emFile component memory usage varies depending on the application. The documentation from SEGGER gives a detailed description of how to calculate the memory resources. The following table contains memory usage values for some commonly used features of the file system. All values are in bytes.

	Keil_PK51				**GCC**			
	Release		**Debug**		**Release**		**Debug**	
emFile Module	**Flash**	**SRAM**	**Flash**	**SRAM**	**Flash**	**SRAM**	**Flash**	**SRAM**
File system core (SPI driver)	28035	4762	28799	4849	10872	4272	12376	4272
Read file	2668	6	2672	3	832	0	832	0
Write file	1927	0	1932	0	808	0	816	0
Open directory	2714	47	2733	47	408	0	424	0
Create directory	898	0	898	0	464	0	464	0
Remove file	75	0	75	0	56	0	64	0
Long file name support [2]	-	-	-	-	2216	0	2216	0
Low-level format	769	0	769	0	248	0	256	0
Format SD card	6063	0	6063	0	2288	0	2288	0

Component Errata

This section lists known problems with the component.

Cypress ID	**Component Version**	**Problem**	**Workaround**
191257	v1.20	This component was modified without a version number change in PSoC Creator 3.0 SP1. For further information, see Knowledge Base Article KBA94159 (www.cypress.com/go/kba94159).	No workaround is necessary. There is no impact to designs.

emFile Library Changes

This section lists the major changes in the Library Files for different versions.

2. Long file name support code is included automatically as part of the file system core for PSoC 3.

Version	Description of Changes	Reason for Changes / Impact
emFile_V322c	A call to UART_PutString() was removed from the following functions in FS_ConfigIO.c: FS_X_Log(); FS_X_Warn(); FS_X_ErrorOut().	The UART_PutString() was causing compile error when it was added to the project.
	RVDS libraries were removed from the emFile Library.	Support of RVDS compiler was dropped from PSoC Creator.
emFile_V322b	Changed type of internal variable – NumBytesAvail from U16 to U32 in FS_FAT_Read() routine.	There was an issue with U16 value overflow caused failures while trying to read file larger than 65536 bytes. (This change only applied to PSoC 3 version of the library).
emFile_V322a	Two internal routines of MMC driver were updated to return U32 value instead of U16.	These routines were involved to process of fetching SD card medium size. And in the case when High Speed SD card was used and the card had large memory capacity (from 4 GB) the routines returned improper SD card medium size. (This change only applied to PSoC 3 version of the library).
	Replaced usage of generic pointer with the usage of xdata pointer in internal routine called by FS_UnmountForced().	There was an issue that made system to hung for the case when FS_UnmountForced() was called to unmount the medium after SD card was removed. The reason for that relied in incorrect usage of memory specific pointers. (This change only applied to PSoC 3 version of the library).
emFile_V322	Initial version	

Component Changes

Version	Description of Changes	Reason for Changes / Impact
1.20.f	Datasheet update. Performance numbers for PSoC 5 LP was added to the table in **Performance and Resource Usage** section.	To fill in the gap in the component characterization data.
1.20.e	Datasheet update. Added more description to section "Creating an emFile Project for a PSoC 5LP Application using MDK toolchain"	To avoid possible confusion in creating emFile project for MDK compiler.
	Datasheet update. Added emFile Library Changes section.	To document various changes made to the library files.
1.20.d	Edited datasheet to add Component Errata section.	Document that the component was changed, but there is no impact to designs.

Version	Description of Changes	Reason for Changes / Impact
1.20.c	Edits to the datasheet.	Fixed a few typos.
1.20.b	Added **Selecting the File System Library** section to the datasheet.	This section provides guidelines for selection of proper FS library depending on features required for a project.
	New section "Creating an emFile Project for a PSoC 5LP Application using MDK and RVDS toolchains" was added to the datasheet.	
	More description was added to section "Creating an emFile Project for a PSoC 5LP Application using GCC toolchain".	To give more details on creating an emFile project for PSoC 5LP.
	Section "Creating an emFile Project for a PSoC 5 Application" was renamed to "Creating an emFile Project for a PSoC 5LP Application using GCC toolchain".	
1.20a	Updated Figure 2, emFile directory organization diagram.	
1.20	Added MISRA Compliance section.	The component was not verified for MISRA compliance.
	Updated emFile with the latest version of the SPI Master, clock and pin components.	
1.10	Updated Performance table values.	
	Updated emFile Directory Organization diagram.	The emFile File system library (.zip) was updated so this was reflected in the datasheet.
	Updated emFile with the latest version of the SPI Master component.	
1.0	First release.	