FlashFile v3.0

File System Library
and
Communications Protocol

for
Atmel AVR Microcontrollers
and

Secure Digital, miniSD, microSD, and Multimedia Cards

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

priio™
developers of intelligent, interactive products

Revision History

Revision	Description	Date
0.1	Initial Release	3/7/08
1.1	Removed CompactFlash references from document since they are no longer supported by the software package.	7/8/08

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

priio™
developers of intelligent, interactive products

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

priio™
developers of intelligent, interactive products

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

priio™
developers of intelligent, interactive products

1 Updates in version 3.0

1. Added setup information for the Rowley CrossWorks AVR IDE Compiler.
2. The IAR Embedded Workbench IDE Compiler is no longer supported.
3. CompactFlash media is no longer supported.
4. Changed code and examples to be more MISRA compatible.
5. Added "Block Write" functions for SD/MMC media.

2 Updates in version 2.11

1. Added pin out for miniSD and microSD in the Implementation section.

3 Updates in version 2.10

Version 2.10 represents a shift in addressing scheme and code efficiency. The full revision history with bug fixes can be found at the top of the "file_sys.c" file. Be sure to read the documentation thoroughly especially concerning the follow points:

1. The directory listings addressing scheme for the Flash File has changed to use pointers to the Directory/File Entry to optimize code space.
2. The addition of the _NO_MALLOC_ definable allows better allocation of memory and code space if malloc is no used elsewhere.

4 Updates in version 2.0

1. The addressing scheme for the Flash Media has changed from using direct addressing to sector addressing. This allows for less code since most of the calculations for addressing were being done using sector addressing anyway.
2. Unions and structures were used to get values to and from the Flash Media so the instructions used could be cut down from using bit shifting and bitwise AND's.
3. The "options.h" setup file has more handles included so that the user has more control over different features, including hard coding the number of bytes per sector of the Flash Media used.
4. fread() and fwrite() functions add to the Flash File system.

5 Description

This document describes the setup and usage of the SD/miniSD/microSD/MMC package (sold separately) of the FlashFile. The FlashFile is a file system library for flash media, which reads and writes data in a FAT12/FAT16 format to SD/miniSD/microSD/ MMC cards using the Atmel AVR processors.

The use of a FAT12/FAT16 file format allows the user to read, write, append, create, or delete files on a flash media card via an Atmel AVR processor with the ability to read, write, append, create, or delete the same files using a PC card reader (and vise

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

versa).

The fully functional file system requires approximately 16-20kB of Programmable Flash (assuming all code optimizations are used for size), but the program flash usage can be lowered if not all functionality is needed (i.e. Read Only, No Directory Support, etc.).

Note: Atmel DataFlash cards are NOT standard PC Multimedia cards and will NOT work with the FlashFile without modification.

6 Overall Scheme

The FlashFile library works by connecting a standard FAT12/FAT16 formatted Secure Digital/Multimedia card to the SPI bus of the Atmel AVR processor. When the file system is included in the user's project, Standard Input/Output C Functions used for file handling like fopen, fclose, fgetc, fputc, etc. are available to the Atmel AVR Processor to read from and write to files located on a SD/MMC.

When using SD/MMC media, the "sd_cmd.c" file interfaces with the processor and handles all of the "behind the scenes" interfacing between the SD/MMC card and the Atmel AVR processor. The "file_sys.c" file is where all of the "C" file commands are located.

The 16-20kB of programmable Flash that the library requires is for fully functional read and write commands to the selected flash media, with both FAT12 and FAT16 translations. The amount of programmable Flash used for the FlashFile library can be significantly decreased if support for FAT12, directories, and/or writing to the card are not needed.

A complete example of how to use the FlashFile is located at the end of this document.

7 Implementation

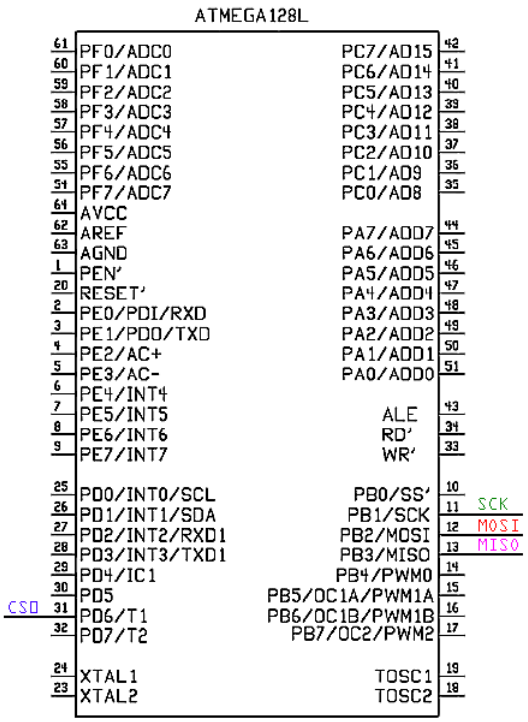
The FlashFile library is implemented by inserting source files into the code produced by a code generator (like the CodeVisionAVR wizard) or written by a programmer; a complete example is presented in a later section. The library files are "sd_cmd.c", "file_sys.c", "options.h", ("twi.c" if using the real time clock option), and their respective header files. The "options.h" file should be included in the main project source file, and the other source files should be located in the directory specified in "options.h" (see the **OPTIONS.H Header File** section for details).

7.1 Secure Digital, miniSD, microSD, and Multimedia Cards

The Secure Digital card should be hooked up to the SPI bus of the Atmel AVR Processor with the Atmel AVR processor as the master. The test code included

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

with the project assumes an Atmel ATMega128 (with 3.3V conversions if necessary) is used, with the MOSI, MISO, and SCLK pins of the SD card connected to the SPI bus of the Mega128. CS0 is the card select of the SD card, and could be changed to use any output pin desired (as long as it is properly defined by SD_CS_ON(), SD_CS_OFF(), and CS_DDR_SET() in the "options.h" file).



SPI Mode			Pin Out		
Name	I/O Type	Description	SD	miniSD	TransFlash
~CS	I	Chip Select (CS0)	1	1	2
DI	I	Data In (MOSI)	2	2	3
V_{ss}	S	Ground	3,6	3,6	6
V_{DD}	S	Supply Voltage	4	4	4
SCLK	I	Clock (SCLK)	5	5	5
DO	O	Data Out (MISO)	7	7	7
RSV	-	Reserved	8,9	8,9	8
N/C	-	No Connection	-	10,11	-

Figure 1 - Hardware Diagram for an ATMega128 Interfaced with a SD, miniSD or microSD Card

Note: Secure Digital Cards are rated at a Maximum of **3.6V**, all voltage signals to and from the card should be changed so that a "high" or a Logic "1" and the V_{dd} is 3.3V, and a "low" or a Logic "0" is 0V.

8 File System

The FlashFile is setup so that it can read and write to either FAT12 or FAT16 formatted flash media. Most flash cards 64MB and lower come preformatted in FAT12, although if formatted on the PC, all cards 32MB and above are formatted in FAT16. FAT12 support is only needed for reading/writing to flash cards that are less than 32MB (the format must be checked if 64MB or less); anything larger requires

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

FAT16. Media larger than 2 GB must be formatted in FAT32, which is currently not supported by the FlashFile library (SD/MMC cards are currently only available up to 1GB). If partitions are used, the FlashFile will assume the first partition will be used to read and write files and folders. File and path names used in the FlashFile must be in the 8.3 format, which mean up to 8 characters (no spaces or symbols other than '-' and '_') and the 3 character file type. If a file that needs to be accessed was previously saved in the long file name format, the 8.3 filename format must be used to access the card with the Atmel AVR processor (i.e. if a file is called "Long Filename.txt" in the PC, the AVR processor will see it as "LONGFI~1.TXT").

9 Memory Considerations

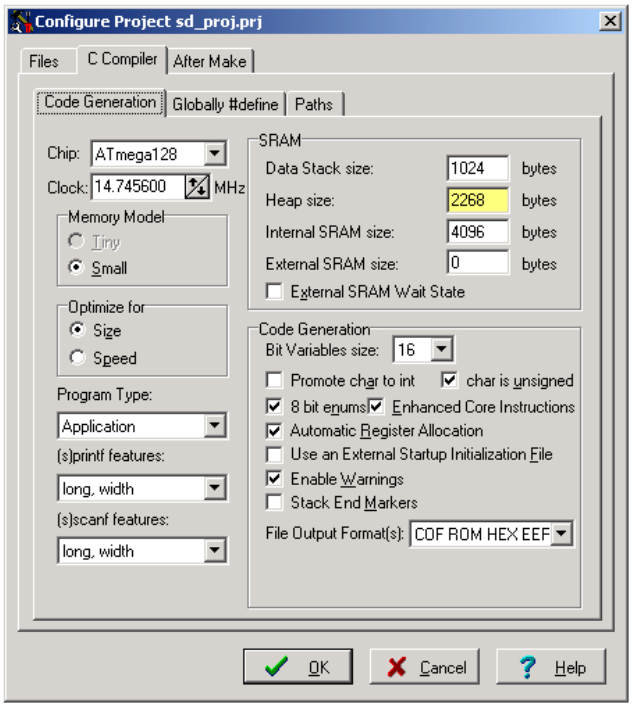
The Flash cards must be read and written in blocks of 512 bytes. This means that a 512 byte buffer must be held for reading and writing to the card at all times. All global variables (including the SD buffer) take up approximately 630 bytes and the data stack can take up 128-512 bytes of SRAM (depending on the project requirements). The Heap size of the file system is variable, based on the number of files that need to be open simultaneously. Each file that is open requires 560 bytes in the heap (512 bytes for the FILE data buffer, 41 bytes for the needed FILE structure information, + 8 bytes for the FILE pointer overhead). This means that if only one file needs to be opened at a time, the Heap size must be at least 561 bytes, two open files at once requires at least a 1122 byte Heap size, three open files at once requires at least a 1683 byte Heap size, etc. A heap size larger than the required amount requires more SRAM use, but gives the user greater margin for error to deal with when allocating memory space.

Note: A heap is not required if the *#define _NO_MALLOC_* is compiled, and the number of files that can be opened at once will be the defined by the *#define _FF_MAX_FILES_OPEN*. This option puts the file structures used in global variable space

9.1 CodeVision AVR Heap Setup

To set up the Heap size in CodeVisionAVR v1.24 or later (earlier versions do not support memory allocation), from the main menu select "*Project => Configure*" and select the "*C Compiler => Code Generation*" tabs. Under "*SRAM => Heap size*", enter the Heap size the project requires:

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com



CodeVisionAVR Project Configuration Menu

9.2 ImageCraft AVR Heap Setup

To set up the Heap size in ImageCraft AVR, the "void _NewHeap(void *start, void *end)" function must be called when initializing the AVR processor (before the while(1) in the main() function). The **start* variable is a pointer address of where the heap starts, and the **end* variable is a pointer address of where the heap ends. The global library variable *extern char _bss_end* holds the SRAM address of the last used global variable. A common starting address for the heap is *&_bss_end +1*, with the ending address for the heap at *&_bss_end + _heap_size_ +1*.

Example:

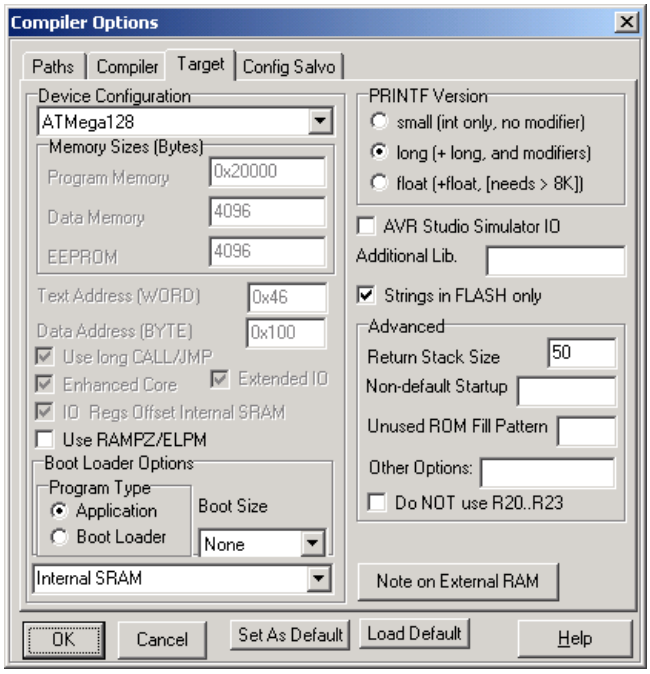
To declare a heap size of 2048 bytes (2kB):

```
extern char _bss_end;
main()
{
 init_devices();
    _NewHeap(&_bss_end + 1, &_bss_end + 2048 + 1);
    // remaining code typed below
```

ImageCraft AVR automatically saves constant strings (i.e in a printf statement "Hello World") to both the SRAM and Flash space. Since the heap is required to be at least 560 bytes, be sure to select "Strings in FLASH only" and use "cprintf" instead of "printf" if applicable (the "options.h" file #defines

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

printf as cprintf for where they are required in code) in the “Compiler Options => Target” window. Since Long variables are used also in “read_directory” and debugging, select “long (+ long, and modifiers)” under “FPRINTF Version”. The “Return Stack Size” must be set to at least 50 to work with the nested functions within the FlashFile.

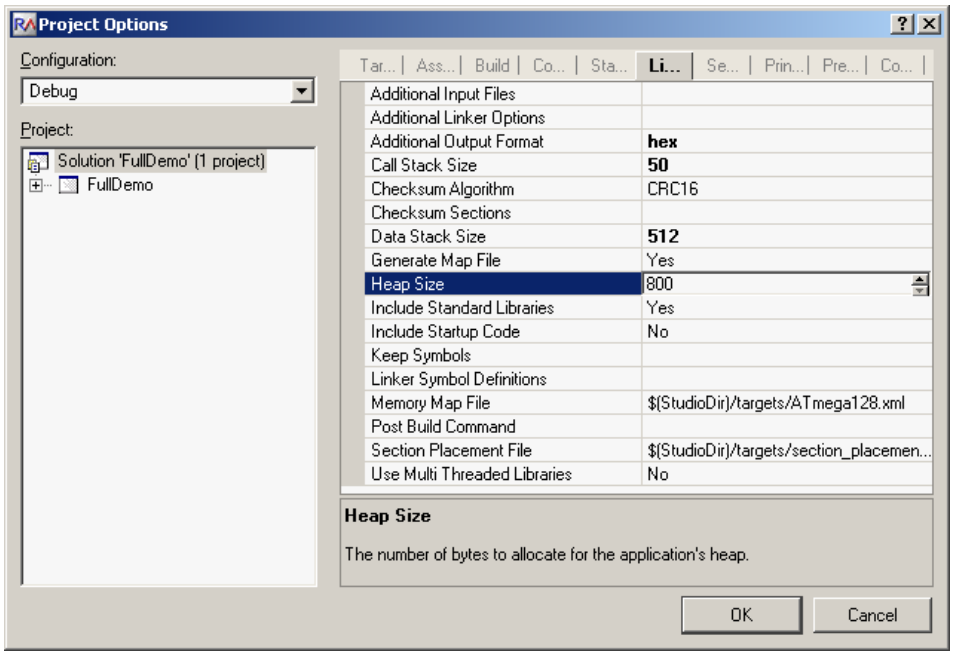


ICC Compiler Options

9.3 Rowley CrossWorks for the AVR Heap Setup

To set up the Heap size in Rowley CrossWorks for AVR, select from the main menu, “*Project => Properties*”, then select the applicable Project (not the file) in the Project window. Under the “Linker” tab, enter the heap size required for the project in the “*Heap Size*” row. The “Call Stack Size” is also accessible from this menu, and should be set to at least 50 to deal with all of the nested functions in the FlashFile.

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com



Rowley CrossWorks for the AVR Project Options

9.4 V2.10 Update

An update to the FlashFile was added so that a malloc() function is no longer required, allowing the #define _NO_MALLOC_ to be in the "options.h" file and specifying the number of files to be opened at once will allocate that space to global memory.

10 Timing Considerations

The FlashFile is set up so that all files are read and written to using a 512 byte SRAM buffer for each file. Because the Flash cards must be read and written to using these 512 byte buffers, there may be short delays while reading and writing the card data when program needs to fill and/or flush the buffer. If a large card is being used (512MB, 256MB, 128MB) and there is a large portion of it previously filled, the initial write (the first time a write to file is called) may take longer than usual due to the processor looking for an empty space. If the card being used has a lot of fragmented data, read and write operations may also take longer since the processor has to jump around to different parts of the card to read or write data. The best method to reduce the fragmentation on a card is to copy all of the data to the PC, delete all of the data on the card, and then copy the data saved on the PC back to the card.

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

The file system used (i.e. FAT12/FAT16) and the size of media used (i.e. 32MB, 64MB, 128MB, etc.) in the project also determines the speed that the data transfer works at. A FAT12 system usually uses larger cluster sizes, which means that the FAT table does not have to be updated as often as a FAT16 system.

11 Disk Space Management

The FlashFile uses the FAT12/16 specifications for files, which means that the disk space used is dependant on the cluster size of the file. The total disk space available depends on the size of media used. Each file takes up disk space in the same manner that a PC would, the disk space is equal to n * cluster size (where n is the size of the file divided by the cluster size, + 1 (if any partial cluster is used, else 0)).

example:

GIVEN:
sector size = 512 bytes (most flash media)
cluster size = 8 sectors = 4096 bytes (dependant on FAT type and media size)
file1 size = 1,000 bytes
file2 size = 12,288 bytes
file3 size = 100,000,000 bytes

FILE1 SIZE (ON DISK):
(file1 size) % (cluster size) = (1,000) % (4096) = *1,000 (Not 0)*
disk space used = ((file1 size) / (cluster size) + 1) * (cluster size)
= ((1,000) / (4096) + 1) * (4096)
= ((0) + 1) * (4096)
= 4096 bytes used on disk

FILE2 SIZE (ON DISK):
(file2 size) % (cluster size) = (12,288) % (4096) = *0*
disk space used = (file2 size)
= 12,288 bytes used on disk

FILE3 SIZE (ON DISK):
(file3 size) % (cluster size) = (100,000,000) % (4096) = *256 (Not 0)*
disk space used = ((file3 size) / (cluster size) + 1) * (cluster size)
= ((100,000,000) / (4096) + 1) * (4096)
= ((24,414) + 1) * (4096)
= 100,003,840 bytes used on disk

(NOTE: If the file size is exactly divisible by the cluster size, that file size is the same size that is used on the disk.)

This is how the file system MUST use disk space for it to be compatible with a PC system. Any other disk space management system would not allow you to view any of the data on a PC, that was written from the device.

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

12 Directory Support

The FlashFile is set up so that directories and sub-directories can be accessed when a file name or path is entered in a command. File commands (fopen, fcreate, remove, etc.) and directory commands (mkdir, chdir, rmdir, etc.) can be used to access files or folders by inputting the full path name or relative path name. A full path name string has a leading back slash ('\') to indicate that it is relative to the root directory; a relative path name string starts with a folder name, a ".." (indicating one directory back), or just the file name.

Note: When entering a backslash in a constant sring, the backslash must be preceded by a backslash to prevent the compiler from interpreting the backslash as a control character (i.e. '\\' = 0x5C, "\\\\" is = 0x5C, 0x5C).

Examples:

Given:

Folder Trees: ROOT\\FOLDER1\FOLDER2\FOLDER3\
ROOT\\LEVEL1\LEVEL2\LEVEL3\

Current Directory: ROOT\\FOLDER1\

Problem:

Create the file "TEST.TXT" in the ROOT\\LEVEL2\ directory

Solution 1:

```
chdirc("\\LEVEL1\\LEVEL2");          // change to the \LEVEL1\LEVEL2 directory
fcreate("TEST.TXT");                 // create "TEST.TXT" in the current directory
```

Solution 2:

```
chdirc("..\\LEVEL1\\LEVEL2");        // go back a folder, then change to
LEVEL1\LEVEL2 directory
fcreate("TEST.TXT");                 // create "TEST.TXT" in the current directory
```

Solution 3:

```
fcreate("\\LEVEL1\\LEVEL2\\TEST.TXT"); // create "TEST.TXT" in the directory
\LEVEL1\LEVEL2\
```

The FlashFile has some limitations on directory usage; if used beyond these limitations unexpected results may occur.

* Folder names must be accessed in the 8.3 Dos format, an 8 character name and a 3 character extension (always 3 spaces [0x20] for folders)

13 Real-Time Clock

The FlashFile is set up so that a real-time clock can be used to date/time stamp the creation and modification of files. The directory in which the file information is stored has the date/time information stored, and can be read/modified with a PC. The real-time clock option of the FlashFile can be turned off in the control block of the main header file. If the real-time clock is not in use, the date/time stamp for the file is incremented (so the user knows the file is newer than before) whenever the file is modified.

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

The "twi.c" file included with the FlashFile has all of the interface functions for a Philips PCF8563 real-time clock/calendar hooked up to the I^2C bus of the Atmel AVR processor (see the Priio Mega128-MMC development board for hookup information). Using a different real-time clock requires replacing the real-time clock routines in the twi.c file. The real-time clock function **int8 rtc_get_timeNdate(int8 *hour, int8 *min, int8 *sec, int8 *date, int8 *month, int16 *year)** is required by the FlashFile to use the real-time clock option. A search in all files for "_RTC_ON_" will help the user to locate where real-time clock functions are used in the files.

14 OPTIONS.H Header File

14.1 Control Block

All of the compiling options for the FlashFile project are defined in the "options.h" file of the project; all of the control and project compile options for the project. Since the FlashFile can take up a considerable amount of flash code space on the AVR processor, the compiling options enable the user to reduce the amount of flash code space used depending on the user's needs. Section A of the "options.h" file lists the compile options for the file.

- ***_RTC_ON_*** - this option enables the real-time clock options, time and date stamping files on creation and modification; if this feature is commented out, the time/date code on the file increments whenever modified.
- ***_SECOND_FAT_ON_*** - this option allows the system to update both FAT tables when writing files; if not in use, the system wipes out the second FAT table when creating or modifying a file so the system does not conflict with the PC.
- ***_FAT12_ON_*** - this option enables the FAT12 file system to be supported by the project; cards 64MB and under can come pre-formatted in FAT12, although all cards 32MB and above are formatted in FAT16 when formatted by a PC.
- ***_READ_ONLY_*** - this option allows users the option to significantly reduce their code space by eliminating all writing functions to the Secure Digital cards; any attempts to write to the card will error or be invalid.
- ***_DEBUG_ON_*** - this option enables an output to the serial port so that the status of the card can be displayed in HyperTerminal. This also must be on to use the read_directory() and GetVolID() functions.
- ***_DIRECTORIES_SUPPORTED_*** - this option enables directories to be

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

used.

- ***_NO_MALLOC_*** - this option is used if the user does not want to use a malloc() function. One FILE structure will automatically be allocated in global memory if this option is defined.
- ***_BYTES_PER_SEC_512_*** - In all current flash media, there are 512 bytes per sector, and there really is no need to have it as a variable. Defining *_BYTES_PER_SEC_512_* will hard code all references to BPB_BytsPerSec as 0x200 or 512. This will cut down on code space and speed the functions up a bit since *<< 9* and *>> 9* can replace ** 512* and */ 512*, and *& 0x1FF* can replace *% 512*.
- ***_CVAVR_***, ***_ICCAVR_***, & ***_ROWLEY_CWAVR_*** - Only ONE of these three options should be in the code, the other two should be commented out or unexpected results could occur. *_CVAVR_* if using the CodeVisionAVR IDE, *_ICCAVR_* if using the ImageCraftAVR IDE, and *_ROWLEY_CWAVR_* if using the Rowley CrossWorks for the AVR IDE.
- ***_LITTLE_ENDIAN_*** & ***_BIG_ENDIAN_*** - Only ONE of these two options should be defined in the code based on the type processor used. This will determine how unions will be defined in the code. All AVR processors are *_LITTLE_ENDIAN_*, and this option is only designed for those who wish to port the code to a different type of processor.
- ***_SD_MMC_MEDIA_*** - **S**hould be included if a Secure Digital or Multimedia Card is used.
- ***_FF_MAX_FILES_OPEN*** - This is the maximum number of files that can be opened at once (if *_NO_MALLOC_* is defined, if a malloc() function is used, this is ignored).
- ***_MEGA128NET_*** - this option sets up the SD/MMC SPI bus automatically for a Priio Mega128-Net board (else the SPI bus is setup for a Priio Mega128-MMC board)
- ***_SD_BLOCK_WRITE_*** (For SD/MMC cards only) - This enables block writing to the SD/MMC card.
- ***_FF_SPCR_SET*** - This is what the SPCR register will be set to, if different settings are needed (the default value is 0x50).
- ***SD_CS_ON()***, ***SD_CS_OFF()***, & ***CS_DDR_SET()*** (For SD/MMC cards only) - These are macros defined to be the I/O pin of the Atmel AVR processor that the chip select line of the SPI bus is connected to, so the microcontroller can control the SD card (mask bits to your needs).

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

- ***_FF_MAX_FPRINT*** - this determines the maximum length that a *fprintf()* string can be; the longer the string, the more that can be written in one line of an *fprintf()* command, however the memory requirements for the program will also be larger.
- ***_FF_PATH_LENGTH*** - this determines how long the full path (directory) or an entered string path can be (flash or RAM).

FlashFile "Includes"

Some of the functions in the FlashFileSD require the use of several standard C libraries. Section C of the "options.h" file includes the libraries required and #defines needed in some of the functions. (See "Memory Considerations" for use of printf in ICC if applicable). This is also where you could change which processor you are using. Since the "sd_cmd.c" file includes a functions that sets up the I/O of the processor, you would also have to change the #include <xxx128.h> to the appropriate file for your compiler and processor.

The actual FlashFile files "sd_cmd.h", "sd_cmd.c", "file_sys.h", "file_sys.c", "twi.h" (if applicable), and "twi.c" (if applicable) must also be included to use the file system functions.

```
/***************************** C HEADER FILE *********************************
**
**  Project:    FlashFile
**  Filename:   OPTIONS.H
**  Version:    3.0.4
**  Date:       July 8, 2008
**
*****************************************************************************
**
**  VERSION HISTORY:
**  ----------------
**  Version:    3.0
**  Date:       March 29, 2006
**  Revised by: Erick M. Higa
**  Description:
**      - See "FILE_SYS.C" file for any chages up to this point.
**
**  Version:    3.0.4
**  Date:       July 8, 2008
**  Revised by: David L. Brown
**  Description:
**      - Updated for newest version of compilers (CVAVR, ICC, CrossWorks).
**
*****************************************************************************/
#ifndef _OPTIONS_INCLUDED
#define _OPTIONS_INCLUDED

/*****************************************************************************
**
**    DEFINITIONS AND MACROS
**
*****************************************************************************/

/* Control Block */
#define _RTC_ON_
#define _SECOND_FAT_ON_
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

```
#define _FAT12_ON_
//#define _READ_ONLY_
#define _DEBUG_ON_
#define _DIRECTORIES_SUPPORTED_
#define _NO_MALLOC_
#define _BYTES_PER_SEC_512_

/* The settings below should be modified */
/* to match your hardware/software settings */
#define _CVAVR_
/*#define _ICCAVR_*/
/*#define _ROWLEY_CWAVR_*/

#define _LITTLE_ENDIAN_
//#define _BIG_ENDIAN_

#define _SD_MMC_MEDIA_

#ifdef _NO_MALLOC_
    #define _FF_MAX_FILES_OPEN    1
#endif

/*#define _MEGA128NET_*/
/*#define _MEGAAVRDEV_*/

#ifdef _DEBUG_ON_
    //#define _DEBUG_FUNCTIONS_
#endif

#if defined(_SD_MMC_MEDIA_)
    #ifndef _READ_ONLY_
        #define _SD_BLOCK_WRITE_
    #endif
    #define _FF_SPCR_SET    0x50

    #if defined(_MEGA128NET_)
        #define SD_CS_OFF()     PORTB |= 0x10
        #define SD_CS_ON()      PORTB &= 0xEF
        #define CS_DDR_SET()    DDRB |= 0x10
    #elif defined(_MEGAAVRDEV_)
        #define SD_CS_OFF()     PORTB |= 0x10
        #define SD_CS_ON()      PORTB &= 0xEF
        #define CS_DDR_SET()    DDRB |= 0x10
    #else
        #define SD_CS_OFF()     PORTD |= 0x40
        #define SD_CS_ON()      PORTD &= 0xBF
        #define CS_DDR_SET()    DDRD |= 0x40
    #endif
#endif

#define _FF_MAX_FPRINTF     75
#define _FF_PATH_LENGTH     50

#if defined(_CVAVR_)
    #define     _FF_SEI()       #asm("sei")
    #define     _FF_CLI()       #asm("cli")
    #define     _FF_NOP()       #asm("nop")

    #define     _FF_strcpyf     strcpyf
    #define     _FF_sprintf     sprintf
    #define     _FF_strlen      strlen
    #define     _FF_strncmp     strncmp
    #define         _FF_strstr              strstr
    #define         _FF_strstrf             strstrf
    #define                     _FF_strcat              strcat
    #define                     _FF_strcatf             strcatf
#elif defined(_ICCAVR_)
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

```
    #define     _FF_SEI         SEI
    #define     _FF_CLI         CLI
    #define     _FF_NOP         NOP

    #define     _FF_strcpyf     cstrcpy
    #define     _FF_sprintf     csprintf
    #define     _FF_strrchr     strrchr
    #define     _FF_strncmp     strncmp
    #define     _FF_strlen      strlen
    #define         _FF_strstr              strstr
    #define         _FF_strstrf             cstrstr
    #define                     _FF_strcat              strcat
    #define                     _FF_strcatf             strcat
#elif defined(_ROWLEY_CWAVR_)
    #define     flash           __code const
    #define     _FF_SEI         _SEI
    #define     _FF_CLI         _CLI
    #define     _FF_NOP         _NOP

    #define     _FF_strcpyf     strcpy_c
    #define     _FF_sprintf     sprintf_c
    #define     _FF_strrchr     strrchr
    #define     _FF_strncmp     strncmp
    #define     _FF_strlen      strlen
    #define         _FF_strstr              strstr
    #define         _FF_strstrf             strstrf
    #define                     _FF_strcat              strcat
    #define                     _FF_strcatf             strcatf
#endif



/*****************************************************************************
**
**    TYPEDEFS AND STRUCTURES
**
*****************************************************************************/
/*****************************************************************************
**
**    MODULES USED
**
*****************************************************************************/

#if defined(_CVAVR_)
    #ifdef _MEGAAVRDEV_
        #include <mega32.h>
    #else
        #include <mega128.h>
    #endif
#elif defined(_ICCAVR_)
    #include     <macros.h>
    #ifndef _MEGAAVRDEV_
        #include     <iom128v.h>
    #else
        #include     <iom32v.h>
    #endif
#elif defined(_ROWLEY_CWAVR_)
    #include    <__cross_studio_io.h>
    #include    <ina90.h>
    #include    <pgmspace.h>
    #include    <stdio_c.h>
    #ifndef _MEGAAVRDEV_
        #include    <atmega128.h>
    #else
        #include    <atmega32.h>
    #endif
#elif defined(_IAR_EWAVR_)
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

```
    #include     <pgmspace.h>
    #ifndef _MEGAAVRDEV_
        #include     <iom128.h>
    #else
        #include     <iom32.h>
    #endif
#endif

#include <stdarg.h>

#ifndef _NO_MALLOC_
    #include <stdlib.h>
#endif

#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "..\flash\stddefs.h"

#if defined(_SD_MMC_MEDIA_) && !defined(_SD_CMD_INCLUDED)
    #include "..\flash\sd_cmd.h"
#endif
#if !defined(_FILE_SYS_INCLUDED)
    #include "..\flash\file_sys.h"
#endif
#if defined(_RTC_ON_) && !defined(_TWI_INCLUDED)
    #include "..\flash\twi.h"
#endif

#endif  /*_OPTIONS_INCLUDED*/

/*****************************************************************************
**
**    EXPORTED VARIABLES
**
*****************************************************************************/
#if defined(_ICCAVR_) && !defined(_NO_MALLOC_)
    extern int8 _bss_end;
#endif

/*****************************************************************************
**
** GLOBAL VARIABLES
**
*****************************************************************************/
/*****************************************************************************
**
**    EXPORTED FUNCTIONS
**
*****************************************************************************/
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

15 FlashFile Function Calls

The functions below are provided by the FlashFile library and are included in the *"sd_cmd.c"*, and the *"file_sys.c"* files. These functions are used to communicate to the Secure Digital card, and organize the data in the files in a FAT12/16 format. All function explanations and examples assume the *"options.h"* file has already been included and variable type declarations have been defined as above.

15.1 **uint8** initialize_media(**void**)

This function is required to initialize the Flash card so reading and writing to the card is possible. This function also reads the card's partition table and boot sector, storing the card information needed to handle the file system. No data can be read from or written to the card without this function being run first. This function returns a *1* if the card is successfully initialized, and a *0* if the initialization fails.

Example:

```
    if (initialize_media())     /* If the media is initialized ok, */
        putsf("OK");            /* output "OK" */
    else                                    /* Otherwise */
        putsf("ERROR");                     /* output "ERROR" */
```

int16 fquickformat(void)

This function is used to do a quick format of the Flash media. When called, all data on the current media will be destroyed. If the function is successful a '0' is returned, if the function fails an EOF is returned.

Example:

```
    if (initialize_media())     /* If the media is initialized ok, */
        fquickformat();                     /* Delete all information on the card
    */
```

15.2 **void** GetVollD(**void**)

This function prints the Flash media's serial number and volume label to the serial port (UART/USART must be properly set up) of the processor. (Note: The volume serial number and label can also be accessed through the global variables located in the "sd_cmd.c" file once "initialize_media()" is called. The serial number is stored as the unsigned long *BP_VolID*, and the volume label is stored in the character string *BP_VolLabel[12]*)

Example:

```
    if (initialize_media())     /* If the media is initialized ok, */
        GetVolID();             /* Send Volume information of card to serial port
    */

    Terminal Output
      Volume Serial:  [0x8D8293D]
      Volume Label:  [SD64CARD   ]
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

priio™
developers of intelligent, interactive products

15.3 **void** read_directory(**void**)

This function outputs the list of files located in the current directory of the Flash card to the serial port (UART/USART must be properly set up) so the user can see what files (and their size in bytes) are available for reading and writing to the card. This function also shows the working directory string of the project relative to the root directory. This function uses the printf defined in the Standard Input/Output Library, and may take up a large portion of code space if not used anywhere else. This function is conditionally compiled when the *#define _DEBUG_ON_* is not commented out in the "options.h" file. This function does not return any information other then the data output to the serial port.

Example:

```
if (initialize_media())     /* If the media is initialized ok, */
    read_directory();                   /* Send file list of card to serial
port */

Terminal Output (Assume TEST is a directory, TEST1.TXT and TEST2.TXT are
files):
    File Listing for:  ROOT\\
      [TEST]
      TEST1.TXT             [131289]  bytes
      TEST2.TXT             [229327]  bytes
```

15.4 **int16** fget_file_info(**int8 ***fname, **int32 ***fsize, **int8** *create_date, **int8 ***mod_date, **uint8 ***attribute, **uint16 ***fclus)

This function stores the information of the file designated by the character string **NAME*, into strings designated by the pointers **fsize* for file size, **create_date* for create time/date, **mod_date* for last modified time/date, **attribute* for the attribute byte, and **fclus* for the starting cluster of the file. The function returns a *'0'* if the variables are successfully updated, or an *EOF* if an error occurs (i.e. Invalid path or filename, read error).

Example:

```
int8 filename[12];                      /* String to save file name */
uint32 filesize;            /* Variable to save file size in */
int8 create_date_str[20];   /* String to save create time/date */
int8 modify_date_str[20];   /* String to save last modified time/date */
uint8 file_attr;            /* Variable to save file attribute byte to */
uint16 file_clus_start;     /* Variable to save starting cluster of file to */

strcpyf(filename, "TESTFILE.TXT");      /* Save file name to string

// Save file attributes for "TESTFILE.TXT" to variables
fget_file_info(filename, &filesize, create_date_str, modify_date_str,
              &file_attr, &file_clus_start);
```

15.5 **int16** fget_file_infoc(**int8 flash ***fname, **uint32 ***fsize, **int8** *create_date, **int8 ***mod_date, **uint8 ***attribute, **uint16**

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

*****fclus)

This function is the same as "fget_file_info", except that a constant strings can be used to directly open a file whose name is located in the in codeflash data area.

Example:

```
uint32 filesize;            /* Variable to save file size in */
int8 create_date_str[20];   /* String to save create time/date */
int8 modify_date_str[20];   /* String to save last modified time/date */
uint8 file_attr;            /* Variable to save file attribute byte to */
uint16 file_clus_start;     /* Variable to save starting cluster of file to */

// Save file attributes for "TESTFILE.TXT" to variables
fget_file_infoc("TESTFILE.TXT", &filesize, create_date_str, modify_date_str,
             &file_attr, &file_clus_start);
```

15.6 **int16** chdir(**int8 ***f_path)

This function changes the working directory to the directory designated by the character string **f_path*. If the string starts with a '\', the path is relative to the root directory; a '\' in the middle of the path string indicates sub-directories. A ".." within the string (between or before a '\') indicates the previous directory relative to the current directory. This function returns a *'0'* if the working directory is successfully changed, or an *EOF* if the working directory could not be changed (i.e. Folder does not exist, read failure, or path does not exist).

Example:

```
if (initialize_media())         /* If the media is initialized ok, */
    read_directory();                       /* Send file list of card to
serial port */
strcpyf(foldername, "TESTFOLD");  /* Save folder name to string */
flag = chdir(foldername);       /* Change the working directory to [TESTFOLD]
*/
if (flag==0)
    read_directory();                       /* Send file list of card to
serial port */

Terminal Output (Assume TESTFOLD is a directory):
    File Listing for:  ROOT\\
      [TESTFOLD]
      TEST1.TXT             [131289]  bytes
      TEST2.TXT             [229327]  bytes

    File Listing for:  ROOT\\TESTFOLD\
      [TESTDIR]
      TEST3.TXT             [32623]  bytes
      TEST4.TXT             [43974]  bytes
```

15.7 **int16** chdirc(**int8 flash ***f_path)

This function is the same as "chdir", except that a constant strings can be used to directly change directories to a directory whose name is located in the in codeflash data area.

Example:

```
if (initialize_media())              /* If the media is initialized ok, */
    read_directory();                /* Send file list of card to serial port
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

```
*/
flag = chdirc("\\TESTFOLD\\TESTDIR");  /* This call changes the working
directory to */
                                       /* [\TESTFOLD\TESTDIR] */
if (flag==0)
    read_directory();                  /* Send file list of card to serial port
*/

Terminal Output (Assume TESTFOLD is a directory):
    File Listing for:  ROOT\\
      [TESTFOLD]
      TEST1.TXT             [131289]  bytes
      TEST2.TXT             [229327]  bytes

    File Listing for:  ROOT\\TESTFOLD\TESTDIR\
      FILE1.TXT             [252684]  bytes
      FILE2.TXT             [185201]  bytes
```

15.8 FILE *fopen(int8 *fname, uint8 fmode)

This function opens the file designated by the character string **fname*, and associates it to an input and/or output stream based on the *fmode* handle. The function allocates the required memory to handle files, then returns a FILE pointer to the allocated space. If there is an error allocating the memory or opening the file, a NULL pointer is returned. (Note: Filenames used must be in an 8.3 DOS format, long filenames are not currently supported)

The *fmode*'s can be *READ*, *WRITE*, or *APPEND:*

* *READ* – Opens a file from the beginning, to read only
* *WRITE* – Destroys contents of a file and opens it from the beginning for writing
* *APPEND* – Opens a file to the end for writing

Example:

```
strcpyf(filename, "TEST.TXT");    /* Save file name to string */
fp = fopen(filename, READ);       /* This call opens the file "TEST.TXT" in
                                  the current */
                                  /* directory for reading, referenced by the
                                  type FILE */
                                  /* pointer *fp */
```

15.9 FILE *fopenc(int8 flash *fname, uint8 fmode)

This function is the same as "fopen", except that a constant string is used to directly open a file whose name is located in the in codeflash data area.

Example:

```
fp = fopenc("\\FOLD\\TEST.TXT", READ);       /* This call opens the file
                                              "TEST.TXT" in  */  /* the
                                              directory [FOLD] for reading */
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

15.10 **int16** mkdir(**int8 ***f_path)

This function creates a directory designated by the character string **f_path.* This function returns a *'0'* if the folder is successfully created, and an *EOF* if the folder could not be created (i.e. Folder with same name already exists, write to media failure, or media is full).

Example:

```
if (initialize_media())         /* If the media is initialized ok, */
    read_directory();                       /* Send file list of card to
serial port */
// Save file name to string
strcpyf(foldname, "TESTFOLD");    /* Save folder name to string */
flag = mkdir(foldname);         /* This call creates the folder "TESTFOLD" */
read_directory();                       /* Send file list of card to
serial port */

Terminal Output (Assume TEST is a directory, TEST1.TXT and TEST2.TXT are
files):
    File Listing for:  ROOT\\
      [TEST]
      TEST1.TXT             [131289]  bytes
      TEST2.TXT             [229327]  bytes

    File Listing for:  ROOT\\
      [TEST]
      TEST1.TXT             [131289]  bytes
      TEST2.TXT             [229327]  bytes
      [TESTFOLD]
```

15.11 ***FILE** fcreate(**int8 ***fname, **uint8** fmode)

This function creates a file of size 0 bytes, designated by the character string **fname*, with file attributes *fmode*, then opens the file in WRITE mode. The *fmode* switch designates what will be written to the "attribute" byte of the file (the "archive" bit is always assumed to be set). Multiple *fmode*s can be used by inserting a bitwise "OR" symbol '|' between the different attributes being set. If a file with the same name already exists, it over writes the file to a blank file, then opens the file in WRITE mode (unless the existing file is "read only"). This function returns a *FILE* pointer to the opened FILE structure if the file is successfully created, a *NULL* pointer is returned if the file could not be created (i.e. File with same name already exists as "READ_ONLY", write to media failure, or media is full).

The *fmode*(s) can be *ATTR_READ_ONLY*, *ATTR_HIDDEN*, and/or *ATTR_SYSTEM*:

* *ATTR_READ_ONLY* – Once the file is closed, it cannot be modified with this attribute
* *ATTR_HIDDEN* – The file will not be visible to the user
* *ATTR_SYSTEM* – The file will be marked as a system file

Example:

```
if (initialize_media())         /* If the media is initialized ok, */
    read_directory();                       /* Send file list of card to
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

```
serial port */
strcpyf(filename, “TEST.TXT”);    /* Save file name to string */
file1 = fcreate(filename, 0);              /* This call creates the file
“TEST.TXT” */
fclose(file1);                             /* Close the file */
read_directory();                          /* Send file list of card to
serial port */

Terminal Output (Assume TEST is a directory, TEST1.TXT and TEST2.TXT are
files):
    File Listing for:  ROOT\\
      [TEST]
      TEST1.TXT             [131289]  bytes
      TEST2.TXT             [229327]  bytes

    File Listing for:  ROOT\\
      [TEST]
      TEST1.TXT             [131289]  bytes
      TEST2.TXT             [229327]  bytes
      TEST.TXT              [0]  bytes
```

15.12 *FILE fcreatec(int8 flash *fname, uint8 fmode)

This function is the same as “fcreate”, except that a constant string is used to directly create a file whose name is located in the in codeflash data area.

Example:

```
file = fcreatec(“TEST.TXT”, 0);        /* This call creates and opens the
file “TEST.TXT” */
```

15.13 int16 rmdir(int8 *f_path)

This function deletes the directory designated by the character string **f_path.* This function returns a *‘0’* if the folder is successfully removed, and an *EOF* if the folder could not be deleted (i.e. Folder does not exist, folder is not empty, write to media failure, or a read only file).

Example:

```
strcpyf(foldername, “TESTFOLD”);       /* Save folder name to string */
flag = rmdir(foldername);              /* This call deletes the folder
[TESTFOLD] */
```

15.14 int16 remove(int8 *fname)

This function deletes the file designated by the character string **fname.* This function returns a *‘0’* if the file is successfully removed, and an *EOF* if the file could not be deleted (i.e. File does not exist, write to media failure, or a read only file).

Example:

```
strcpyf(filename, “TEST.TXT”);         /* Save file name to string */
flag = remove(filename);               /* This call deletes the file
“TEST.TXT” */
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

15.15 int16 removec(int8 flash *fname)

This function is the same as "remove", except that a constant strings can be used to directly delete a file whose name is located in the in codeflash data area.

Example:

```
flag = remove("TEST.TXT");          /* This call deletes the file
"TEST.TXT" */
```

15.16 int16 rename(int8 *name_old, int8 *name_new)

This function renames one file designated by the character string **name_old*, to be designated by the character string **name_new*. This function returns a *'0'* if the file is successfully renamed, and an *EOF* if the file could not be renamed (i.e. File does not exist, File with new name already exists, Write to media failure, Read only file). If using full directory addressing, the *name_old* string can have the full path length, where the *name_new* has just the new name of the file.

Example:

```
// Save file names to strings
strcpyf(filename1, "FOLDER\\TEST1.TXT");          /* Save file name to string
*/
strcpyf(filename2, "TEST2.TXT");                  /* Save file name to string
*/
flag = rename(filename1, filename2);         /* This call renames the file
                                             "TEST1.TXT" */     /* located in
                                             the directory [FOLDER] to */
                                             /* "TEST2.TXT" */
```

15.17 int16 fclose(FILE *rp)

This function saves the data (if applicable) and closes the file designated by *FILE* pointer **rp* and frees the associated memory allocated for the file. This function returns a *'0'* if the file is closed, and an *EOF* if an error occurred while closing the file (i.e. *FILE* pointer is *NULL*).

Example:

```
fp = fopenc("TEST.TXT", APPEND);       /* open file "TEST.TXT" to append */
do_stuff(fp);                                  /* do stuff to file */
flag = fclose(fp);                             /* save and close file */
```

15.18 int16 fflush(FILE *rp)

This function writes all buffered data of the file designated by *FILE* pointer **rp* to the Secure Digital card. This function returns a *'0'* if the file is saved successfully, and an *EOF* if an error occurred while saving the file (i.e. *FILE* pointer is *NULL* or file is read only).

Example:

```
fp = fopenc("TEST.TXT", APPEND);       /* open file "TEST.TXT" to append */
do_stuff(fp);                                  /* do stuff to file */
flag = fflush(fp);                             /* save file w/o closing */
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

15.19 **int16** ffreemem(**FILE ***rp)*

This function releases the associated memory allocated for the file designated by FILE pointer *rp. This function returns a *'0'* if the memory is freed successfully, and an *EOF* if an error occurred while saving the file (i.e. *FILE* pointer is *NULL*).

Example:

```
fp = fopenc("TEST.TXT", APPEND);        /* open file "TEST.TXT" to append */
do_stuff(fp);                                   /* do stuff to file */
flag = ffreemem(fp);                            /* Free memory associated
with FILE pointer *fp */
```

15.20 **int32** ftell(**FILE ***rp)

This function returns the current position (relative to the beginning of the file) of the data pointer for the file designated by *FILE* pointer **rp.* This function returns an *EOF* if an error occurred while finding the position (i.e. *FILE* pointer is NULL or a read failure).

Example:

```
fp = fopenc("TEST.TXT", APPEND);        /* open file "TEST.TXT" to append */
do_stuff(fp);                                   /* do stuff to file */
location = ftell(fp);                           /* find the position within
data of the file */
```

15.21 **int16** fseek(**FILE ***rp, **uint32** off_set, **uint8** fmode)

This function moves the data pointer associated with the file designated by *FILE* pointer *rp. The position the data pointer is moved to is based on the *off_set* and *fmode* handles. This function returns a *'0'* if the data pointer is successfully moved, and an *EOF* if an error occurs during the positioning of the data pointer in the file (i.e. The *off_set* relative to the *fmode* is outside the range of the file or a read failure).

The *mode*'s can be *SEEK_CUR*, *SEEK_END*, or *SEEK_SET:*

* *SEEK_CUR* – Positions by *off_set* relative to the current data pointer location
* *SEEK_END* – Positions by *off_set* relative to the end of the file
* *SEEK_SET* – Positions by *off_set* relative to the beginning of the file

Example:

```
fp = fopenc("TEST.TXT", APPEND);        /* open file "TEST.TXT" to append */
do_stuff(fp);                                   /* do stuff to file */
location = ftell(fp);                           /* find the position within
data of the file */
do_stuff(fp);                                   /* do stuff to file */
flag = fseek(fp, location, SEEK_SET);   /* returns data pointer to "location"
*/
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

15.22 **int16** fgetc(**FILE ***rp)

This function returns the character at the location the data pointer is pointing to in the file designated by *FILE* pointer **rp*, then increments the data pointer. This function returns an *EOF* if an error occurs (i.e. *NULL* file pointer or a read failure).

Example:

```
fp = fopenc("TEST.TXT", READ);          /* open file "TEST.TXT" to read */
putchar(fgetc(fp));                                 /* output first data byte,
increment file pointer */
```

15.23 **int16** fread(**void ***rd_buff, **int16** dat_size, **int16** num_items, **FILE ***rp)

This function reads *num_items* of size *dat_size* bytes from the file designated by *FILE* pointer **rp*, into the buffer designated by the pointer **rd_buff*, from the location the data pointer is pointing to. This function returns the number of successfully read items, less the number of errors.

Example:

```
uint16 temp_buff[10];
fp = fopenc("TEST.TXT", READ);          /* open file "TEST.TXT" to read */
if(fread(temp_buff, 2, 10, fp) != 10)   /* read 10 Words of data from the file
*/
    return(EOF);                        /* return an error if not 10 items */
```

15.24 **int8 ***fgets(**int8 ***buffer, **int16** n, **FILE ***rp)

This function returns a character pointer to the string designated by the pointer **buffer*, filling the string with data from the file designated by *FILE* pointer **rp*, up to *n* characters or a line feed ('\n'). This function returns an *EOF* if an error occurs (i.e. *NULL* file pointer or a read failure).

Example:

```
int8 data_buff[20];                     /* read data buffer */
int8 *pntr;                             /* Data pointer */
fp = fopenc("TEST.TXT", READ);          /* open file "TEST.TXT" to read */
pntr = fgets(data_buff, 20-1, fp);      /* save first line of data to the
string data_buff */
```

15.25 **int16** ungetc(**uint8** file_data, **FILE ***rp)

This function decrements the data pointer in the file designated by *FILE* pointer **rp*, then pushes the character *file_data* back to the file. This function returns the character *file_data* if successful, or it returns an *EOF* if an error occurs (i.e. *NULL* file pointer or a read failure).

Example:

```
fp = fopenc("TEST.TXT", READ);          /* open file "TEST.TXT" to read */
flag = fseek(fp, 0, SEEK_END);          /* move the data pointer to the end of
the file */
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

```
putchar(ungetc('T', fp));              /* Put a 'T' in the last data byte,
then -- pntr */
```

15.26 int16 fputc(uint8 file_data, FILE *rp)

This function inserts the character *file_data*, in the file designated by *FILE* pointer **rp*, at the location the data pointer is pointing to, then increments the data pointer. If the character to be written is at the start of a new sector, this function will first flush and save the data buffer before writing the next character to the file. This function returns the value of the character *file_data* if the input was successful, or an *EOF* if an error occurs (i.e. *NULL* file pointer, write failure, or file in read-only mode).

Example:

```
fp = fopenc("TEST.TXT", APPEND);       /* open file "TEST.TXT" to the end in
append mode */
fputc('S', fp);                        /* write an 'S' to the file, increment
file pointer */
```

15.27 int16 fwrite(void *wr_buff, int16 dat_size, int16 num_items, FILE *rp)

This function inserts *num_items* of size *dat_size* bytes from the buffer designated by the pointer **wr_buff*, into the file designated by *FILE* pointer **rp*, at the location the data pointer is pointing to. If the string to be written crosses sectors or is at the start of a new sector, this function will first flush and save the data buffer of the previous sector before filling the next write buffer of the file. This function returns the number of successfully written items, less the number of errors.

Example:

```
int16 temp_buff[10];
int16 i;
for(i = 0; i < 10; i++)
    temp_buff[i] = (i * 0x200) + i;    /* Fill buffer with data */
fp = fopenc("TEST.TXT", APPEND);       /* open file "TEST.TXT" to the end in
append mode */
if(fwrite(temp_buff, 2, 10, fp) != 10)/* add the 10 Words of data to the end
of the file */
    return(EOF);                       /* return an error if not 10 */
```

15.28 int16 fputs(int8 *file_data, FILE *rp)

This function inserts the string designated by the pointer **file_data*, in the file designated by *FILE* pointer **rp*, at the location the data pointer is pointing to, followed by a line feed (0x0A) and carriage return (0x0D). If the string to be written crosses sectors or is at the start of a new sector, this function will first flush and save the data buffer of the previous sector before filling the next write buffer of the file. This function returns a *'0'* if the input string was successfully

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

added to the file, or an *EOF* if an error occurs (i.e. *NULL* file pointer, write failure, or file in read-only mode).

Example:
```
strcpyf(inputstring, "Hello World!");   /* Save string to SRAM */
fp = fopenc("TEST.TXT", APPEND);        /* open file "TEST.TXT" to the end in
append mode */
fputs(inputstring, fp);                 /* add "Hello World!\r\n" to the end
of the file */
```

15.29 int16 fputsc(int8 flash *file_data, FILE *rp)

This function is the same as "fputs", except that a constant string located in the in codeflash data area can be used as a direct input to the file designated by the *FILE* pointer **rp*.

Example:
```
fp = fopenc("TEST.TXT", APPEND);        /* open file "TEST.TXT" to the end in
append mode */
fputsc("Hello World!", fp);             /* add "Hello World!\r\n" to the end
of the file */
```

15.30 void fprintf(FILE *rp, int8 flash *pstr, ...)

This function inserts the constant string **pstr*, at the location the data pointer is pointing to, in the file designated by *FILE* pointer **rp*, with the same options that printf supports (i.e. %x, %lx, %X, %lX, %d, %ld, etc.).

Example:
```
fp = fopenc("TEST.TXT", APPEND);        /* open file "TEST.TXT" to the end in
append mode */
sum = 3 + 4;                                    /* assign sum the value of
7 */
fprintf(fp, "The number %d is the answer", sum);    /* output the string to the
file */
    /* "The number 7 is the answer" will be written to the file incrementing
the data */
    /* pointer as the string is written */
```

Note: The maximum length of both the input string and the formatted string is defined in the "options.h" file as _FF_MAX_FPRINT.

15.31 int16 feof (FILE *rp)

This function indicates whether the data pointer is at the end of the file designated by *FILE* pointer **rp*. This function returns a *1* if the data pointer is at the end of the file, a *0* if the data pointer is not at the end of the file, or an *EOF* if an error occurs (i.e. *NULL* file pointer).

Example:
```
fp = fopenc("TEST.TXT", READ);          /* open file "TEST.TXT" to read */
while(feof(fp) == 0)                            /* output file stream if
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

```
    not at the end or error */
        putchar(fgetc(fp));
```

15.32 **int16** feof (**FILE ***rp)

This function indicates whether the data pointer is at the end of the file designated by *FILE* pointer **rp*. This function returns a *1* if the data pointer is at the end of the file, a *0* if the data pointer is not at the end of the file, or an *EOF* if an error occurs (i.e. *NULL* file pointer).

Example:

```
    fp = fopenc("TEST.TXT", READ);          /* open file "TEST.TXT" to read */
    while(feof(fp) == 0)                                /* output file stream if
    not at the end or error */
        putchar(fgetc(fp));
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

16 FlashFile SD Block Write Function Calls

The functions below are provided by the FlashFile library and are included in the *"sd_cmd.c"* and the *"file_sys.c"* files. These functions are used to write a full continuous file on an SD/MMC card. Use of the Block Write functions assume *_SD_BLOCK_WRITE_* is defined. The "run_stream_demo()" function in the "fdemo.c" file has a full example on how to effectively use the block writing functions to its fullest capabilities.

Note: The Block Write functions must **not** be used with the other FlashFile functions. The block writing can only be done if the file size is set when the file is created. A FILE structure is required to store file information, however the FILE structure's file buffer is not used to write any data, so the file data pointer cannot be repositioned and the file data cannot be modified until the file is done writing. This also means that block writing cannot be used to append a file, only creating a new file can utilize the block writing. Using block writing in ways not described here will produce unexpected results, and can destroy other data on your media.

16.1 **int16** fstream_file(**int8 ***fname, **uint32** fsize)

This function **must** be called first when using the block writing. This will create an non-fragmented file of size *fsize* and named the character string designated by **fname*. This also starts the block write, and once called, the *SD_write_block_byte()* function must be used to write data to the file, and the *SD_write_block_end()* function must be called when done. This function returns a *0* if the file was created and the start of the clock write was successful, an *EOF* is returned if the file could not be created.

16.2 **int8** SD_write_block_byte (**uint8** sd_data)

This function is used to write one byte of data within a block write. The global variable *SDBlockWriteBlockCnt* indicates how many bytes have been written since the *fstream_file()* was called to start the block write. This function returns a *0* if the data was written successfully, or an *EOF* if an error occurs.

Note: Verify when using *SD_write_block_byte()* that *SDBlockWriteBlockCnt* is not greater than the file size that was reserved. Using this function once *SDBlockWriteBlockCnt* reaches the file size limit will write data to the card at the next sector regardless of what is there, possibly overwriting other data. Any data written once *SDBlockWriteBlockCnt* reaches the file size limit will also not be accessible using the FlashFile or other FAT file systems.

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

priio™
developers of intelligent, interactive products

16.3 int8 SD_write_block_end(void)

This function is used to end a block write session. This function returns a *0* if the data was written successfully, or an *EOF* if an error occurs.

Simple Example:

```
strcpyf(filename, "TEST.DAT");                    /* Save file name to string
*/
if(fstream_file(filename, 1048576))               /* Make "TEST.DAT" a 1MB
contiguous file */
    return;     /* Error, Return */
while(_SDBlockWriteBlockCnt < 1048576)
{
    /* SDBlockWriteBlockCnt is position in the file that the block write is
using */
    _SD_write_block_byte(c++);                    /* Write data */
}
_SD_write_block_end();
```

17 Example FlashFile Application

The example application program for the FlashFile ("demo.prj" included in the package) is designed to create the data file "demo.dat" on a preformatted SD/MMC card (FAT12 or FAT16). If no card exists (or the card is not formatted correctly) PORTB.6 toggles until an initialization of a card is successful. If the file already exists, the program deletes the existing data in that file and replaces it with the new data. The first line of data will be the Date/Time stamp of the file, followed by a line of column headers, then each line of data. A comma between each column and a line feed/carriage return for each row separates the data so the file can be directly imported into a spreadsheet program like MS Excel. The SD/MMC example application is designed to run directly on the Priio Mega128-MMC development board.

Note: The included demo project is pre-setup to integrate the media referenced (SD/MMC or CF) in the ordered package. If both the SD/MMC and CF packages were purchased, the media used depends on the "options.h" file as described in the "OPTIONS.H Header File" section of this document.

17.1 Including the FlashFile

The FlashFile files must be included in the project to use any of the communications and/or file system functions for the preferred flash card. At the top of the main file, declaring the options header enables the use of any of the functions. Any information that needs to be accessed from the main source file must also be declared (i.e. global variables from other files' function calls).

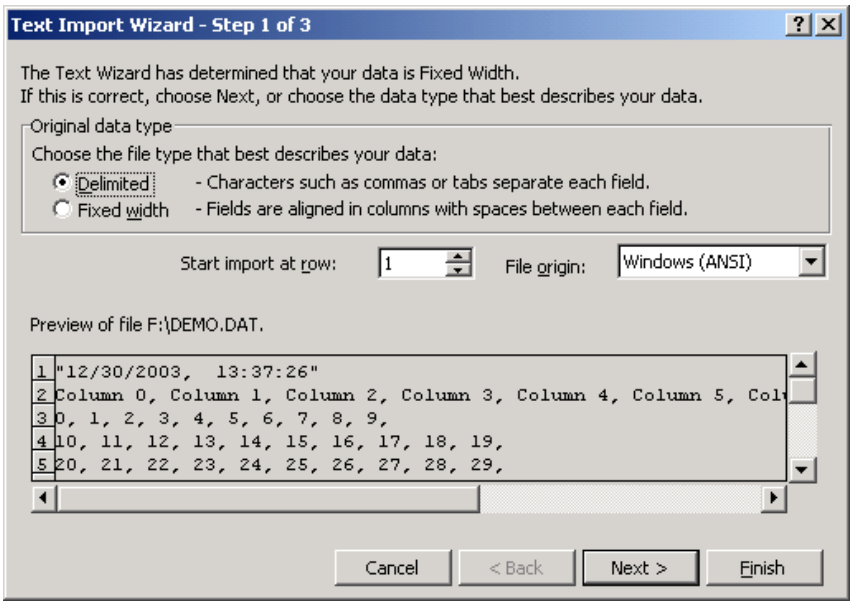
```
#include "options.h"

// Declare your global variables here
```

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

```
#ifdef _ICCAVR_
    extern char _bss_end;
#endif
```

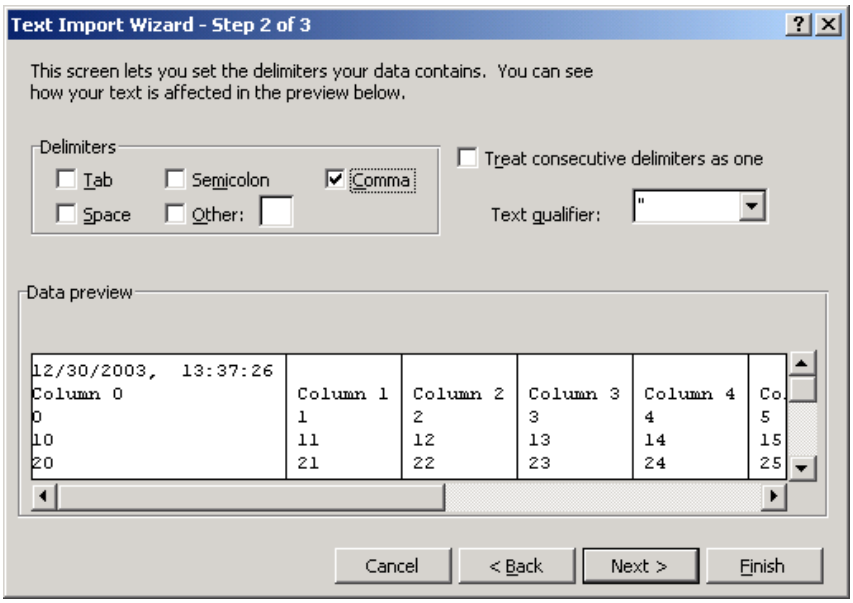
17.2 Importing the Data

Once the file is created, processing the document into a spreadsheet program like MS Excel is simple. The program that will be used to view the file must be opened, then select "file => open" and select "All Files (*.*)" under "Files of Type", then select the desired file to be opened (in this case the file to be opened will be "DEMO.DAT"). When opening an non “*.xls” file, Excel should automatically start the Import Wizard to see how the data is separated. If the FlashFile is used properly when gathering data, a delimiter will be used to separate the data (a comma is used in the demo), and the import wizard will put all of the data into a properly formatted rows and columns. An example of importing the data into MS Excel is listed below:



MS Excel - Import Wizard: Select "Delimited"

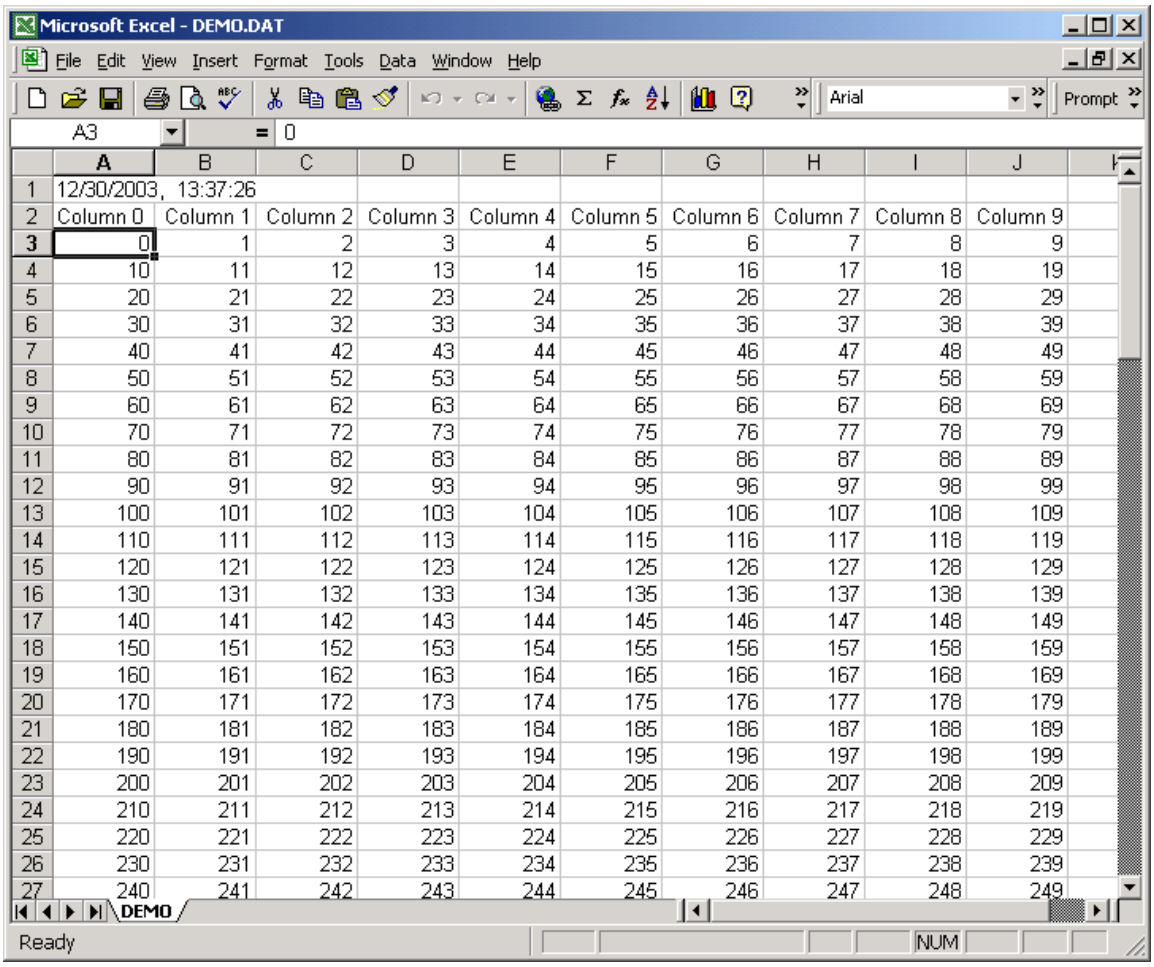
Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com



MS Excel - Import Wizard: Deselect "Tab" - Select "Comma"

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

priio™
developers of intelligent, interactive products



MS Excel – Open file "DEMO.DAT"

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

priio™
developers of intelligent, interactive products

Source Code Solution Software
License Agreement

IMPORTANT: READ THIS AGREEMENT CAREFULLY.

BY SELECTING THE "I ACCEPT THE AGREEMENT" OPTION AND CLICKING ON THE "NEXT" BUTTON, OR BY INSTALLING, COPYING, RUNNING, OR OTHERWISE USING THE PRIIO SOFTWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE AGREEMENT, PLEASE CLICK THE "CANCEL" BUTTON, AND DO NOT INSTALL, RUN, COPY, OR OTHERWISE USE THE PRIIO SOFTWARE.

This End User License Agreement ("License") is a legal agreement between you and Progressive Resources LLC DBA **Priio**, 8645 Guion Rd, Suite A, Indianapolis, IN 46268, USA ("Priio") concerning your use of the Priio Software, together with any electronic documentation that may be provided therewith (collectively, "the software") through the Software. YOU HEREBY AGREE, BOTH ON YOUR OWN BEHALF AND AS AN AUTHORIZED REPERSENTATIVE OF ANY ORGANIZATION FOR WHICH YOU ARE USING THE SOFTWARE ("EMPLOYER"), THAT YOU AND THE EMPLOYER WILL USE THE SOFTWARE ONLY IN ACCORDANCE WITH THE FOLLOWING TERMS:

1. **Disclaimer of Warranty.** You expressly acknowledge and agree that use of the Software is at your sole risk. THE SOFTWARE IS PROVIDED"AS IS" WITH ALL FAULTS AND WITHOUT WARRANTY OF ANY KIND, PRIIO does not warrant that the functions contained in the Software will meet your requirements or those of the Employer, or that the operation of the Software will be uninterrupted or error-free, or that defects in the Software will be corrected. Furthermore, Priio does not warrant or make any representation regarding the use or the results of the use of the Software (including the related documentation) in terms of their correctness, accuracy, reliability, or otherwise. Should the Software prove defective, you (and not Priio) assume the entire cost of all necessary servicing, repair, or correction.

**PRIIO EXPRESSLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, OR NONINFRINGEMENT WITH RESPECT TO THE SOFTWARE. **

The Software may be provided with third-party plug-ins or other third-party software, or this Software may be provided as a plug-in for or otherwise in association with third-party software. Use of any such third-party software will be governed by the applicable license agreement, if any, with such third party.

***PRIIO IS NOT RESPONSIBLE FOR ANY THIRD-PARTY SOFTWARE AND WILL HAVE NO LIABILITY OF ANY KIND FOR YOUR USE OF SUCH THIRD-PARTY SOFTWARE AND MAKES NO WARRANTY OF ANY KIND WITH RESPECT TO SUCH THIRD-PARTY SOFTWARE. *

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSIONS MIGHT NOT APPLY TO

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

YOU.

2. **Limitation of Liability.** In no event will Priio's total liability for all damages exceed the amount of fifty dollars ($50.00).

UNDER NO CIRCUMSTANCES, INCLUDING NEGLIGENCE, WILL PRIIO BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST DATA, LOST REVENUE, OR LOST PROFITS, ARISING OUT OF OR RELATING TO THIS LICENSE OR THE SOFTWARE.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSTION OR LIMITATION OF CONSEQUENTIAL OR INDIRECT DAMAGES, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

3. **License Grant.** Priio grants to you, and you accept, a personal, nonexclusive, nontransferable license to install and use the Software in source code format as a single site. You may not transfer, share or distribute the Software in any reusable form, without the express written permission of Priio. You may embed and deploy the executable forms of this Software with your product without royalty. You may make one copy of the Software in human-readable form for backup purposes only. The backup copy must include all copyright and license information contained on the original. This License is effective until terminated as provided below. You may terminate this License by destroying the Software and any copies of the Software in your possession. This License will terminate automatically upon any violation of its terms by you or the Employer. You acknowledge that this License does not entitle you to any support, maintenance, or upgrade from Priio.

4. **License Restrictions.** You may not do any of the following yourself, or through any third party, and you may not permit any third party with whom you have a business relationship to do any of the following: (A) copy the Software, except as expressly set forth in paragraph 3 above; (B) sell, license, sublicense, lease, or rent, to any third party, whether for profit or without charge, any portion of the Software, or, in particular, without limiting the generality of the foregoing, distribute the Software on any media; make the Software accessible to the public or third parties, whether over networks, electronic bulletin boards, website, or otherwise; or allow any third to use the Software except for purpose of your internal business; (C) publish or otherwise communicate any review of or information about the performance of the Software to any third party without the prior written consent of Priio; (D) export, re-export, download, or otherwise use the Software in violation of any laws or regulations, including U.S. Department of Commerce Export Administration regulations and other applicable laws; or (E) use the Software in connection with life support systems, human implantation, medical devices, nuclear facilities, nuclear systems or weapons, aviation, mass transit, or any application where failure or malfunction could lead to possible loss of life or catastrophic property damage.

5. **Title and Ownership.** This software is protected by United State Patent, Copyright Law and International Treaty provisions. Except for the rights expressly granted

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com

above, this License transfers to you no right, title, or interest in the Software, or any copyright, patent, trademark, trade secret, or other intellectual property or proprietary right in the Software. Priio retains sole and exclusive title to all portions of the Software and any copies thereof, and you hereby assign to Priio all right, title and interest in and to any modifications you make to the Software, whether or not such modifications are permitted. You agree not to disclose the Software to anyone.

6. **Export Law Assurances.** You may not export, re-export, download, or otherwise use the Software except as authorized by United States law and the laws of the jurisdiction in which it is obtained.

7. **Notice to Government End Users.** The Software, including any documentation, is provided to the United States Government with restricted rights. If the Software is supplied to the United State Government, the software is classified as "restricted computer software" as defined in clause 52.227-19 of the FAR. The United States Government's rights to the Software are as provided in clause 52.227-19 if the FAR**.**

8. **Controlling Law and Severability.** This License will be governed by the laws of the United States and the State of Indiana without regard to their provisions regarding conflicts of laws. This License will not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. You irrevocably submit to the jurisdiction of the state and federal courts sitting in Indiana, and any action or proceeding arising out of this License will be heard and determined in such courts. If for any reason a court of competent jurisdiction finds any provision, or portion thereof, to be unenforceable, such provision will be interpreted in order to give effect to such provision to the maximum extent permitted by law, and the remainder of this License will continue in full force and effect.

9. **Complete Agreement.** This License constitutes the entire agreement between the parties with respect to the use of the Software and supersedes all prior or contemporaneous understandings regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by Priio.

BY SELECTING THE "I ACCEPT THE AGREEMENT" OPTION AND CLICKING ON THE "NEXT" BUTTON, OR BY INSTALLING, COPYING, RUNNING, OR OTHERWISE USING THE PRIIO SOFTWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE AGREEMENT, PLEASE CLICK THE "CANCEL" BUTTON, AND DO NOT INSTALL, RUN, COPY, OR OTHERWISE USE THE PRIIO SOFTWARE.

Priio
8645 Guion Rd.
Suite A
Indianapolis, IN 46268
317 | 471.1577
317 | 471.1580 (fax)
www.priio.com