Freescale MSD FATFS

Users Guide

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://www.freescale.com

The following revision history table summarizes changes contained in this document.

Revision Number	Revision Date	Description of Changes
Rev. 0	02/2011	Initial Release.

MSDFATFS User Guide, Rev. 0

Appendix C
FATFS Test Application

Chapter 1 Before You Begin

1.1   About this book

This book describes how to use File Allocation Table File System module with Freescale USB Host Stack. Table 1-1 shows the summary of chapters included in this book.

Table 1-1. MSDFATFSUG summary

Chapter Title	Description
Before You Begin	This chapter provides the prerequisites for reading this book.
Getting Familiar	This chapter provides the information about the File Allocation Table File System software module.
MSD FATFS Architecture	This chapter discusses the architecture design of the File Allocation Table File System module for Freescale USB Host Stack.
Developing Applications	This chapter provides the steps that a developer must take to develop applications on top of the FATFS module.
Working with the Software	This chapter provides the steps to building, running the applications.
FATFS Demo	This chapter provides the setup and running MSD FATFS demo example for CFV1 processors.
FATFS Test Application	This chapter provides the setup and running MSD FATFS test example for CFV1 processors.

1.2   Reference Material

Use this book in conjunction with:

- *Freescale USB Stack with PHDC Host Users Guide* (document MEDUSBHOSTUG, Rev. 4)
- File Allocation Table information at *http://en.wikipedia.org/wiki/File_Allocation_Table*
- FATFS Module Application Note at *http://elm-chan.org/fsw/ff/en/appnote.html*
- *Freescale MSD FATFS API Reference Manual* (document MSDFATFSAPIRM, Rev. 0)
- USB Host source code.
- MSD FATFS source code

We assume that you are familiar with the following reference material:

- USB Specification Revision 1.1
- USB Specification Revision 2.0

1.3 Acronyms and Abbreviations

Table 1-2. Acronyms and Abbreviations

API	Application Programming Interface
COM	Communication
DBCS	Double-Byte Character Set
EVB	Evaluation
FAT	File Allocation Table
FATFS	File Allocation Table File System
IDE	Integrated Development Environment
HCI	Host Controller Interface
MBR	Master Boot Record
MSD	Mass Storage Device
OEM	Original Equipment Manufacturer
PC	Personal Computer
SCSI	Small Computer Systems Interface
USB	Universal Serial Bus

1.4 Key Terms

Table 1-3 shows the terms used throughout the book.

Table 1-3. Importance terms

Term	Description
Code Page	Code page is another name for character encoding. It consists of a table of values that describes the character set for a particular language.
Cluster	To reduce the overhead of managing on-disk data structures, the file system does not allocate individual sectors, but contiguous groups of sectors, called clusters.
FAT12	A type of FAT file system that uses 12 bits value to address clusters.
FAT16	A type of FAT file system that uses 16 bits value to address clusters.
FAR32	A type of FAT file system that uses 32 bits value (in which 4 bits are reserved) to address clusters.
Long File Name	In a file system that supports long file names, a file or directory name can be as long as 255 characters including one or more dots and extensions. A complete path of the file has a maximum of 260 characters, so volumes with many levels of directories must use shorter names.
Sector	Sector is the smallest storage unit in a mass storage medium. Typically, a sector holds 512 bytes of information. However, some medium can have sector size more than 512 bytes.
Partition	A partition is a logical division on mass storage device. The term is also known as Volume or Logical Disk.

Chapter 2 Getting Familiar

2.1　Introduction

The FATFS module is developed based on MSD class of Freescale USB Stack with PHDC Software Suite. Its architecture contains USB driver code, disk I/O interface functions, FAT APIs, and some applications. This document intends to help you gain an insight into the File Allocation Table and capabilities to develop your own applications. The document is targeted for firmware application developers who would like to develop the applications using FATFS file system module.

2.2　File Allocation Table Overview

The mass storage media is organized logically as a Master Boot Record and several partitions. Figure 2-1 describes the logical structures of a mass storage medium.
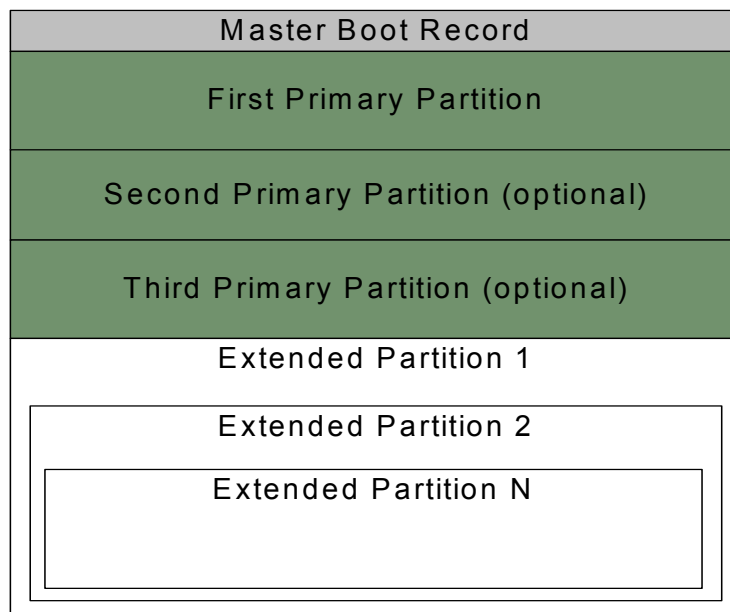
Figure 2-1. Logical structure of mass storage media

The Master Boot Record is located at sector zero. It contains three items: an area for executable code, a partition table, and a boot signature. The partition table enables defining one or more partitions, or logical volumes, in the storage media. Many devices have just one volume. The partition table in the MBR sector has room for four 16-byte entries that specify the sectors that belong to a partition.

A FAT partition composed of four different sections as shown in the following figure.
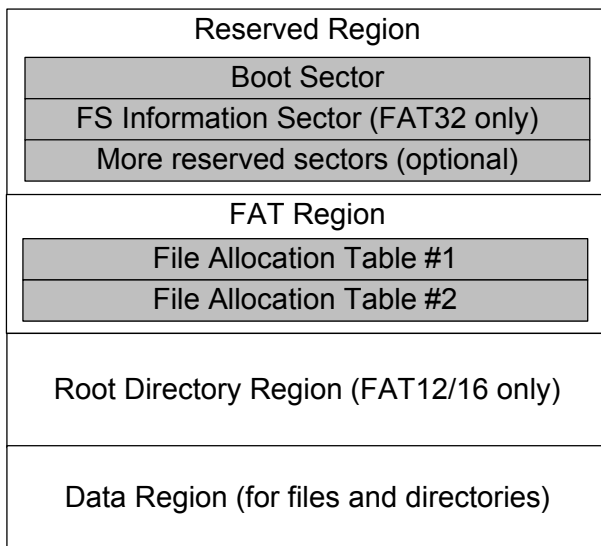
Figure 2-2. FAT partition structure

- **Reserved Region** These sectors are located at the very beginning. The first reserved sector (sector 0) is the Boot Sector (*Partition Boot Record*). It includes an area called the BIOS Parameter Block (with some basic file system information, in particular its type, and pointers to the location of the other sections) and usually contains the operating system's boot loader code. The total count of reserved sectors is indicated by a field inside the Boot Sector. For FAT32 file systems, the reserved sectors include a *File System Information Sector* at Sector 1 and a *Backup Boot Sector* at Sector 6.

- **FAT Region**. This typically contains two copies of the *File Allocation Table* for the sake of redundancy checking, although the extra copy is rarely used, even by disk repair utilities. These are maps of the Data Region, indicating which clusters are used by files and directories. In FAT16 and FAT12, they immediately follow the reserved sectors.

- **Root Directory Region**. This is a *Directory Table* that stores information about the files and directories located in the root directory. It is only used with FAT12 and FAT16, and imposes on the root directory a fixed maximum size which is pre-allocated at creation of this volume. FAT32 stores the root directory in the Data Region, along with files and other directories, allowing it to grow without such a constraint. Therefore, for FAT32, the Data Region starts here.

- **Data Region**. This is where the actual file and directory data is stored and takes up most of the partition. The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the FAT. Note that the files are allocated in units of clusters, so if a 1 KB file resides in a 32 KB cluster, 31 KB are wasted. FAT32 typically commences the Root Directory Table in cluster number 2, the first cluster of the Data Region.

FAT uses little endian format for entries in the header and the FAT(s).

2.3   Software Module

2.3.1   MSD FATFS Feature

The MSD FATFS software module uses class MSD's APIs of Freescale USB Stack with PHDC Host to access mass storage device. The module supports:

- FAT sub-types: FAT12, FAT16, and FAT32
- Number of open files: Unlimited, depends on available memory
- Multi-partition: Number of volumes (up to 10)
- File size: Depends on FAT specs (up to 4 GB)
- Volume size: Depends on FAT specs (up to 2 TB on 512 bytes/sector)
- Cluster size: Depends on FAT specs (up to 64 KB on 512 bytes/sector)
- Sector size: Depends on FAT specs (up to 4 KB)
- Long file name support in ANSI/OEM or Unicode
- Multiple ANSI/OEM code pages including DBCS
- Code size reduction depending on user configuration

The class drivers are programmed with generic code, so they can be used with other processors if standard SCSI commands are provided like MSD class of the Freescale USB Host Stack. In this version, it supports the following Freescale hardware platforms.

- ColdFire V2 family
  — M52221Demo
  — M52259Demo
  — M52259EVB
- CodeFire V1 family
  — CFV1JM128EVB
  — CFV1MM256Demo
  — CFV1JE123Demo
- K40

2.3.2   Module license

FATFS is an open source module. It follows the BSD-style license. Redistributions of source code must retain the following copyright notice.

```
/*-------------------------------------------------------------------------/
/  FATFS - FAT file system module  R0.08b              (C)ChaN, 2011
/--------------------------------------------------------------------------/
/ FATFS module is a generic FAT file system module for small embedded systems.
/ This is a free software that opened for education, research and commercial
/ developments under license policy of following terms.
/
/  Copyright (C) 2011, ChaN, all right reserved.
/
/ * The FATFS module is a free software and there is NO WARRANTY.
/ * No restriction on use. You can use, modify and redistribute it for
/   personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY.
/ * Redistributions of source code must retain the above copyright notice.
/
/--------------------------------------------------------------------------/
```

Because, FATFS is for embedded projects, the conditions for redistributions in binary form, such as embedded code, hex file, and binary library are not specified to increase its usability. The documentation of the distributions need not include FATFS and its license notice.

2.4   Directory structure

The software module has a standard directory structure. You can extend it easily to accommodate more applications for different processor families.

Figure 2-3 shows the directory structure:

```
host ─────────────────────────────────►   USB Host code
  └ examples ─────────────────────────►    Example code
      ⊞ audio ─────────────────────────►   Audio Application
      ⊞ cdc_serial ────────────────────►   CDC Virtual COM Application
      ⊞ common ────────────────────────►   Common files for all application
      ⊞ hid ───────────────────────────►   HID Application
      ⊞ msd ───────────────────────────►   Mass Storage Device Appilcation
      └ msd_mfs_generic ───────────────►   MSD FATFS code and Demo Application
          └ codewarrior ───────────────►   CodeWarrior Project Files
              ⊞ cfv1usbje256 ──────────►   Project for CFV1JE256
              ⊞ cfv1usbjm128 ──────────►   Project for CFV1JM128
              ⊞ cfv1usbmm256 ──────────►   Project for CFV1MM256
              ⊞ m52221demo ────────────►   Project for M52221Demo
              ⊞ m52259demo ────────────►   Project for M52259Demo
              ⊞ m52259evb ─────────────►   Project for M52259EVB
          └ cw10 ──────────────────────►   CodeWarrior 10 Project Files
              ⊞ cfv1usbjm128 ──────────►   Project for CFV1JM128
              ⊞ kinetis ───────────────►   Project for K40
              ⊞ m52221demo ────────────►   Project for M52221Demo
              ⊞ m52259demo ────────────►   Project for M52259Demo
              ⊞ m52259evb ─────────────►   Project for M52259EVB
      ⊞ msd_mfs_generic_test ──────────►   MSD FATFS Test Applications
      ⊞ phdc ──────────────────────────►   PHDC Application
  └ source ────────────────────────────►   Class driver and low level source code
      ⊞ bsp ───────────────────────────►   Dependent source code in low level layer
      ⊞ classes ───────────────────────►   Source code for different USB classes
      ⊞ driver ────────────────────────►   Driver source code
      host_common ─────────────────────►   Host common source code
```

Figure 2-3. MSD FATFS with Freescale USB Host Stack directory structure

Chapter 3 MSD FATFS Architecture

This chapter provides an overview of MSD FATFS architecture and its software flow.

3.1   Architecture Overview

The architecture of MSD FATFS is shown in the following figure.
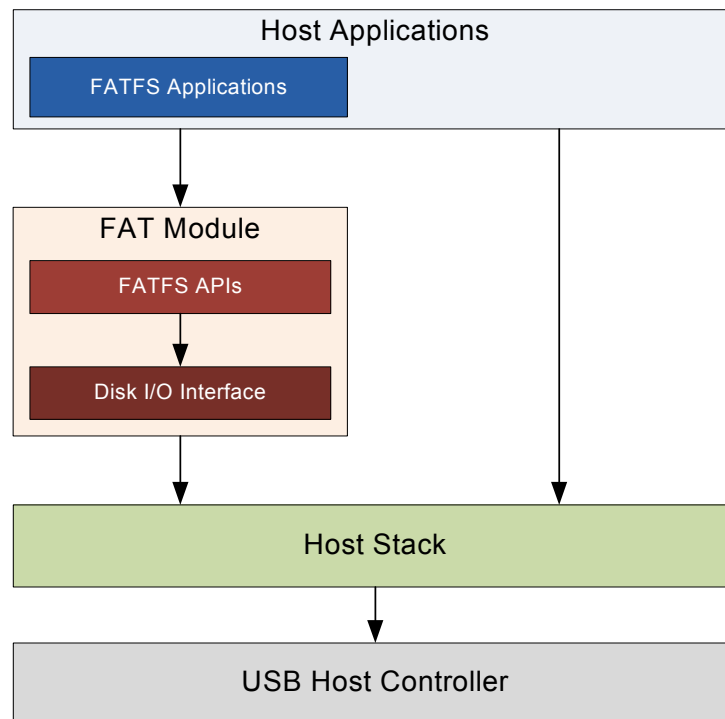
Figure 3-1. FATFS Architecture

The remainder of the document describes only the FATFS module. For more information about the host stack structure and functionality or about the demo application for the different USB classes, refer to the *Freescale USB Stack with PHDC Host Users Guide* (document MEDUSBHOSTUG, Rev. 4).

3.2   FATFS Module Overview

3.2.1   FATFS APIs

The FATFS APIs layer implements file system APIs such as f_open, f_read, f_writes, and so on. This layer is independent with USB Host Stack. It uses Disk I/O interface to communicate with mass storage device. The set of APIs is divided into four groups:

1. Group of APIs that operates with logical volume or partition.

2. Group of APIs that operates with directory.
3. Group of APIs that operates with file.
4. Group of APIs that operates with both file and directory.

APIs of FATFS are listed in Section 4.2, "Configuration Options."

3.2.2 Disk I/O Interface

The Disk I/O Interface consists of six APIs that are used by FATFS API to access and manage data in mass storage device. To confirm with FATFS APIs, the functions must follow the prototype described in section **Disk I/O Interface** of *FatFs Generic File System Module* document. The layer operates with USB Host Stack via three SCSI commands: READ10, WRITE10, and READ CAPACITY that are implemented on Host stack.

The following table lists the APIs of Disk I/O Interface layer.

Table 3-1. Disk I/O interface APIs

APIs	Descriptions
disk_initialize	Initialize disk drive
disk_status	Get disk status
disk_read	Read data sector(s) from mass storage device
disk_write	Write data sector(s) to mass storage device
disk_ioctl	Get information about sector size, sector count and physical volume size.
get_fattime	Get current time of system. At this time, system time utility has not been implemented, so that the function always returns to a fixed date.

Chapter 4 Developing Applications

4.1   Background

FATFS module contains various configuration options. Therefore, this chapter provides information to help user select proper options depending on his requirement to reach the highest performance. Moreover, how to create a new FATFS project is also mentioned here.

4.2   Configuration Options

The following table shows the options for module size reduction.

Table 4-1. Module size reduction options

API	_FS_MINIMIZE				_FS_READ ONLY		_USE_STR FUNC		_FS_RPATH			_USE_MKF S		_USE_FOR WARD	
	0	1	2	3	0	1	0	1	0	1	2	0	1	0	1
f_mount															
f_open															
f_close															
f_read															
f_write						x									
f_sync						x									
f_lseek				x											
f_opendir			x	x											
f_readdir			x	x											
f_stat		x	x	x											
f_getfree		x	x	x		x									
f_truncate		x	x	x		x									
f_unlink		x	x	x		x									
f_mkdir		x	x	x		x									
f_chmod		x	x	x		x									
f_utime		x	x	x		x									
f_remane		x	x	x		x									
f_chdir										x					
f_chdrive										x					

Table 4-1. Module size reduction options

API	_FS_MINIMIZE				_FS_READ ONLY		_USE_STR FUNC		_FS_RPATH			_USE_MKFS		_USE_FOR WARD	
	0	1	2	3	0	1	0	1	0	1	2	0	1	0	1
f_getcwd									x	x		x			
f_mkfs						x									
f_forward														x	
f_putc						x	x								
f_puts						x	x								
f_printf						x	x								
f_gets							x								
f_eof															
f_error															
f_tell															
f_size															

x- API is removed

Other configuration options for FATFS module are described in the following table.

Table 4-2. General FATFS configuration options

Feature	Option	Value	Description
Multi-partitions	_VOLUMES	1 to 4	Number of volumes to be used
	_MULTI_PARTITION	0	Disable multi-partitions feature
		1	Enable multi-partitions feature
Memory access	_WORD_ACCESS	0	Retrieve data from FAT volume byte by byte
		1	Retrieve data from FAT volume word by word
Open multi-files	_FS_SHARE	integer	Number of files can be opened simultaneously for write
Memory size	_FS_TINY	0	FATFS uses the sector buffer in the system for file data transfer. This reduces memory consumption 512 bytes each file object
		1	FATFS uses a sector buffer for the individual file object for file data transfer
Sector size	_MAX_SS	512, 1024, 2048, 4096	Maximum sector size to be handled

Table 4-2. General FATFS configuration options (continued)

Feature	Option	Value	Description
Long File Name	**_CODE_PAGE**	437	Used U.S. (OEM)
		720	Used Arabic (OEM)
		737	Used Greek (OEM)
		775	Used Baltic (OEM)
		850	Used Multilingual Latin 1 (OEM)
		858	Used Multilingual Latin 1 + Euro (OEM)
		852	Used Latin 1 (OEM)
		855	Used Cyrillic (OEM)
		866	Used Russian (OEM)
		857	Used Turkish (OEM)
		862	Used Hebrew (OEM)
		874	Used Thai (OEM, Windows)
		1	ASCII only (valid for non - LFN configuration)
		1250	Used Central Europe (Windows)
		1251	Used Cyrillic (Windows)
		1252	Used Latin 1 (Windows)
		1253	Used Greek (Windows)
		1254	Used Turkish (Windows)
		1255	Used Hebrew (Windows)
		1256	Used Arabic (Windows)
		1257	Used Baltic (Windows)
		1278	Used Vietnam (OEM, Windows)
	_USE_LFN	0	Disable LFN feature. _MAX_LFN and _LFN_UNICODE have no effect
		1	Enable LFN with static working buffer on the BSS
		2	Enable LFN with dynamic working buffer on the STACK
		3	Enable LFN with dynamic working buffer on the HEAP
	-MAX_LFN	12 to 255	Maximum LFN length to handle
	_LFN_UNICODE	0	The character code set on FATFS APIs is ANSI/OEM
		1	The character code set on FATFS APIs is Unicode
	_FS_RPATH	0	Disable relative path
		1	Enable relative path

MSDFATFS User Guide, Rev. 0

Table 4-2. General FATFS configuration options (continued)

Feature	Option	Value	Description
Multi-partitions	**_VOLUMES**	1 to 4	Number of volumes to be used
	_MULTI_PARTITION	0	Disable multi-partitions feature
		1	Enable multi-partitions feature
Memory access	**_WORD_ACCESS**	0	Retrieve data from FAT volume byte by byte
		1	Retrieve data from FAT volume word by word
Open multi-files	**_FS_SHARE**	integer	Number of files can be opened simultaneously for write
Memory size	**_FS_TINY**	0	FATFS uses the sector buffer in the system for file data transfer. This reduces memory consumption 512 bytes each file object
		1	FATFS uses a sector buffer for the individual file object for file data transfer
Sector size	**_MAX_SS**	512, 1024, 2048, 4096	Maximum sector size to be handled

4.3 Create a Project

Perform these steps to develop a new application:

1. Create a new project under /**host/examples/msd_mfs_generic/codewarrior** or **/host/examples/ msd_mfs_generic/cw10** directory.
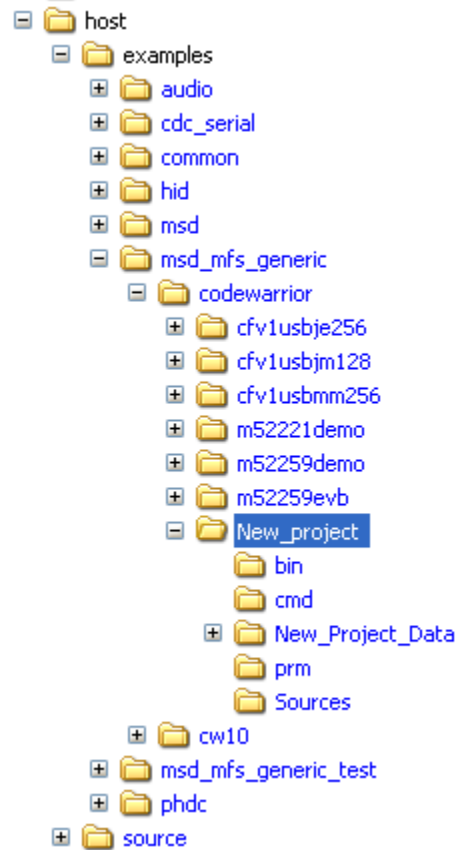

Figure 4-1. Create a new project

2. Add **ccsbcs.h, diskio.h, diskio.c, ff.h, ff.c, ffconf.h, main.c, usb_class.h, msd_fat_demo.c**, and other files to the created project similar to the pre-existing FATFS applications.
3. Modify FATFS module options in the file **ffconf.h**.
4. Modify FATFS application task in the file **msd_fat_demo.c** (**fat_demo** function) as you want.

Appendix A Working with the Software

A.1 Introduction

This chapter gives you insight on how to use the FATFS module with Freescale USB Stack with PHDC Host. The following sections are described in this chapter:

- Preparing the setup
- Building the application
- Running the application

A.1.1 Preparing the setup

A.1.1.1 Software setup

1. Make sure you have Freescale USB Stack with PHDC v3.0 package already installed.
2. Double-click on the FSL_USB_MSD_FATFS_v1.0.exe file.
3. The Freescale USB MSD FATFS 1.0 Setup Wizard window appears. Click the **Next** button.
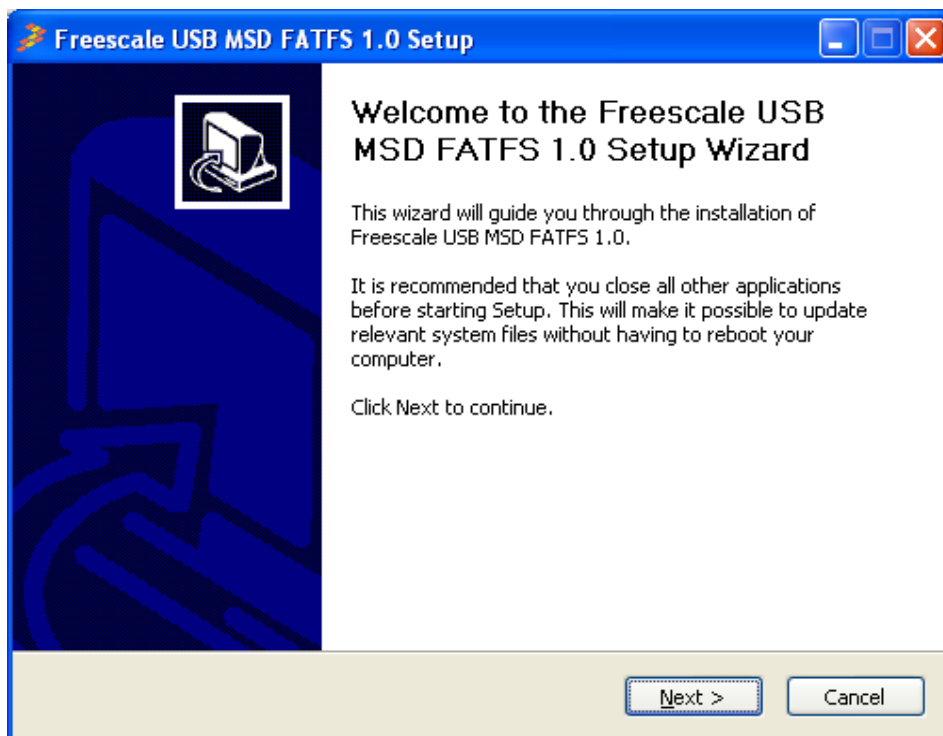

Figure A-1. Freescale USB MSD FATFS v1.0 Setup Wizard

4. In the following window, click **I Agree** to accept the license agreement.
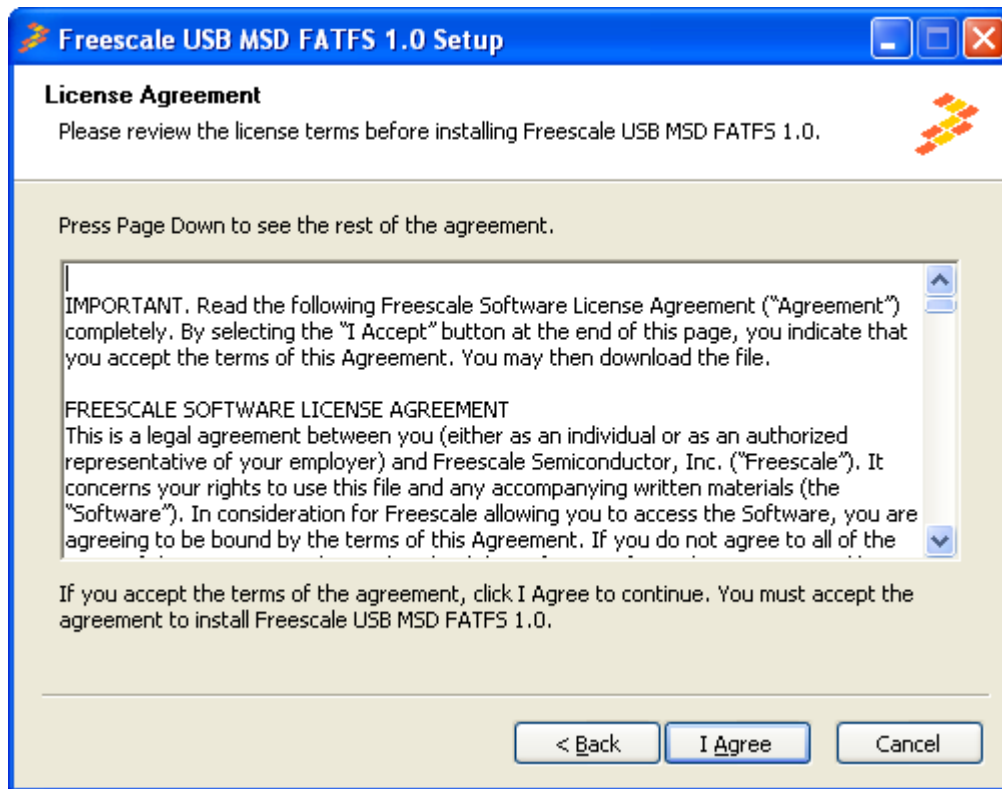
Figure A-2. Freescale USB MSD FATFS v1.0 license agreement

5. In the following window, select the USB MSD FATFS components to be installed and click on the **Next** button.
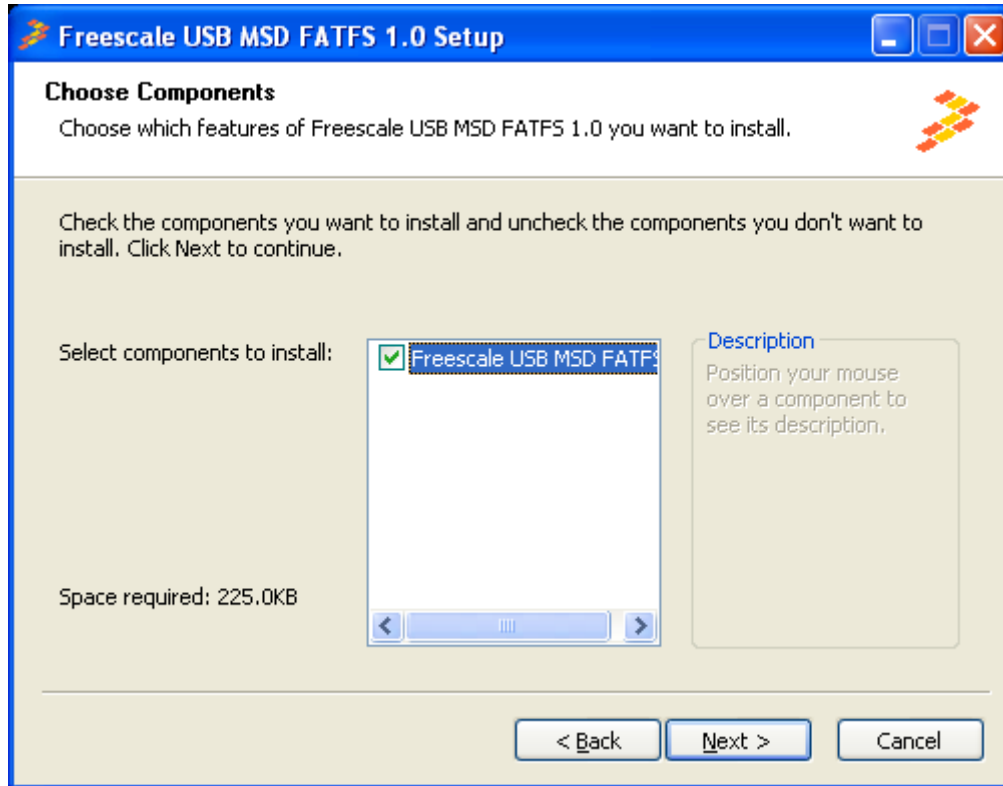
Figure A-3. Freescale USB MSD FATFS v1.0 component

6. In the following window, select the location where you want to install the Freescale USB MSD FATFS v1.0 (usually the same location as Freescale USB Stack with PHDC v3.0) and click the **Install** button.

Figure A-4. Freescale USB MSD FATFS v1.0 installation folder location

7. Click the **Finish** button to complete the installation of the Freescale USB MSD FATFS v1.0 package.

Figure A-5. Freescale USB MSD FATFS v1.0 installation finish

8. Click in the Windows Start Menu on **Start->Programs->Freescale USB MSD FATFS v1.0->Source->USB Host->Source** to find the installed demo applications.

A.1.1.2   Hardware setup

Make the connections as shown in the following figure.
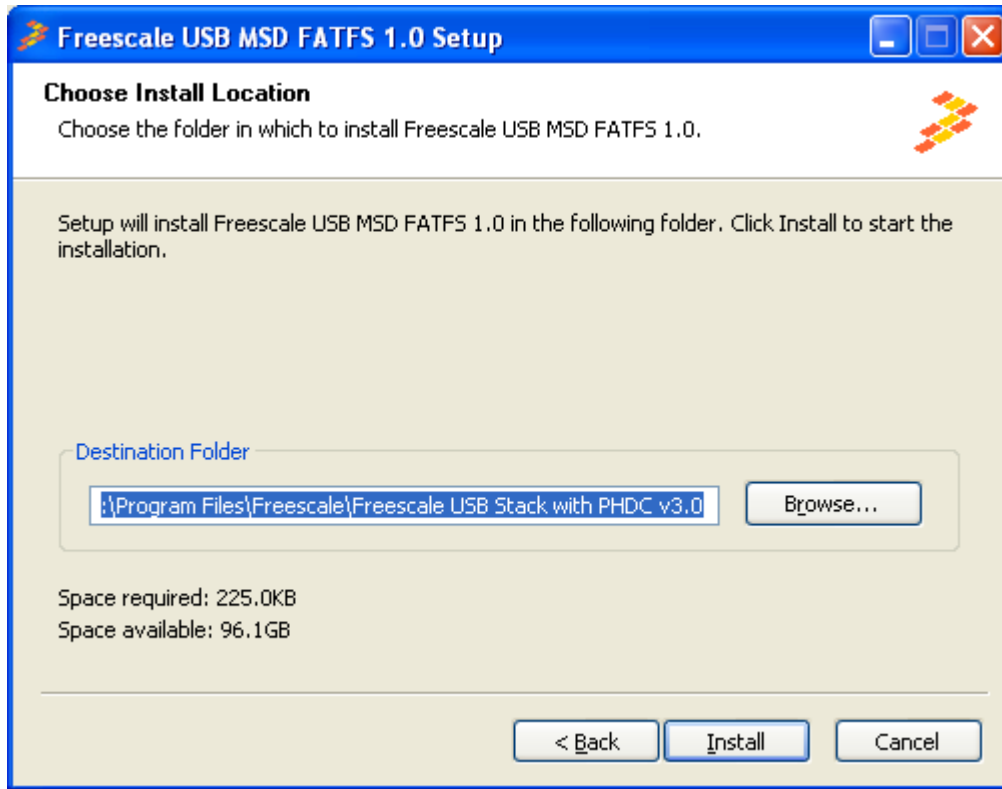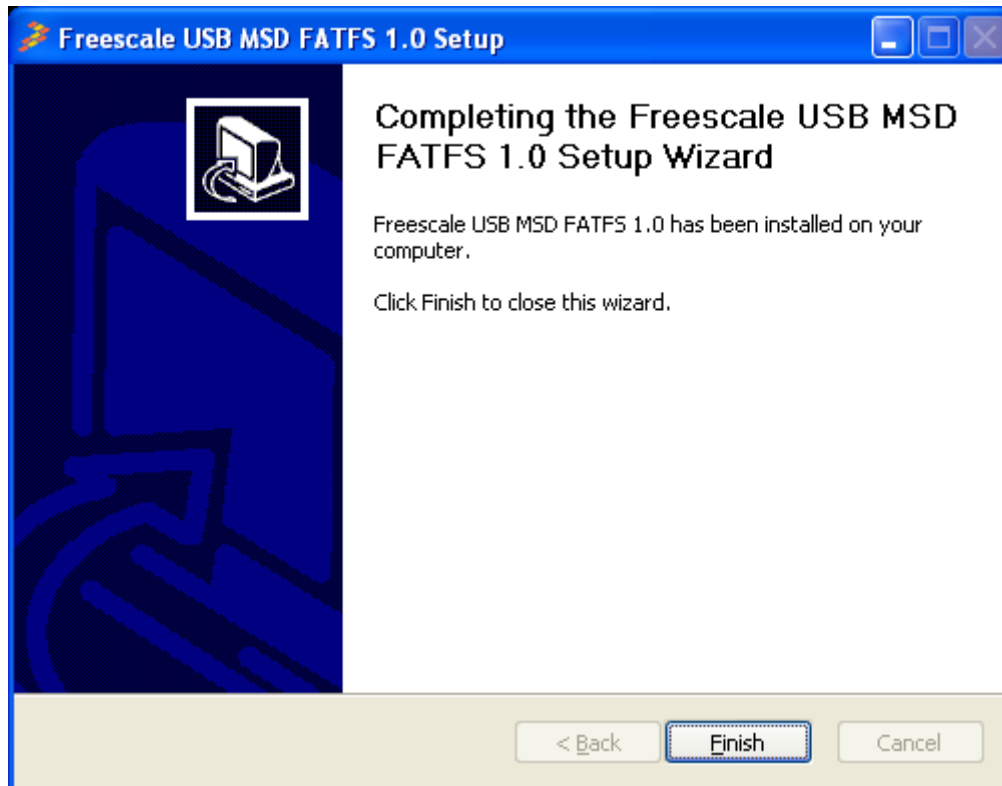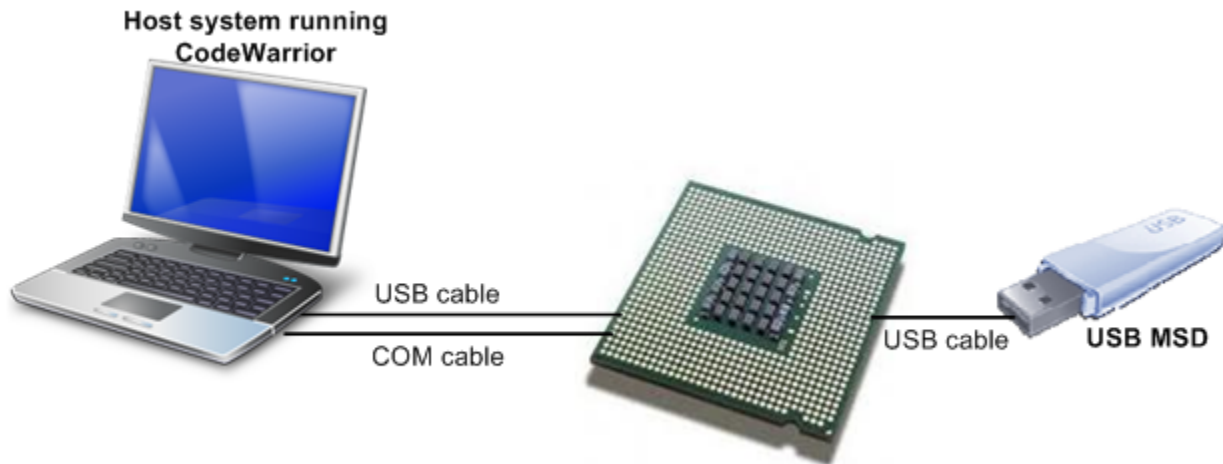

Figure A-6. Coldfire V1 USB Setup

— Make the first USB connection between the personal computer where the software is installed and the DemoJM board where the silicon is mounted. this connection is required to provide power to the board and downloading image to the flash.

— Make the second connection between the DemoJM board and the personal computer to display the log of DemoJM.

— Make the third connection between the device and DemoJM.

A.1.2   Building the application

For information on how to build the demo application, refer to **A.1.2** section of *Freescale USB Stack with PHDC Host Users Guide* (document MEDUSBHOSTUG) or *Freescale USB Stack with PHDC Device Users Guide* (document MEDUSBUG.).

A.1.3   Running the application

For information on how to run the demo application, refer to **A.1.3** section of *Freescale USB Stack with PHDC Host Users Guide* (document MEDUSBHOSTUG) or *Freescale USB Stack with PHDC Device Users Guide* (document MEDUSBUG).

A.2   Setup HyperTerminal to get log

To ensure that applications run correctly, the HyperTerminal is used on your computer to get events from the devices which connect to the CFV1. These steps are used to configure HyperTerminal:

1. Open HyperTerminal applications as shown in the following figure.
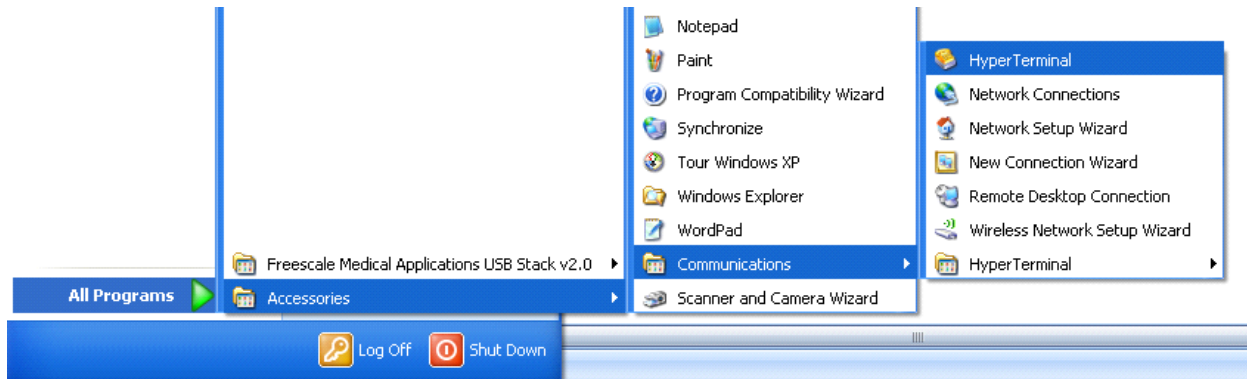
Figure A-7. Launch HyperTerminal application

2. The HyperTerminal opens as shown in the following figure. Enter the name of connection and click on the **OK** button.
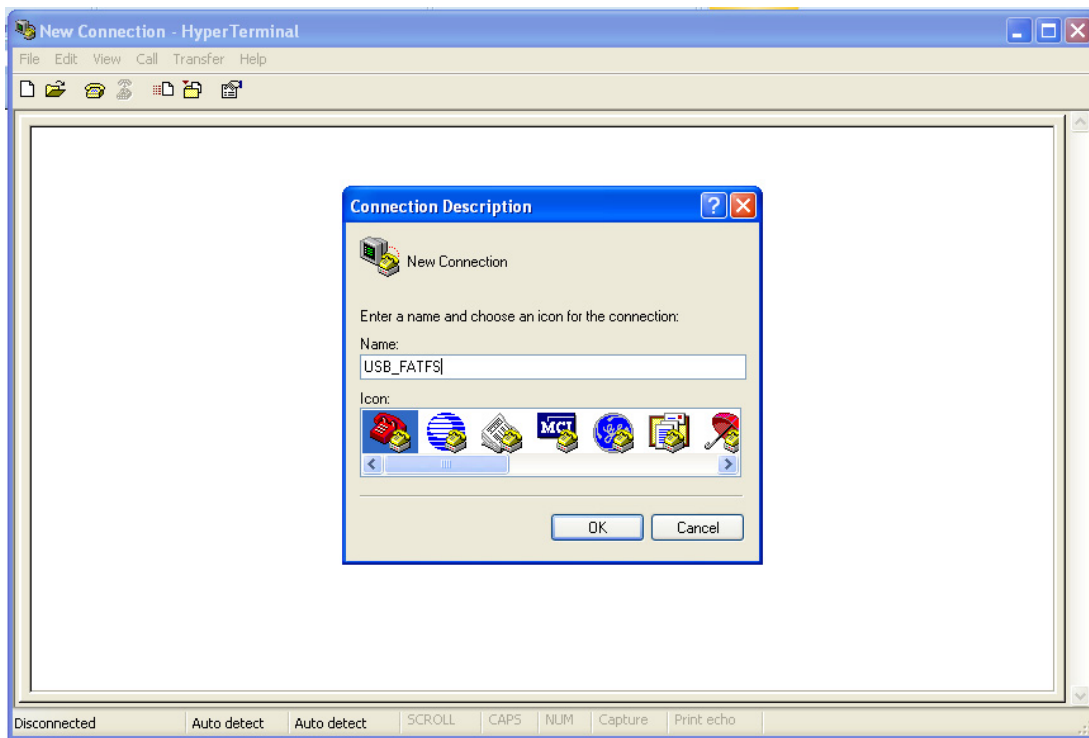
Figure A-8. HyperTerminal GUI

3. The window shown in the following figure appears. Select the COM port identical to the one that shows up on the device manager.
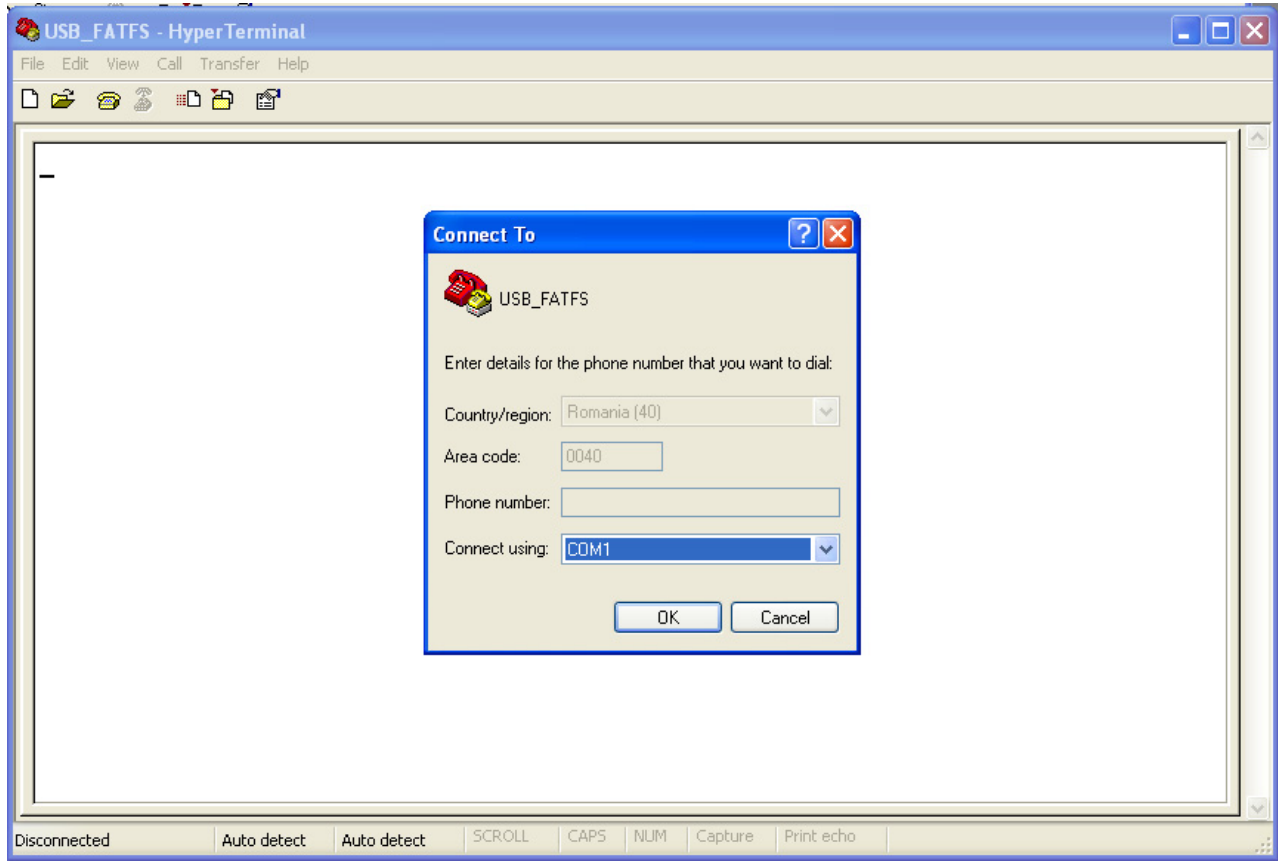
Figure A-9. Connect using COM1

4. In the next window set the communication baud rate to 115200, data length to 8, no parity, one stop bit and no flow control, then click **OK** to complete the HyperTerminal configuration. Configure the virtual COM port baud rate and other properties as shown in the following figure.
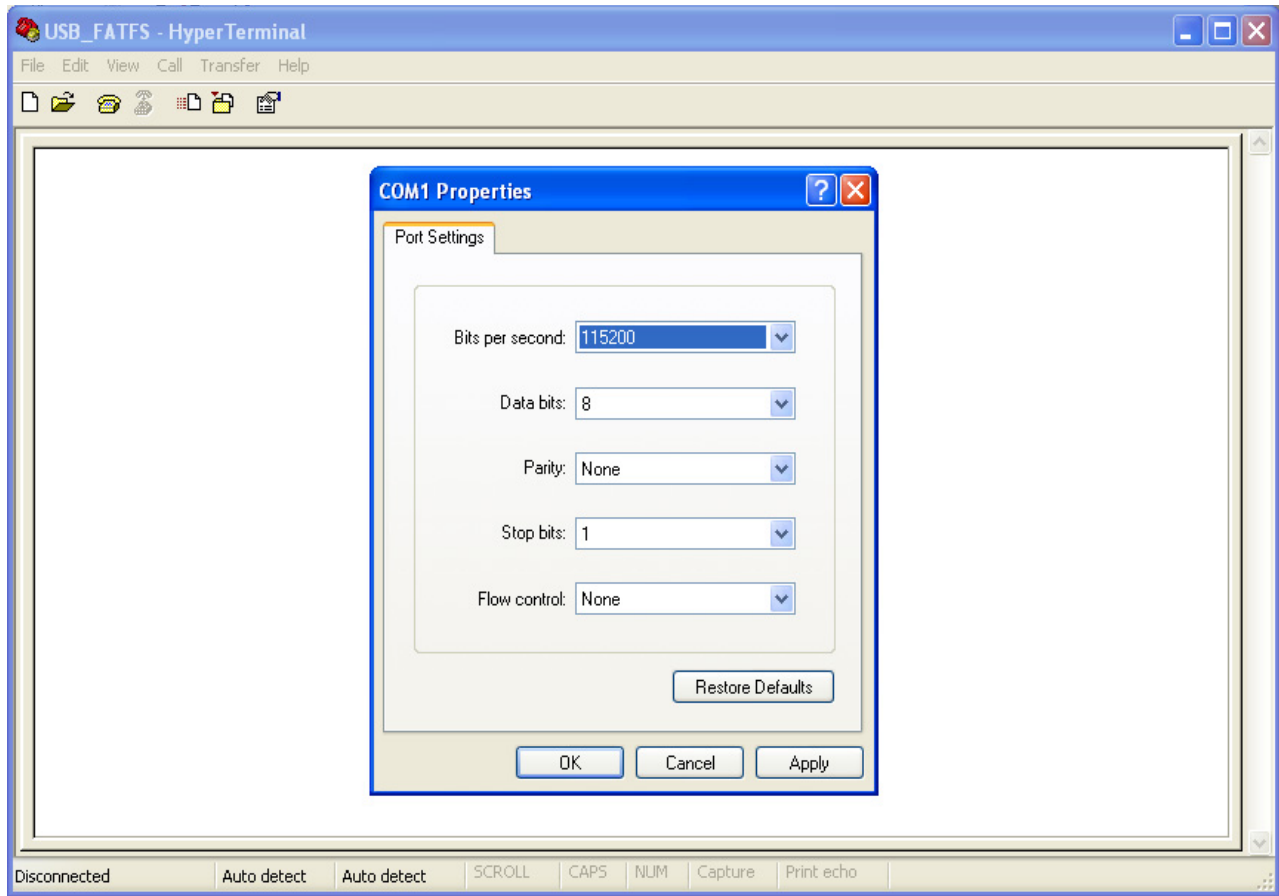
Figure A-10. COM1 properties

5. The HyperTerminal is configured now as shown in the following figure.
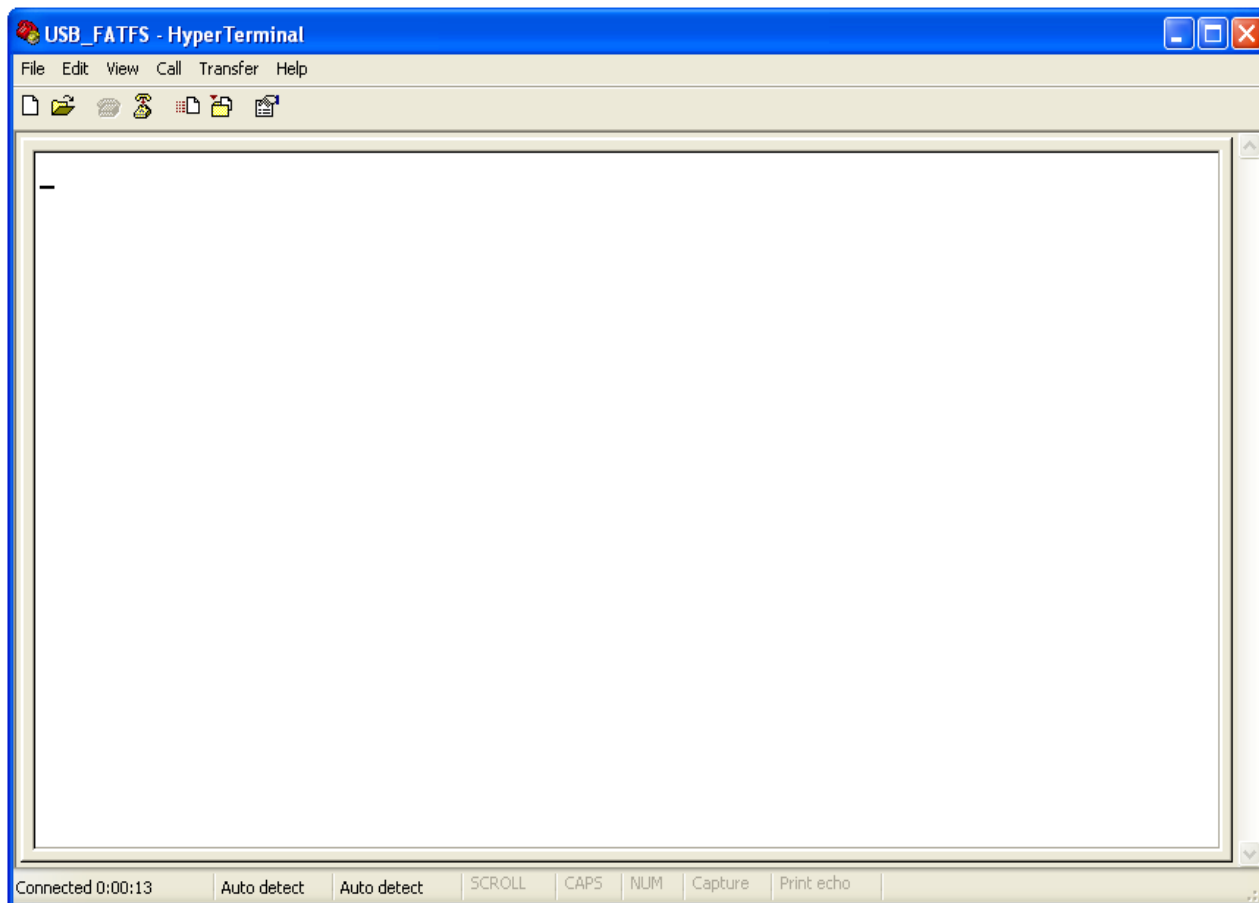
Figure A-11. HyperTerminal startup

Appendix B FATFS Demo

The demo application demonstrates how to use application interface functions of the FATFS module to operate with file and directory of mass storage devices.

B.1   Setting up the demo

Set the system as described in the Section A.1.1.2, "Hardware setup."

B.2   Running the demo

B.2.1   Mouse demo

Perform the following steps to run the mouse demo:

1. Open and load the image of MSD FATFS Demo application to the board.
2. After the image has been loaded successfully, the HyperTerminal appears as shown in Figure B-1.

Figure B-1. The USB Host is waiting the mass storage device attachment event

3. Plug an USB Mass Storage Device into the board. The Mass Storage Device will be attached and all functionalities of FATFS are implemented, sequentially and the results are shown in the HyperTerminal. The detail of display content is shown as following:

```
FAT demo
Waiting for USB mass storage to be attached...
Mass Storage Device Attached
****************************************************************************
*                       FATfs DEMO                        *
*        Configuration:  LNF Enabled, Code page  =1258              *
****************************************************************************

****************************************************************************
*                       DRIVER OPERATION                       *
****************************************************************************
1. Demo funciton: f_mount


  Initializing logical drive 0...
  Initialization complete
-------------------------------------------------------------------------


2. Demo funcitons:f_getfree, f_opendir, f_readdir

getting drive 0 attributes...............
Logical drive 0 attributes:
 FAT type = FAT32
 Bytes/Cluster = 512
 Number of FATs = 2
 Root DIR entries = 0
Sectors/FAT = 618
 Number of clusters = 78931
 FAT start (lba) = 158
 DIR start (lba,clustor) = 623
 Data start (lba) = 1394


...
39465 KB total disk space.
25666 KB available.
-------------------------------------------------------------------------
FAT type = FAT32
 Bytes/Cluster = 512
 Number of FATs = 2
Root DIR entries = 0
```

```
 Sectors/FAT = 618
 Number of clusters = 78931
 FAT start (lba) = 158
 DIR start (lba,clustor) = 623
 Data start (lba) = 1394


...
39465 KB total disk space.
25666 KB available.
-------------------------------------------------------------------------
*****************************************************************************
*                   DRECTORY OPERATION                         *
*****************************************************************************
1. Demo funcitons:f_opendir, f_readdir


 Directory listing...
   D---- 2010/12/23 15:41        0  New Folder
   DR--- 2010/12/25 23:30        0  Directory_1
   ----A 2010/12/23 15:42       33  dsgsgsg.dat
   D---- 2010/01/01 00:00        0  Directory_2
   ----A 2010/01/01 00:00       32  file_test.txt
   ----A 2010/12/28 16:26  1307648  FSL_USB_MSD_FATFS_Development_Design_v1.1.doc
   ----A 2010/12/09 08:43   826338  ff8a.zip
   D-HS- 2010/12/28 18:12        0  Recycled
   D-HS- 2010/12/28 18:12        0  System Volume Information
   ----A 2010/12/28 10:19   302592  FSL_USB_MSD_FATFS_Demo_SDD_V1.1.doc
   D---- 2010/12/28 18:19        0  Freescale USB Stack with PHDC v2.6
   ----A 2010/12/30 17:52    65024  FSL_USB_MSD_FAT_Development_System Test
Case.v0.1.xls
   ----A 2010/12/29 19:15   477734  2010_12_29_MSD_FATFS_Source_Code.zip
   ----A 2010/12/29 16:23  3880022  fat-2006-12-03.zip


   8   File(s),   6859423 bytes total
   6   Dir(s)

-------------------------------------------------------------------------
2. Demo funcitons:f_mkdir


2.0. Create <Directory_1>
2.1. Create <Directory_2>
2.2. Create <Sub1> as a sub directory of <Directory_1>
2.3. Directory list
Directory listing...
```

```
 D---- 2010/12/23 15:41      0  New Folder
  DR--- 2010/12/25 23:30       0  Directory_1
  ----A 2010/12/23 15:42       33  dsgsgsg.dat
  D---- 2010/01/01 00:00       0  Directory_2
  ----A 2010/01/01 00:00       32  file_test.txt
  ----A 2010/12/28 16:26   1307648  FSL_USB_MSD_FATFS_Development_Design_v1.1.doc
  ----A 2010/12/09 08:43   826338  ff8a.zip
  D-HS- 2010/12/28 18:12       0  Recycled
  D-HS- 2010/12/28 18:12       0  System Volume Information
  ----A 2010/12/28 10:19   302592  FSL_USB_MSD_FATFS_Demo_SDD_V1.1.doc
  D---- 2010/12/28 18:19       0  Freescale USB Stack with PHDC v2.6
  ----A 2010/12/30 17:52    65024  FSL_USB_MSD_FAT_Development_System Test
Case.v0.1.xls
  ----A 2010/12/29 19:15   477734  2010_12_29_MSD_FATFS_Source_Code.zip
  ----A 2010/12/29 16:23   3880022  fat-2006-12-03.zip


  8   File(s),   6859423 bytes total
  6   Dir(s)
--------------------------------------------------------------------------
3. Demo funcitons:f_getcwd, f_chdir

3.0. Get the current directory
  CWD: 0:/
3.1. Change current directory to <Directory_1>
3.2. Directory listing
 Directory listing...
  D---- 2010/01/01 00:00      0  .
  D---- 2010/01/01 00:00      0  ..
  D---- 2010/01/01 00:00      0  sub1


  0   File(s),       0 bytes total
  3   Dir(s)
3.3. Get the current directory
  CWD: 0:/Directory_1
--------------------------------------------------------------------------

4. Demo funcitons:f_stat(File status), f_chmod, f_utime

4.1. Get directory information of <Directory_1>
  DR--- 2010/12/25 23:30      0  DIRECT~1
4.2  Change the timestamp of Directory_1 to 12.25.2010: 23h 30' 20
```

```
4.3. Set Read Only Attribute to Directory_1
4.4. Get directory information (Directory_1)
   DR--- 2010/12/25 23:30        0  DIRECT~1
------------------------------------------------------------------------
5. Demo funcitons:f_rename

Rename <sub1> to <sub1_renamed> and move it to <Directory_2>
 Directory listing...
   D---- 2010/01/01 00:00        0  .
   D---- 2010/01/01 00:00        0  ..
   D---A 2010/01/01 00:00        0  sub1_renamed


   0   File(s),       0 bytes total
   3   Dir(s)
------------------------------------------------------------------------

6. Demo funcitons:f_unlink

 Delete Directory_1/sub1_renamed
 Directory listing...
   D---- 2010/01/01 00:00        0  .
   D---- 2010/01/01 00:00        0  ..


   0   File(s),       0 bytes total
   2   Dir(s)


****************************************************************************
*                      FILE OPERATION                       *
****************************************************************************
1. Demo funcitons:f_open,f_write, f_printf, f_putc, f_puts, fclose

1.0. Create new file <New_File_1> (f_open)
   File size =    0
1.1. Write data to <New_File_1>(f_write)
1.2. Flush cached data
   File size =   52
1.3. Write data to <New_File_1> (f_printf)
1.4. Flush cached data
   File size =  103
1.5. Write data to <New_File_1> (f_puts)
```

```
1.6. Flush cached data
   File size =  152
1.7. Write data to <New_File_1> uses f_putc function
1.8. Flush cached data
   File size =  199
1.9. Close file <New_File_1>
-----------------------------------------------------------------------
2. Demo funcitons:f_open,f_read, f_seek, f_gets, f_close


2.0. Open <New_File_1> to read (f_open)
2.1. Get a string from file (f_gets)
   Line 1: Write data to  file uses f_write function
2.2 Get the rest of file content (f_read)
ine 2: Write data to file uses f_printf function
Line 3: Write data to file uses f_puts function
Line 4: Write data to file uses f_putc function Š
2.2. Close file (f_close)
-----------------------------------------------------------------------

2. Demo funcitons:f_stat, f_utime, f_chmod

3.1. Get  information of <New_File_1> file (f_stat)
   ----A 2010/01/01 00:00      199  NEW_FI~1.DAT
3.2  Change the timestamp of Directory_1 to 12.25.2010: 23h 30' 20 (f_utime)
3.3. Set Read Only Attribute to <New_File_1> (f_chmod)
3.4. Get directory information of <New_File_1> (f_stat)
  -R--A 2010/12/25 23:30      199  NEW_FI~1.DAT
3.5. Clear Read Only Attribute of <New_File_1> (f_chmod)
3.6. Get directory information of <New_File_1>
   ----A 2010/12/25 23:30      199  NEW_FI~1.DAT
-----------------------------------------------------------------------
4. Demo funcitons:f_ulink

 Rename <New_File_1.dat> to  <File_Renamed.txt>
 Directory listing...
   D---- 2010/01/01 00:00        0  .
   D---- 2010/01/01 00:00        0  ..
   ----A 2010/12/25 23:30      199  File_Renamed.txt


1   File(s),     199 bytes total
```

```
 2   Dir(s)
--------------------------------------------------------------------------
5. Demo funcitons:f_truncate


 Truncate file <File_Renamed.txt>
5.0. Open <File_Renamed.txt> to write
5.1. Seek file pointer
   Current file pointer:   0
   File pointer affter seeking:  102
5.2. Truncate file
   File size =  102
5.3. Close file
--------------------------------------------------------------------------


6. Demo funcitons:f_forward


6.0. Open <File_Renamed.txt> to read
6.1. Forward file to ternimal
Line 1: Write data to  file uses f_write function
Line 2: Write data to file uses f_printf function
6.2. Close file
--------------------------------------------------------------------------
7. Demo funcitons:f_ulink


 Delete <File_Renamed.txt>
 Directory listing...
   D---- 2010/01/01 00:00       0 .
   D---- 2010/01/01 00:00       0 ..


   0   File(s),       0 bytes total
   2   Dir(s)


*------------------------------   DEMO COMPLETED     ------------------------ *
**************************************************************************
```

4. Unplug mouse from board. The HyperTerminal shows a message as shown in Figure B-2.

Figure B-2. Mass storage device detached

Appendix C FATFS Test Application

The test application is used to verify whether or not application interface functions of the FAT module work properly.

C.1   Setting up the demo

Set the system as described in the Section A.1.1.2, "Hardware setup."

C.2   Running the demo

Steps to run test application are similar to demo application described in Section B.2, "Running the demo."

NOTE

Make sure that your USB mass storage device under test is divided into two partitions which do not contain any data.

There are some test cases that need special setting in FATFS module configuration (ffconf.h), so test case set is divided into three exclusive running groups:

1. Test group 1
2. Test group 2
3. Test group 3

C.2.1 Test Group 1

The test group 1 contains the following subgroups:

Table 4-3. Test group 1

Subgroup	Description	FATFS module configuration
TestDir1	This test group is to test f_mkdir, f_unlink functions with 0 of recursive level	#define _FS_TINY 1 #define _FS_READONLY 0 #define _FS_MINIMIZE 0 #define _USE_STRFUNC 1 #define _USE_FORWARD 1 #define _USE_LFN 3 #define _MAX_LFN 255 #define _FS_RPATH 2 #define _MULTI_PARTITION 0 #define _VOLUMES 1
TestDir2	This test group is to test f_mkdir in cases of invalid directory names	
TestDir3	This test group is to test f_unlink in cases of invalid directory names	
TestDir4	This test group is to test f_mkdir, f_unlink functions with 1 of recursive level	
TestDir5	This test group is to test f_mkdir, f_unlink functions with 2 of recursive level	
TestDir6	This test group is to test f_chdir, f_getcwd, f_unlink functions	
TestDir7	This test group is to test f_mkdir and f_unlink many of sub-directories	
TestDir8	This test group is to test f_opendir, f_readdir functions	
TestDir9	This test group is to test f_chdir function with ".." directory	
TestDir10	This test group is to test f_readdir in case of there are many files in read directory	
TestDir11	This test group is to test f_stat, f_utime, f_chmod functions	
TestFile1	This test group is to test f_open, f_close, and f_unlink functions	
TestFile2	This test group is to test f_write and f_read functions	
TestFile3	This test group is to test f_lseek function	
TestFile4	This test group is to test f_stat, f_utime, f_chmod functions	
TestFile5	This test group is to test f_forward function	
TestFile6	This test group is to test f_truncate function	
TestFile7	This test group is to test f_sync function	
TestFile8	This test group is to test string functions	
TestDirFileMixup1	This test group is to test mix file and directory	

To enable the test group, define the macro **RUN_TEST_101_111_201_209_301** in file **testcase.h**. Subgroups **TestDir7, TestDir8,** and **TestDir10** contain test cases that make or create a lot of directories and files. It takes long time, if created, the number of directories and files is large. How many directories and files will be created is specified by macro **NUM_REPEAT** in **testcase.h** file.

Expected results of these test cases are shown in the HyperTerminal as follows.

FAT test

Waiting for USB mass storage to be attached...

Mass Storage Device Attached

Test Cases:

 101: Test Directory Functions - 1: f_mkdir, f_unlink functions with 0 of recursive level.

 102: Test Directory Functions - 2: f_mkdir in cases of invalid directory names.

 103: Test Directory Functions - 3: f_unlink in cases of invalid directory names.

 104: Test Directory Functions - 4: f_mkdir, f_unlink functions with 1 of recursive level.

 105: Test Directory Functions - 5: f_mkdir, f_unlink functions with 2 of recursive level.

 106: Test Directory Functions - 6: f_chdir, f_getcwd, f_unlink functions.

 107: Test Directory Functions - 7: Make maximum number of sub-directories.

 108: Test Directory Functions - 8: f_opendir, f_readdir functions.

 109: Test Directory Functions - 9: f_chdir function with .. directory.

 110: Test Directory Functions - 10: f_readdir in case of there are many files in read directory.

 111: Test Directory Functions - 11: f_stat, f_utime, f_chmod functions.

 201: Test File Functions - 1: f_open, f_close, and f_unlink

 202: Test File Functions - 2: f_write and f_read

 203: Test File Functions - 3: f_lseek

 204: Test File Funttions - 4: f_stat, f_utime, f_chmod

 205: Test File Functions - 5: f_forward

 206: Test File Functions - 6: f_truncate

 207: Test File Functions - 7: f_sync

 208: Test File Functions - 8: f_printf, f_puts, f_putc, f_gets

 209: Test File Functions - 9: f_rename

 301: File/Dir: file operations on dirs & vice versa

Test case 101: Test Directory Functions - 1: f_mkdir, f_unlink functions with 0 of recursive level.

Test case passed

Test case 102: Test Directory Functions - 2: f_mkdir in cases of invalid directory names.

Test case passed

Test case 103: Test Directory Functions - 3: f_unlink in cases of invalid directory names.

Test case passed

Test case 104: Test Directory Functions - 4: f_mkdir, f_unlink functions with 1 of recursive level.
Test case passed

Test case 105: Test Directory Functions - 5: f_mkdir, f_unlink functions with 2 of recursive level.
Test case passed

Test case 106: Test Directory Functions - 6: f_chdir, f_getcwd, f_unlink functions.
Test case passed

Test case 107: Test Directory Functions - 7: Make maximum number of sub-directories.
Test case passed

Test case 108: Test Directory Functions - 8: f_opendir, f_readdir functions.
Test case passed

Test case 109: Test Directory Functions - 9: f_chdir function with .. directory.
Test case passed

Test case 110: Test Directory Functions - 10: f_readdir in case of there are many files in read directory.
Test case passed

Test case 111: Test Directory Functions - 11: f_stat, f_utime, f_chmod functions.
Test case passed

Test case 201: Test File Functions - 1: f_open, f_close, and f_unlink
Test case passed

Test case 202: Test File Functions - 2: f_write and f_read
Test case passed

Test case 203: Test File Functions - 3: f_lseek
Test case passed

```
Test case 204: Test File Functions - 4: f_stat, f_utime, f_chmod
Test case passed


Test case 205: Test File Functions - 5: f_forward
Test case passed


Test case 206: Test File Functions - 6: f_truncate
Test case passed


Test case 207: Test File Functions - 7: f_sync
Test case passed


Test case 208: Test File Functions - 8: f_printf, f_puts, f_putc, f_gets
Test case passed


Test case 209: Test File Functions - 9: f_rename
Test case passed


Test case 301: File/Dir: file operations on dirs & vice versa
Test case passed


Test cases:
Executed: 21, Passed: 21, Failed: 0
```

C.2.2   Test Group 2

This test group contains the following subgroup.

Table 4-4. Test Group 2

Subgroup	Description	FATFS module configuration
TestDir12	This test group is to test f_chdrive, f_getfree, f_mount functions. It also test multi-partition feature of FATFS module	#define _FS_TINY 1 #define _FS_READONLY 0 #define _FS_MINIMIZE 0 #define _MULTI_PARTITION 1 #define _VOLUMES 2 #define _FS_RPATH 2

To enable the test group, define the macro **RUN_TEST_112** in file **testcase.h**.

Expected results of the test case are shown in the HyperTerminal as follows.

```
FAT test
Waiting for USB mass storage to be attached...
Mass Storage Device Attached
Test Cases:
  112: Test Directory Functions - 12: f_chdrive, f_getfree, f_mount functions.


Test case 112: Test Directory Functions - 12: f_chdrive, f_getfree, f_mount functions.
Disk '0':
  ClusterSize: 4096
  TotalClusterCount: 1994
  TotalFreeClusterCount: 1994

Disk '1':
  ClusterSize: 512
  TotalClusterCount: 94874
  TotalFreeClusterCount: 94873

Test case passed
Test cases:
Executed: 1, Passed: 1, Failed: 0
```

C.2.3   Test Group 3

The test group consists of following subgroups.

Table 4-5. Test Group 3

Subgroup	Description	FATFS module configuration
TestFile10	This test group is to test file sharing policy.	#define _FS_TINY 1 #define _FS_READONLY 0 #define _FS_MINIMIZE 0 #define _MULTI_PARTITION 0 #define _VOLUMES 1 #define _FS_RPATH 0 #define _FS_SHARE 2 #define _USE_LFN 0
TestFile11	This test group is to test how FAT apis work when drive status is invalid.	
TestFile12	This test group is to test how FAT apis work when LFN is disable.	
TestFile13	This test group is to test how FAT apis work when _RS_PATH = 0.	

To enable the test group, define the macro **RUN_TEST_210_213** in file **testcase.h**.

Expected results of these test cases are shown in the HyperTerminal as follows.

FAT test

Waiting for USB mass storage to be attached...

Mass Storage Device Attached

Test Cases:

  210: Test File Functions - 10: file sharing policy

  211: Test File Functions - 11: invalid drive status - FR_NOT_ENABLED

  212: Test File Functions - 12: LFN disable

  213: Test File Functions - 13: _RS_PATH = 0

Test case 210: Test File Functions - 10: file sharing policy

Test case passed

Test case 211: Test File Functions - 11: invalid drive status - FR_NOT_ENABLED

Test case passed

Test case 212: Test File Functions - 12: LFN disable

Test case passed

Test case 213: Test File Functions - 13: _RS_PATH = 0

Test case passed

Test cases:

Executed: 4, Passed: 4, Failed: 0