CODE TIME TECHNOLOGIES

Abassi RTOS

FatFS Support

Disclaimer

Code Time Technologies Inc. provides this document “AS IS” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.
Code Time Technologies Inc. does not warrant that the contents of this document will meet your requirements or that the document is error-free. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. Code Time Technologies Inc. may make improvements and/or changes in the product(s) and/or program(s) described in the document at any time. This document does not imply a commitment by Code Time Technologies Inc. to supply or make generally available the product(s) described herein.

Table of Contents

List of Figures

List of Tables

1 Introduction

This document describes the support Abassi[1] [R1] (including mAbassi [R2] and µAbassi [R3]) offers to use the open source FatFS, the Generic FAT File System Module – ELM by ChaN [R4].

1.1 Distribution Contents

The set of files supplied with this distribution are listed in Table 1-1 below:

Table 1-1 Distribution

File Name	Description
`Abassi_FatFS_syscall.c`	Abassi OS control for FatFS
`diskio.c`	Target dependent driver interface

1.2 Limitations

Reminder: As stated in the FatFS documentation, the `f_mount(), f_mkfs(), f_fdisk()` functions are not re-entrant on the same volume. This is not a limitation due to Abassi but it is an intrinsic limitation of FatFS itself.

FatFS may not be supported right off the shelf for all target platforms / tool-chain Abassi / mAbassi/ µAbassi combinations. Code Time Technologies can provide porting help and/or work.

1.3 Features

The FatFS open source file system is fully supported by Abassi through the Code Time Technologies customized universal `syscall.c` (named `Abassi_FatFS_syscall.c`) file and the device / tool-chain specific `diskio.c` file.

1.4 Modifications

The FatFS files are provided unmodified exactly as they were downloaded from the site. The directory structure of the downloaded zip was re-arranged though. The original directory structure is as follows (in the root directory name, `n.nn` is the version specific number):

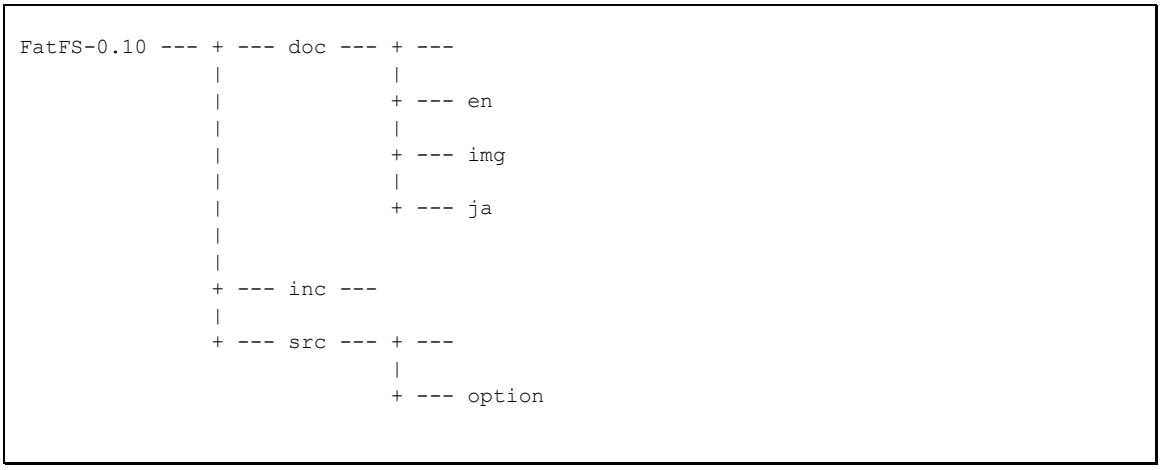
Table 1-2 Original FatFS directory structure

```
FatFS-n.nn --- + --- doc --- + ---
               |             |
               |             + --- en
               |             |
               |             + --- img
               |             |
               |             + --- ja
               |
               |
               + --- src --- + ---
                             |
                             + --- option
```

[1] When Abassi is mentioned in this document, unless explicitly stated, it always means Abassi, mAbassi and µAbassi.

In the distribution from Code Time Technologies, the include files (all files with a `.h` extension) that were originally in the `src` directory have been moved into the new `inc` directory as show in the following table:

Table 1-3 Distribution FatFS directory structure



Moving the include files (`diskio.h`, `ff.h`, `ffconf.h` and `integer.h`) into their own directory (`inc`) was deemed necessary because of the way the "C" standard defines the include file directory search sequence. When a file is included in "C", the pre-processor first looks into the same directory where the file that includes the other one is located. Because of this, using the same FatFS code when dealing with multiple projects may become an issue, as each project could require different configurations (the configuration is defined in the FatFS include files). An easy way to alleviate this problem is to use multiple copies of the FatFS package; but this is not ideal, as it does not lend itself to easy maintenance. The proper way to deal with multiple projects is to keep a single copy of the FatFS package and use the modified include files local to each project for the project specific configuration. This requires to not have the include files in the same directory as the FatFS source files and that is why the include files were moved into the new `inc` directory. All there is then to do is simply to properly specify the order of the search for the include files make the search to look into the project specific include before the `inc` directory of the FatFS package.

2 Target Set-up

All there is to do to configure and enable the use of the FatFS file system in an application based on Abassi is to include the following files in the build:

- `FatsFS-n.nn/src/ff.c`
- `Abassi/Abassi_FatFS_syscall.c`
- `FatsFS-n.nn/src/option/cc????.c`

Then customize FatFS for the project by setting the appropriate information in the FatFS standard include file `ffconf.h`, which should **not** be the original file in the FatFS directory tree but a copy in the project directory tree. Finally, set-up the include search directory order making sure the project own include paths are searched before `FatsFS-n.nn/inc`.

The customized `ffconf.h` file must include, either `Abassi.h`, `mAbassi.h` or `uAbassi.h`, depending on the RTOS package used.

2.1 Build Options

None.

3 Support

A single file is needed to have Abassi support the FatFS FAT file system: `Abassi_FatFS_syscall.c`. The code implemented is quite simple and straightforward. The key details are the following:

- `_VOLUMES`[2] mutexes could be created, as each volume requires its own mutex.
- As Abassi does not allow the deletion/destruction of a service, when a volume is un-mounted the mutex associated with this volume is held in a parking lot to be re-used upon the next mounting of the volume.
- The first volume mounted will trigger an initialization of the whole parking lot. This initialization is protected with the mutex `G_OSmutex` to eliminate the race condition where two tasks could try to initialize the parking lot at the same time.
- The grant request and release use a mutex. The FatFS timeout definition `_FS_TIMEOUT`[3] specified in OS timer tick units is used as the timeout argument in `MTXlock()`.
- If `malloc()` / `free()` are used (when `_USE_LFN == 3`), both calls will be protected with the mutex `G_OSmutex` if the "C" library does not protect or is not configured to protect against re-entrance.

[2] The `_VOLUMES` definition is an internal definition of FatFS specifying the maximum number of volumes that can be used.

[3] The `_FS_TIMEOUT` definition is located in the file `ffconf.c`.

4 Demos

Three demos using the FatFS file system are made available, when applicable. One demo is a tiny file system shell, alike a UNIX shell, but with very limited functionality. Two other cases are part of demos for the lwIP open-source TCP/IP stack.

4.1 lwIP (Demo #12 and #13)

Demo #12 and #13 are demos for lwIP (a lightweight TCP/IP stack) who's main purpose is to demonstrate the use of lwIP with Abassi. As such, although these demos use FatFS, they do not provide as much information on how to use FatFS as Demo #9 (see Section 4.2). For further information on Demo #12 and #13, refer to the lwIP support document [R5].

4.2 Mini Shell (Demo #9)

The mini shell demo is the most complete demo for FatFS, as it implements a UART based tiny UNIX shell-like interface; as such, this demo makes use of almost all the public functions of FatFS. To operate the demo, a terminal (or terminal emulator) must be connected to the UART and a SD/MMC mass storage device inserted on the board. Then, the first command to use is to mount the mass storage device on a volume with the following command:

```
mnt 0
```

Once the mass storage device is mounted, all commands of the tiny shell can be used on that volume.

The following sub-section describes very succinctly the available commands and their syntax. When a file name is needed (shown with the token `filename`) or a directory name (with token `dirname`), the file name can be specified as the name of a file in the current directory (e.g. `MyFile.txt`), or a file name with a relative path (.e.g `../MyFile.txt`), or a filename with an absolute path (e.g. `/MyDir/MyFile.txt`).

The command line prompt looks like:

```
/MyDir >
```

always displaying the current working directory.

4.2.1 cat

- Redirect a file

The `cat` command is used to send the contents of a file to the terminal:

```
cat filename
```

Or to redirect the data received from the terminal to a file:

```
cat >filename
```

The data will be transferred to the file until an end of file (EOF) character is received; the EOF character may be different depending on the run-time libraries (port). It is most likely Ctrl-D, Ctrl-Z, or 0xFF.

4.2.2 cd

- Change Directory

The `cd` command is used to change the current working directory:

```
cd dirname
```

Upon success, the prompt will show the new working current directory.

4.2.3 chmod

- Change a File Mode Access

To set a file / directory access to read-only:

```
chmod -w filename
```

To set a file / directory access to read-writeable:

```
chmod +w filename
```

To see the access mode of a file / directory, use the command `ls`.

4.2.4 cp

- Copy a file

To copy filename_1 to filename_2:

```
cp filename_1 filename_2
```

4.2.5 du

- Show the Disk Usage

Show the available storage, used storage and capacity of the volume of the current working directory. The output will be alike:

```
Disk size:  4294934528 bytes
Disk free:  4294746112 bytes
Disk used:      188416 bytes
SD size:    4294967296 bytes
SD blocks:        1024 bytes
```

4.2.6 fmt

- Format a volume

To format (initialize) volume #0:

```
fmt 0
```

In the demo, the number of byte per allocation units is left to be selected by FatFS.

4.2.7 help / ?

- Help Facility

Show all available commands:

```
help
```

or

```
?
```

Show the usage of a specific command:

```
help command
```

or

```
? command
```

4.2.8 ls

- List the Current Working Directory

List the contents of the current directory:

```
ls
```

The output will be alike:

```
drwx      32768 ..
 rwx    1532463 MyFile_1
 r-x     133464 MyFile_2
drwx      32768 MyDir
```

4.2.9 mkdir

- Make a New Directory

```
mkdir dirname
```

4.2.10 mnt

- Mount a physical storage device on a volume.

The association physical drive – volume is hard-coded. This is a FatFS restriction as the device driver interface does not offer remapping capability. Mounting volume 0:

```
mnt 0
```

4.2.11 mv

- Move (Rename) a File

Rename filename_1 to filename_2:

```
mv filename_1 filename_2
```

4.2.12 perf

- Throughput measurements

```
perf FileSize BufferSize
```

Write `FileSize` bytes to a file, using buffers of `BufferSize` bytes. Once the file is written, it is read back and then deleted. When specifying the number of bytes, the unit multipliers `M` for megabyte and `K` for kilobyte can be used.

The output for `perf 10M 4K` for example, will be alike:

```
10485760 bytes file using R/W block size of 4096 bytes
[  6.400s] Write rate  1638.400 kB/s
[  2.715s] Read rate   3862.158 kB/s
```

4.2.13 pwd

- Print the Current Working Directory

```
pwd
```

The output will be alike:

```
Current directory: /MyDir
```

4.2.14 rm

- Remove (Unlink/Delete) a File

```
rm filename
```

4.2.15 rmdir

- Remove (Unlink/Delete) a Directory

```
rmdir dirname
```

4.2.16 umnt

- Unmount a Volume

Unmount volume #0:

```
umnt 0
```

5 References

[R1] Abassi RTOS – User Guide, available at http://www.code-time.com
[R2] mAbassi RTOS – User Guide, available at http://www.code-time.com
[R3] µAbassi RTOS – User Guide, available at http://www.code-time.com
[R4] FatFS, the Generic FAT File System Module – ELM by ChaN (http://elm-chan.org/fsw/ff/00index_e.html)
[R5] Abassi – lwIP, available at http://www.code-time.com