**Group 1**

Propeller Clock Design

The Interim Report

An MEng. *Interim Report* Submitted for the

Masters of Engineering Electrical and Electronics Engineering

Supervised by: Prof. Guoping LIU

2015

Authors Name : **Martin Lewis Wynne-Jones**

**Tendai Madhlangove-Ngoni**

**Edward Porter**

**Miguel Santos**

**David George**

**Amin Goraine**

Venue: University of South Wales: Treforest Campus

(2015-2016 :-Academic Year)

Table of Contents

[CHAPTER 1 3](#_Toc437640401)

[**By: 14085887 Tendai Madhlangove-Ngoni** 3](#_Toc437640402)

[1.0 Introduction 3](#_Toc437640403)

[1.1 Health and safety 4](#_Toc437640404)

[1.0.2 Risk Assessment: 5](#_Toc437640405)

[1.0.3 Specific risks 5](#_Toc437640406)

[1.0.4 Risk Treatment 5](#_Toc437640407)

[CHAPTER 2 6](#_Toc437640408)

[**By Student 13061747 Martin Lewis Wynne-Jones** 6](#_Toc437640409)

[2.0. Project Planning Procedure 6](#_Toc437640410)

[2.1Gantt Chart 8](#_Toc437640411)

[2.2Team flow chart 9](#_Toc437640412)

[CHAPTER 3 10](#_Toc437640413)

[**By Student 14031329 Miguel Santos** 10](#_Toc437640414)

[3.1.0 Design Methodology 10](#_Toc437640415)

[3.1.1 Schematic Design 10](#_Toc437640416)

[**Schematic’s components –** 10](#_Toc437640417)

[3.1.2 Incremental Design 11](#_Toc437640418)

[voltage regulator dedicated for the LED driver. 12](#_Toc437640419)

[3.1.3 Results and prototypes 13](#_Toc437640420)

[**By Student 06218679 David George** 15](#_Toc437640421)

[3.2.0 PCB Design 15](#_Toc437640422)

[3.2.1 Component Selection 15](#_Toc437640423)

[3.2.2 Package Type Surface mount Vs through hole 16](#_Toc437640424)

[3.2.3 Shift Register vs LED Driver 16](#_Toc437640425)

[**By Student 14071207 Amin Goraine** 17](#_Toc437640426)

[3.3.0 Storing and Manipulating the Image 17](#_Toc437640427)

[3.3.1 Writing to a pixel into the image: 18](#_Toc437640428)

[3.3.2 Drawing the Image 19](#_Toc437640429)

[3.3.3 Animating the clock 20](#_Toc437640430)

[**By Student 14074362 Edward Porter** 21](#_Toc437640431)

[3.4.0 Initial designs 21](#_Toc437640432)

[3.4 .1. Design 1 22](#_Toc437640433)

[3.4.2 Design 2 22](#_Toc437640434)

[3.4.3 Design 3 23](#_Toc437640435)

[3.4.4 Computer Aided Design 24](#_Toc437640436)

[CHAPTER 4 27](#_Toc437640437)

[**This section was done by “All Members”** 27](#_Toc437640438)

[Conclusion 27](#_Toc437640439)

[References 28](#_Toc437640440)

[Bibliography 28](#_Toc437640441)

[Appendices 30](#_Toc437640442)

[Appendix 1: Health and safety Product standards Document 30](#_Toc437640443)

[Appendix 2 SD Card code (used for prototype Demo) 35](#_Toc437640444)

[Appendix 3 134](#_Toc437640445)

[Firmware algorithm: 134](#_Toc437640446)

## CHAPTER 1

## **By: 14085887 Tendai Madhlangove-Ngoni**

# 1.0 Introduction

In this project it is intended to design, simulate, build and program a propeller clock with the aim to improve on the design done previously. The product is to be displayed on the university show case. Currently the clock that has been designed has a big PCB board, which is mounted on a metallic pivot and has no guard which leaves all the components and the rotating board exposed. The health and safety of the user and the components is compromised hence it cannot be used continuously. The aesthetics of the clock if it were to be put on the market or on display by the interested part (University of South Wales) is also debatable. The current project looks into making the clock usable which on the other hand will result into it being marketable in light of the university standards of producing competent engineers and in the product being able to be sold and be used without causing harm. On the other hand it gives the group an opportunity for personal development in the engineering field as they explore and analyse different components, improve circuit designing, programming, problem solving and project management which, is the main aim of the interested part.

To achieve these objective online and other research method were used to determine the components to use in order to improve the product. Research was also carried out on the materials and design of the frame or housing of the clock and the health and safety statutory surrounding producing such a product. In addition to all the later calculations and programme to control the clock were written. However what has been done so far is only part of as more research and construction is still being carried out.

# 1.1 Health and safety

According to Strigini, (1994) stated, “Engineering judgement has an important role in safety or reliability assessment.” It is very crucial for an engineer to use clear judgment when a safety problem is at stake. The current clock has no guard. This is a health and safety issue in terms of:

* The user as it has no guard protecting the rotating part
* The components as they are exposed and the board they are mounted on rotates at high speed such that in the event of hitting any stray objects the components can be destroyed

The damage of the exposed components can result in the malfunction or completely no function of the clock which will result in more costs to be incurred to cater for replacement of components and time taken to repair.

The health and safety electrical product standards regulations (1994) in appendix 1 of this document, states that “All electrical equipment must be safe”, and goes on to list all the possible descriptions of what entails safety of equipment. This project took into consideration the requirements. Those that cannot be carried out such as the EMC testing to prove conformity to CE standard were noted for future reference in product manufacturing. The following health and safety areas were taken into consideration:

* Ergonomic machine operation,
* Assembly and packing
* Electricity
* Portable electrical equipment
* Common problems with flexible leads, their plugs and socket
* Noise ( Electromagnetic interference)
* Vibration which is due to the rotation
* Turning /Rotation since the clock has rotating a part

Literature was reviewed and discussions through meeting using Skype, emails and practical team meetings to address these issues. The risk assessment was carried out as stated in paragraph 1.0.2 which led to solutions highlighted in 1.0.4 to be implemented.

## 1.0.2 Risk Assessment:

The electronics circuits have very reduced rate of electrocution however the legislation was reviewed to confirm the risks involved [Stewart 2015]. On the other hand the clock has rotating a part which is a danger to the user and delicate components which need protection. Like in any other research, human errors and problems are always prone to occur. These may lead to other factors which may cause problems in this research.

## 1.0.3 Specific risks

The risks of working through this project were identified as the following:

1. Components can be damaged during soldering or handling
2. Electric shock during testing
3. During soldering burns can occur
4. Solder fumes may be inhaled
5. Eyes can be affected by long hours on the computer researching and programming
6. Other risks could include that the algorithms programmed on the programming environments

## 1.0.4 Risk Treatment

1. Using the appropriate heat and the correct soldering methods
2. Avoid touching live parts at any cost
3. The soldering iron was placed on the provided stands every time and alertness when lifting the iron using the insulated handle was exercised throughout the exercise
4. Taking Five minute every hour and also using proper lighting, comfortable chair and good seating position was employed.

## CHAPTER 2

# **By Student 13061747 Martin Lewis Wynne-Jones**

# 2.0. Project Planning Procedure

The first task was to break down the project and define the core basic features and the distinctive features. This was achieved by the designing a system flow chart to identify the main tasks, as shown in Figure 1 below. The distinctive features include: RGB display, picture to display and easy user interface.

**Figure 1**

Yes

Yes

**Render last requested display**

No

Yes

**Start**

**Power to Motor & PCB board**

**Begin system checks**

**Find start point using hall effect sensor**

**Has user changed the display?**

No

**Hall effect sensor triggered?**

**Wait**

**Has it been a long time?**

No

**Flash Error   
on LED’s and stop**

**Calibrate display using time taken**

**Begin LED  
flash test**

**Render default  
time display**

**Check Bluetooth communication**

**Render new image to display**

Once the task was broken down into smaller sections, a time goal was set for each part and a Gantt chart highlighted in Figure 3 on page 12 of this report, was used.The team also identified the strengths of members which made it easier to pair in a way that everyone benefits from each other in terms of support.

Each major task has two team members assigned and each member is in two sub-teams. This can be shown easily in the flow diagram, see Figure 2. However every member will be well informed of the other tasks through Github and skype meetings and everyone will learn from each other.

Figure 2

To make sure that each member is participating, the team must have meetings. Each week the entire team will discuss the issue. The issue should be related to the tasks currently ongoing on the Gantt chart and then recorded.

Each sub-team will have a smaller one on one meeting to disgust the aims, objectives, methods and action for that one task. This means that each member of the project will be in one main meeting and two smaller meetings though out the week.

The task completion will start with Research & development, Design & Simulate, Prototype & code, finalize production and conclude the project.

# 2.1Gantt Chart

Management of human resource is also the key to the accomplishment of a project and not just a tool kit of techniques. A Gantt chart is typically used to give an idea of where the task is currently at. Which Casterns *et al,* (2013) explains as “… showing facts against time”. This helps identify very quickly if there is an issue with the time needed to finish the whole project. Without a Gantt chart, the issues may quickly snowball before it is noticed and thus becoming a harder issue to fix.

Using a Gantt chart keeps everyone well informed on what they should be working on, what is next to work on and how much time is remaining. If an issue does occur, it can be spotted quickly and thus rectified before the knock on effect on the rest of the tasks causes major issues.

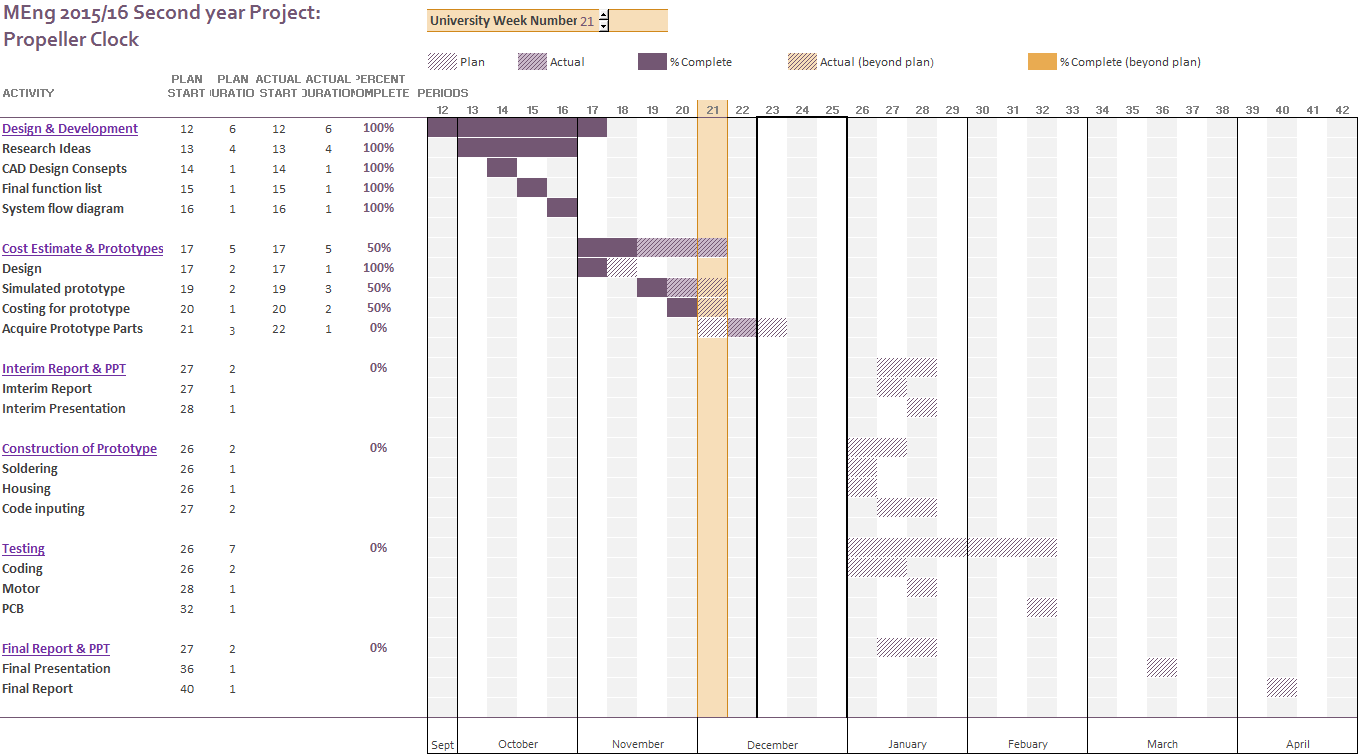


Figure 3

# 2.2Team flow chart

One main issue with any task that involves multiple stages is that parts of the production is waiting for the previous step to complete, thus wasting time which is already constricted in this case. Another issue is that members of the team may get stuck.

Using this method where two members are on one task, the team can brainstorm and attack the issue from multiple angles. It can also share the work load if it is an intensive task to complete.

This also protects the whole team if a team member falls ill or is unable to continue with the project as their will be someone still able to work on the task and is up to date. Thus removing the issue of time wasted to get someone new up to speed.

Once that task is complete, the next issue is that the member typically needs to wait for other team members to catch up or to finish their task before they can continue. Using this method, the person can continue with another task and keep the flow moving in the project.

# CHAPTER 3

# **By Student 14031329 Miguel Santos**

# 3.1.0 Design Methodology

This section discusses the design of the schematic (circuit), as well as the firmware and Bluetooth driver for desktop. More specifically, it was aimed for the propeller clock’s schematic design, which includes the abstract functionality (how all the components work together), the hardware choice as well as the engineering aspect (proper calculation of resistor/capacitor values and microcontroller speed). Moreover, the firmware for the microcontroller and desktop drivers were also a necessity for the project’s success.

# 3.1.1 Schematic Design

For the project to function safely and according to the specification, important design considerations had to be taken. Heavy research was made, followed by incremental design and simulating small parts of the circuit using electronics simulators (Proteus).

The research, brainstorming and simulator experimentation led to a final list of specifications that will be implemented (some are already implemented):

## **Schematic’s components –**

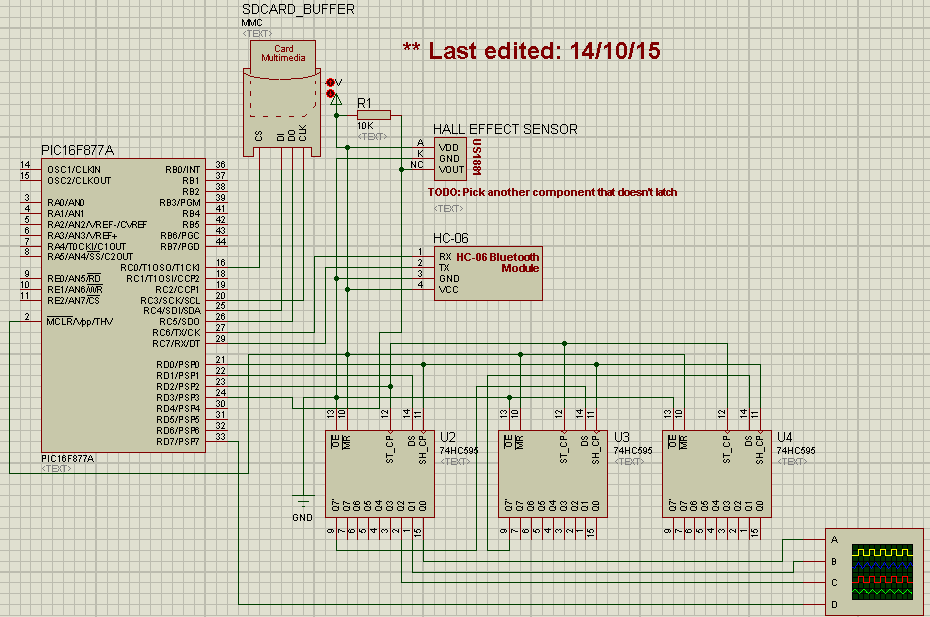
1. Microcontroller (ARM STM32F411 32-Bit 168 MHz)
2. LED Driver (with 32 Full colour RGB LEDs using PWM)
3. Bluetooth module (for communicating with a Computer and/or Smartphone)
4. SD Card (for storing large amounts of data transferred from the Computer)
5. Hall Effect sensor (tells the microcontroller the propeller’s position while rotating)
6. Voltage regulators (one for the microcontroller and another for the LED driver)

With these 6 components, it is possible to generate any kind of image, with a satisfying quality of 32 full colour RGB LEDs.

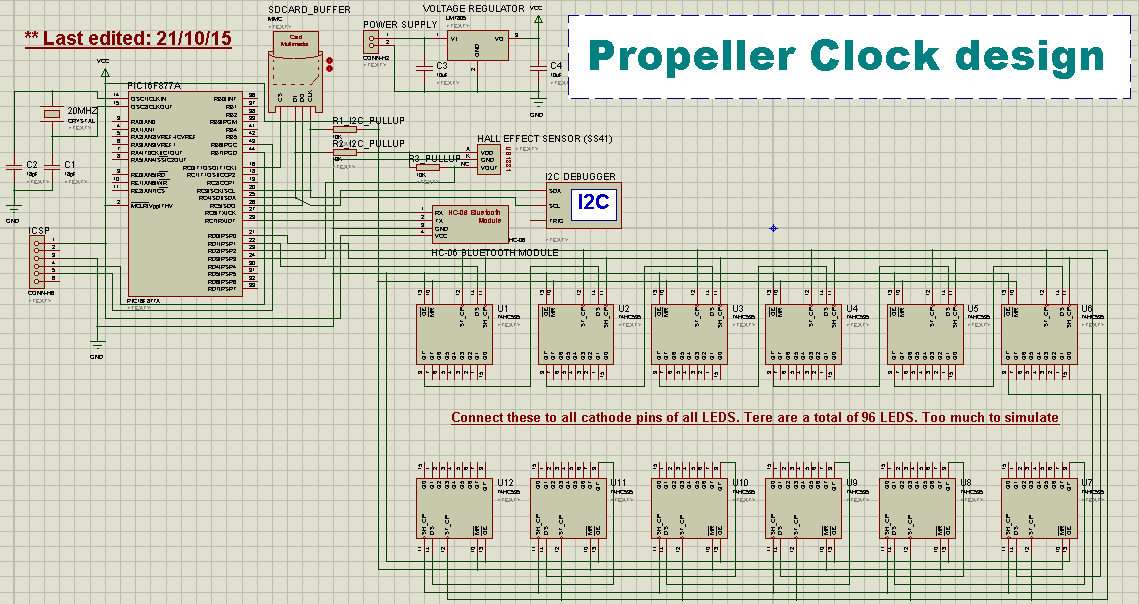
This design’s effectiveness is, however, dependent on the PCB layout and structure, as well as the image conversion and drawing from a Computer/Smartphone.

# 3.1.2 Incremental Design

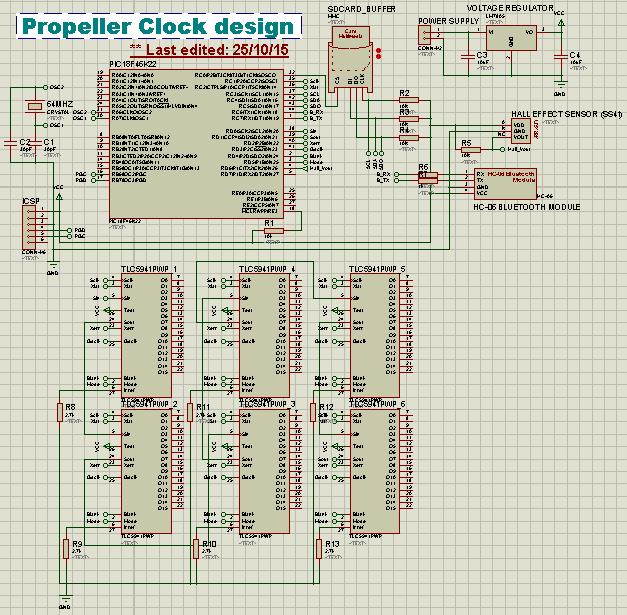
The schematic suffered many alterations since the start of the semester. The first version was an incomplete and poor design. It featured a PIC16 with a very slow clock, and a shift register 74HC595 for generating PWM for the LEDs. The SD Card, Bluetooth and Hall Effect sensors remained until the final design.



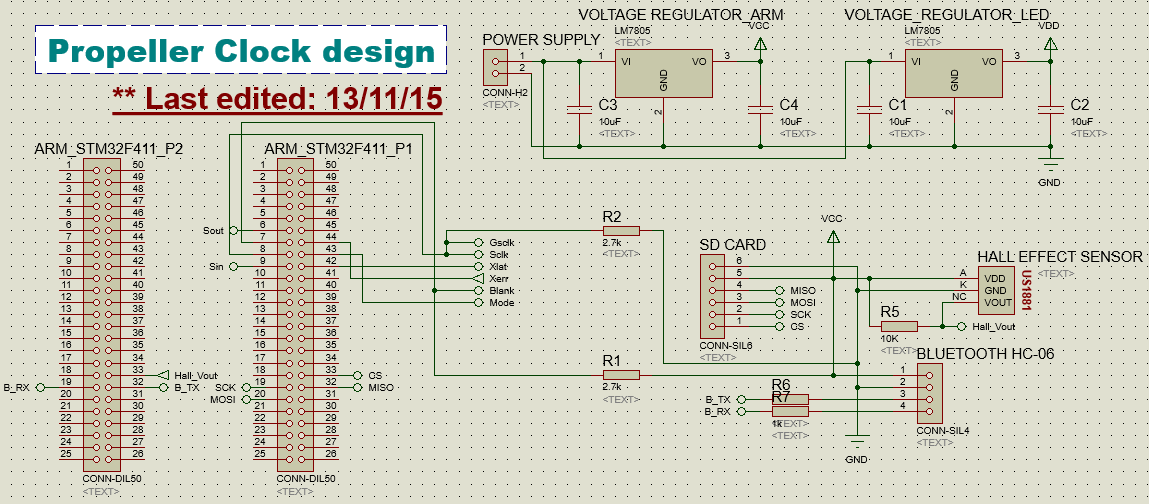
The second version was more complete and had a voltage regulator powering the PIC, however, it was still using shift registers and a PIC16.

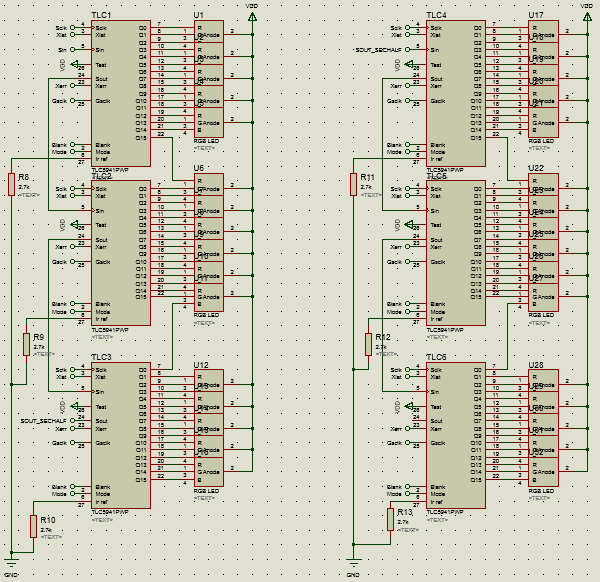


A third version was promptly developed, with a new PIC18 and a decent LED driver, the TLC5941PWP, which is a dedicated chip for controlling LEDs. Even after this change, our group figured we needed more than 64MHz of clock speed which the PIC18 had.



Finally, a complete and final version was made, featuring an ARM dev board STM32F411 Discovery with 168MHz with support for overclocking, and an extra

voltage regulator dedicated for the LED driver. 



# 3.1.3 Results and prototypes

After the design was completed, there was a need to continue development while the PCB wasn’t being manufactured, and also to prevent errors and to make sure the final design was actually functional. A prototype was put together piece by piece, component by component on a breadboard.

First, the LED driver was tested. Then, the drivers for it were developed and committed to the project’s code to be used in the actual board. Secondly, the SD Card was experimented. After it was working properly, it was also put together with the LED driver. Third, the Bluetooth module. As usual, it was tested, and put on the same program as the LED driver and SD card. And finally, the Hall Effect sensor was added, this one being very simple to test and control. As soon as all the separate components were tested and functional, they were all put together making the entire prototype.

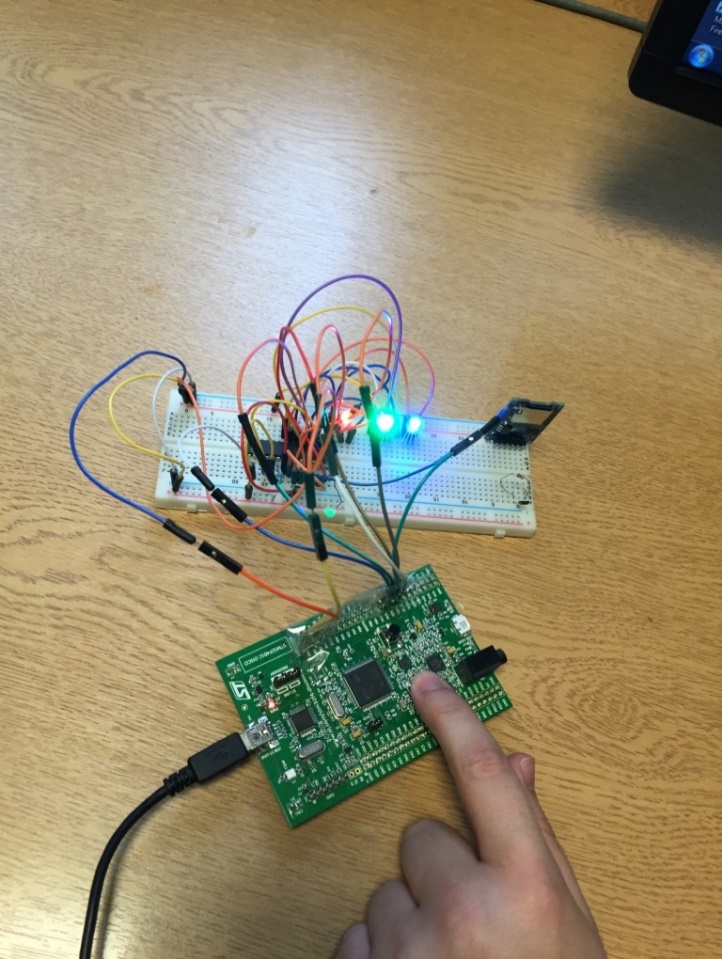
The code for the prototype has quite some complexity. The LED Driver code was fully developed by the group. The SD Card is a complex component by itself. It’s not just a simple piece of memory. It contains circuitry and works with multiple protocols and low level sector handling. This implies a filesystem must exist on it.

Due to the amount of difficult work needed to make the SD Card work (which is beyond the scope of this project), an open source library was obtained in order to utilise the SD Card. Its reference can be found in the references section of this report.

The Bluetooth module follows the same pattern as the SD Card. This one uses simple USART, however, since it’s a Bluetooth module, it supports command execution and has other functionality. Because of this, a library for it was also added, which can be found in the references section.

Finally, the Hall Effect sensor contains almost no code at all. It’s a very simple device, which outputs a digital signal whenever a magnet approaches it. This signal is simply read by the microcontroller.

All in all, these components have been attached together on a single breadboard, and the result can be seen in the following image:



Prototype functionality: The microcontroller initializes and waits for calibration to finish. Calibration is the act of triggering the Hall Effect sensor. This simulates the board rotating and reaching the bottom. Then, it proceeds to communicate via Bluetooth (putty can be used on a computer to see these messages) and enables the LEDs at the same time, simulating the board rotating and displaying the time/images/messages).

The program in the prototype was developed using the CooCox 1.7.8 IDE with GNU ARM GCC 4.9. Both C++ and C are being used.All in all, these are satisfactory results. The design is functional and there is less space for mistakes and errors from now on.

# **By Student 06218679 David George**

# 3.2.0 PCB Design

The operation of the product provides many challenges for the PCB designer. The PCB will need to be able to display the time and messages, while rotating at 1800 rpm. The board must be designed carefully. As the design of the board would have a direct impact on the balance, weight and top circumference speed of the display device.

For example if the led was 15cm long then the maximum size for the board would be 15cm in the longest direction from the centre. As using a square board would put too much additional weight in the design so a board of minimum width and length is required, as this would then put all of the weight on one side we then need to have a counter weight to balance the LED bar.

As the serial data line on the PCB will be running in the region of 30MHz there is a small potential for unwanted EMI spreading around the board. The suggested solution to the problem are to keep tracking lengths as short as possible keep traces at 2 x width apart and put over ground plane (TI)(NXP).The final restriction on size is the limit of production A4 size.

# 3.2.1 Component Selection

There are several points to think about size and weight, availability, suitability for the job, package type and cost. From the table above it can be seen that surface mount has more desirable property’s than through hole but will require careful package selection.

# 3.2.2 Package Type Surface mount Vs through hole

|  |  |  |
| --- | --- | --- |
| Consideration | Through Hole | Surface mount |
| Size | Large board Footprint | Small footprint |
| Weight | Slightly Heavier | Lighter Weight |
| Cost | Greater cost | Lower Cost |
| Ease to hand solder | Easy to solder | Package Dependant  Easy to impossible |
| Ease of rooting | Legs can be used as via | Via required to cross traces. |

Although using low profile low cost SMD LEDs appear to be the best way of meeting the low weight and size requirement. This would prevent another requirement for the project is that the display area is 40cm, to achieve this the LED bar needs to be 20cm, using 32 LEDs requires an LED of 6.25mm. Availability of SMD LED’s is poor and the cost per unit is greater than through hole.

The quantity of LEDs on the board require more outputs than the microchip can drive itself, so shift registers or LED drivers must be employed.

# 3.2.3 Shift Register vs LED Driver

|  |  |  |
| --- | --- | --- |
| **Consideration** | **Shift Register** | **LED driver** |
| Drive Current | Low drive current | Large drive currents |
| Current control | Resistor per LED required | Resistor per driver chip |
| Fine tune Current | No | Yes 6 bit correction |
| Different output level | No on or off | Yes 12 bit grey scale |
| Ease of driving | 1 bit per output | Requires data to be structured |
| Output Channels | Typically 8 | Typically 16 -32 |
| Cost | <£0.54 | >£1.50 |
| Error detection | No | Over temperature and open output |

**(TLC5941)**

As can be seen from the table above the LED driver is far more suited to driving LED’s although it requires more data per output forcing data buss to run at 30MHz it allows for far greater control of the display and error reporting.

With the complexity of the project selecting a suitable micro controller is an important item. As there is a data requirement of 1152 bits per frame and 72 frames an image 10.25Kbytes per image are required. The on-board memory requirements set a minimum of 30Kbytes 2 images and space for program. The arm M4 core allows enough on-board storage for several images and DMA allowing the processor to carry out other tasks as the display is updated as a background process (Yiu, 2013). Allowing time to load new images/data and keep track of time.

Initially the designee consisted of a single micro controller with several LEDs being driven. As the project requirement is to develop the board, to this end the number of LEDs have increases and changed from mono-colour LED to tri-colour RGB. This allows the display to become a full colour display. With the addition of external storage and Bluetooth module interfacing with external devices would allowing the display of an image. With the extra equipment required to facilitate the added features there are modules that cannot be directly incorporated some equipment to be routed via the display PCB.

# **By Student 14071207 Amin Goraine**

# 3.3.0 Storing and Manipulating the Image

The consequence of using a 12 bit led driver is that each pixel is stored as a 36 bit unsigned integer. This complicates the code to access the pixels slightly, since unfortunately C has not got a natural 36 bit type (Warren, 2013). This means that accessing and copying pixels is a little more complicated.  
  
The image is stored as a 2D array of 32 bit integers. With each frame being represented as an array 144 byte array of RGB.

R

G

B

R

G

B

R

G

B

…

12

36

48

24

60

72

1096

1128

1140

1152

Since an individual pixel cannot be simply accessed through indexing the array, it may only be read by masking and shifting the as shown in the pseudo-code below:

Uint64\_t GetPixelAt(Index):

offsetLowBit = (36 \* Index) % 32;

offsetHighBit = (36 \* (Index + 1)) % 32;

lowWordMask = ~mask(offsetHighBit);

highWordMask = mask(offsetLowBit);

wordIndex = floor(36 \* Index / 32);

lowPartPixel = imageArray[wordIndex] & lowWordMask;

highPartPixel = imageArray[wordIndex + 1] & highWordMask;

return lowPartPixel | highPartPixel << offsetLowBit;

# 3.3.1 Writing to a pixel into the image:

Uint64\_t SetPixelAt(Index):

offsetLowBit = (36 \* Index) % 32;

offsetHighBit = (36 \* (Index + 1)) % 32;

lowWordMask = ~mask(offsetHighBit);

highWordMask = mask(offsetLowBit);

wordIndex = floor(36 \* Index / 32);

imageArray[wordIndex] |= (lowWordMask & Pixel) << offsetLowBit;

imageArray[wordIndex + 1] |= (highWordMask & Pixel) << offsetHighBit

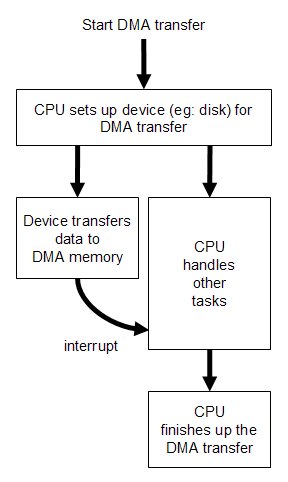
# 3.3.2 Drawing the Image

From the software perspective the LED driver works in 2 stages. First it sends the data to be displayed through the serial interface then the program has to pulse a GPIO port to latch the data and display the image.

The driver that was used in this project needs 12 bits per channel but requires that all 16 channels be update every time. Since there are 96 LEDs to update this means that 144 bytes are needed per segment and roughly 10kB per image.

With 72 segments to display, roughly 10kB will be required to store the image and needed to output the same to show the full image. To keep the refresh rate of the image 30 times a second to stream roughly 311kB per second are desirable. The frequency update needed to maintain 30 refresh was calculated at 2160 Hz. This means there is 0.46ms to draw each individual segment.

To maintain the high refresh rate while wasting CPU time. The image will be transferred to the LED drivers using DMA. (Tanenbaum, 1990).

The pseudo-code works as first it waits until it passes the Hall Effect segment. Then it loads next segment using DMA to the serial interface, when that is complete and the timer interrupt occurs it latches the LED driver output.

Interrupts are modelled as global flags to simplify pseudocode.

void drawLoop():

segmentIndex = numSegments;

DMA\_status = DMA\_incomplete;

while (true):

if (segmentIndex == numSegments):

DMA\_SendSegmentData (

fromImageMemory[SegmentIndex], toSPI);

if (DMA\_CompleteInterrupt):

DMA\_status = DMA\_complete;

if (HallEffect\_SensorInterrupt):

calculateNewTimerSpeed(segmentIndex);

segmentIndex = 0;

if (DrawCall\_TimerInterrupt):

DMA\_status = DMA\_waitForCompletionInterruptOr(20uS);

if (DMA\_status == DMA\_complete):

latchLedDriverOutput();

++segmentIndex;

# 3.3.3 Animating the clock

Since interrupts are used in the drawing system of the clock. Extra care must be taken to make sure that partial drawn/updated clock are not shown. Below is a data structure that should help make the animation for an analogue clock. This approach works by double buffering the clock image and maintaining separate clock handle animation data per buffer.

The approach chosen:

1. Initially Clock Buffer 1 is a complete clock set to be drawn and Clock Buffer 2 is the one that is going to be updated.
2. This means that Draw Buffer is being sent through SPI to the LED driver while Update has it clock handle updated.
3. When the update buffer update is completed, it will set ‘*shouldSwapBuffers’* to true
4. The next time the buffer complete a circle past the hall effect sensor the buffer roles will be swapped
5. Repeat steps 2-5.

Clock Image Buffer 1

Clock Image Buffer 2

Buffer to be drawn

Buffer to be updated

\*

\*

bShould

Swap

Temp Buffer

\*

len

rotation

Temp Buffer

\*

len

rotation

Clock handle animation data

# **By Student 14074362 Edward Porter**

# 3.4.0 Initial designs

In order to design the mechanical casing of the part the clock, the PCB size was taken into consideration so that the housing gives enough space for the rotating part. However while the size of the board was being determined research was carried out on the casing design. Sketches were drawn and a few designs were produced. The produced designs were discussed with the group to determine the advantages and disadvantages of each one of them. ‘Sketchup’ software was used as it allowed ease of drawing components of a part separately and bringing them together (Chopra A.*et al,* 2013).

# 3.4 .1. Design 1

Figure 1, was the initially designs as a possible housing for the clock. However its disadvantages outweighed the advantages and it was eventually dropped. The advantages of this system is that it is the safest of all options as having a frame around it as well as an acrylic screen in front of it will prevent fingers getting caught. The framed design is also more visually appealing than the others and creates a pseudo-screen effect with the acrylic in front. Conversely it has a higher cost of materials used to create the frame as well as increased complexity in building.

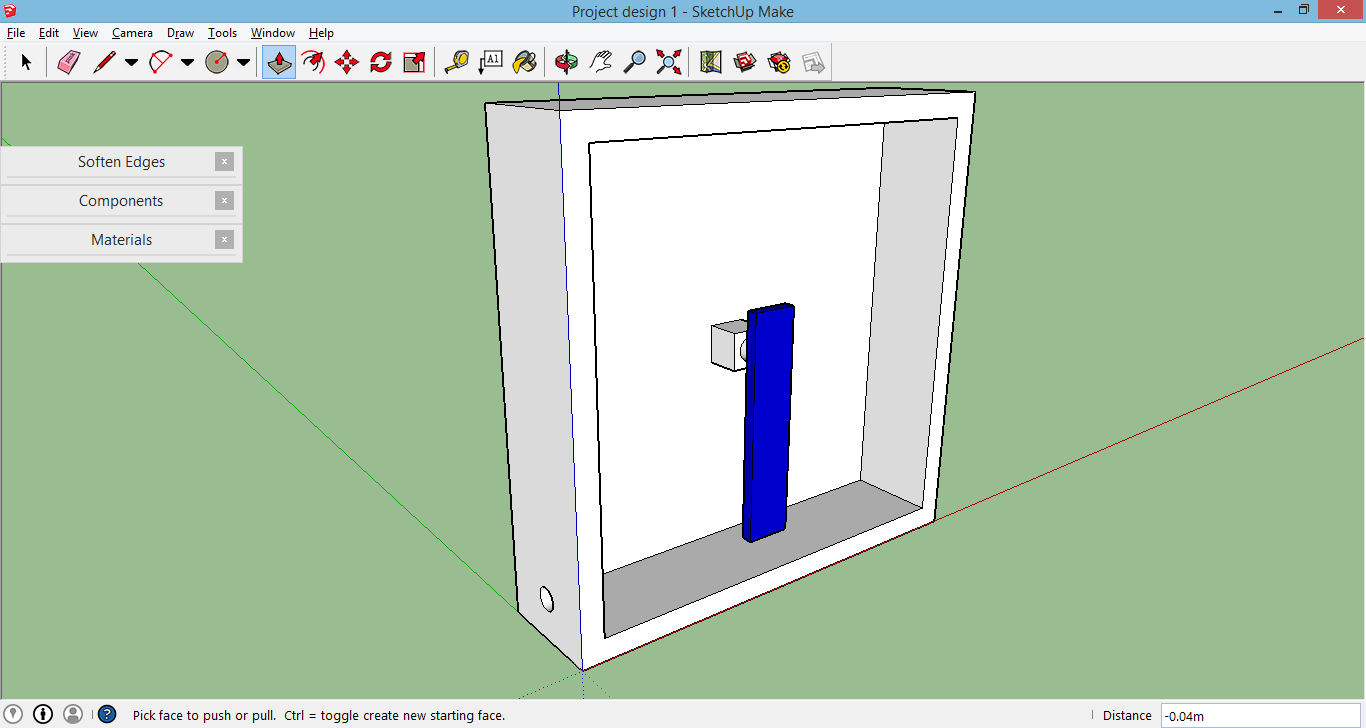
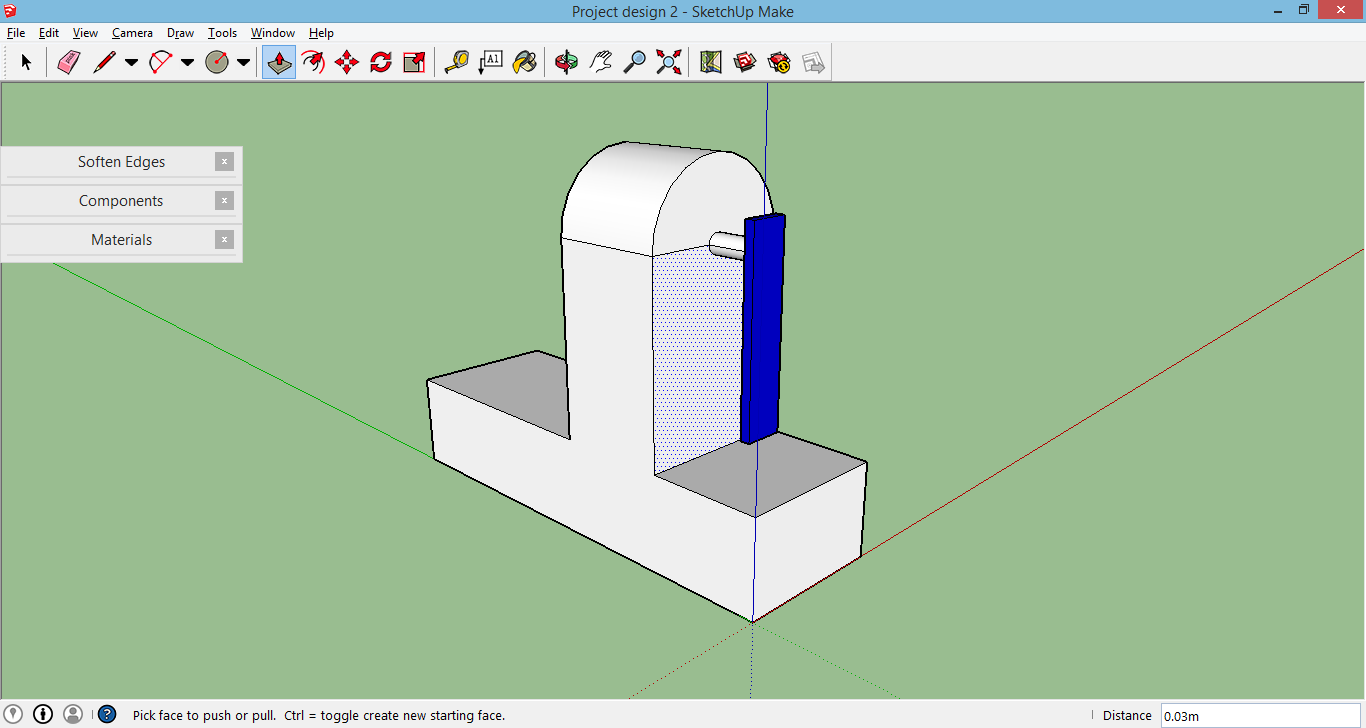


Figure 1

# 3.4.2 Design 2

Figure 2: This design is the sort of design that is most commonly seen from the models that have been designed by other designers previously. They have a simple inverted “T” cross profile and the blades situated at the highest point on the base. The advantage of this is the simplicity of the design reducing costs as well as having previous examples to look at. Contrariwise the unprotected fan blade as well having less aesthetic appeal compared to the first design becomes its disadvantage.



# 3.4.3 Design 3

This design was fairly complex, with a fan mounted on a rotating pillar in order to display the message all around it. The design was unusual and could have a niche application but the disadvantages outweigh the advantages. The twin rotation would put a lot of stress on the design as well as creating issues for power transmission, increasing price and complexity with two motors and the design being very unsafe from the fan rotating in effectively a sphere over the course of a full horizontal cycle.

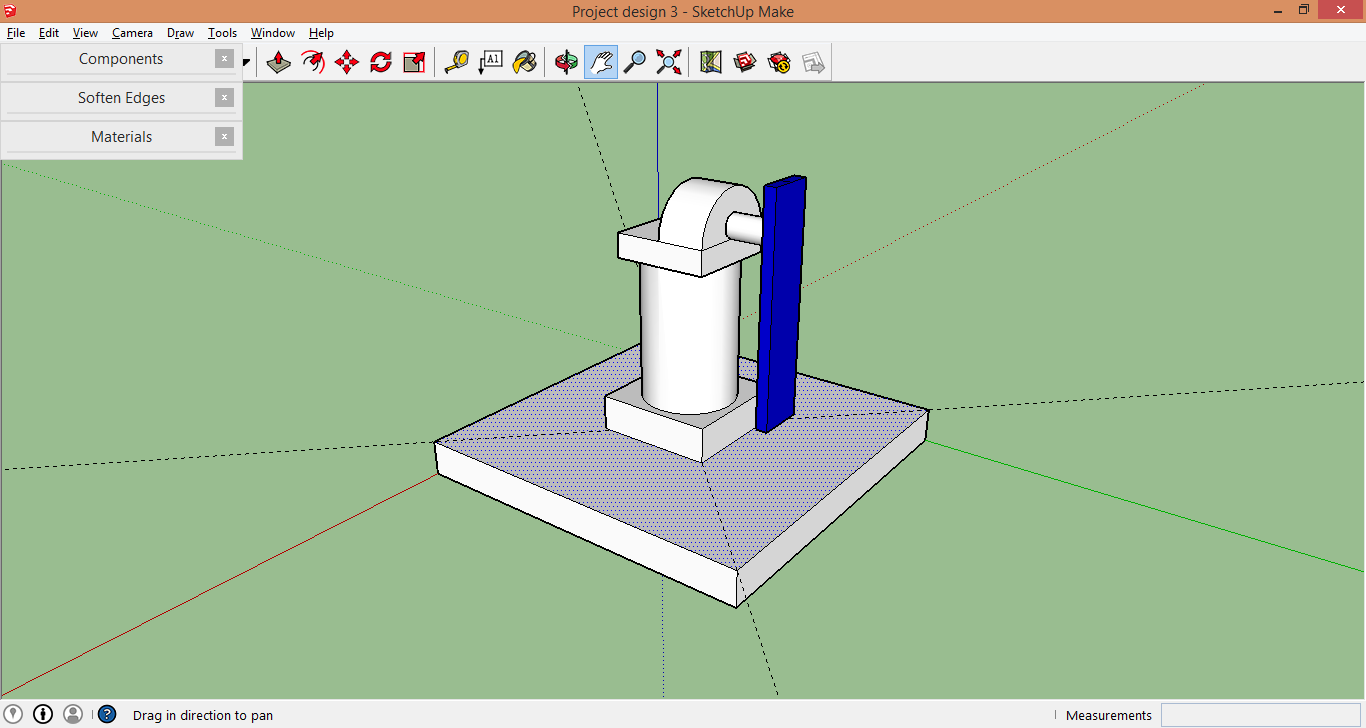


Figure 3

# 3.4.4 Computer Aided Design

SketchUp[1] was used to produce the models for the bracket to hold the boards together as a single fan blade, as well as the case for the fan. The software can be used for the 3D printing of actual designs, only needing a simple file converter to create ‘stl’ files capable of being printed on university printers. The ability to create separate components and manipulate them was a massive advantage, allowing rescaling of parts as well as easily showing the boards mounted and removed from the bracket.

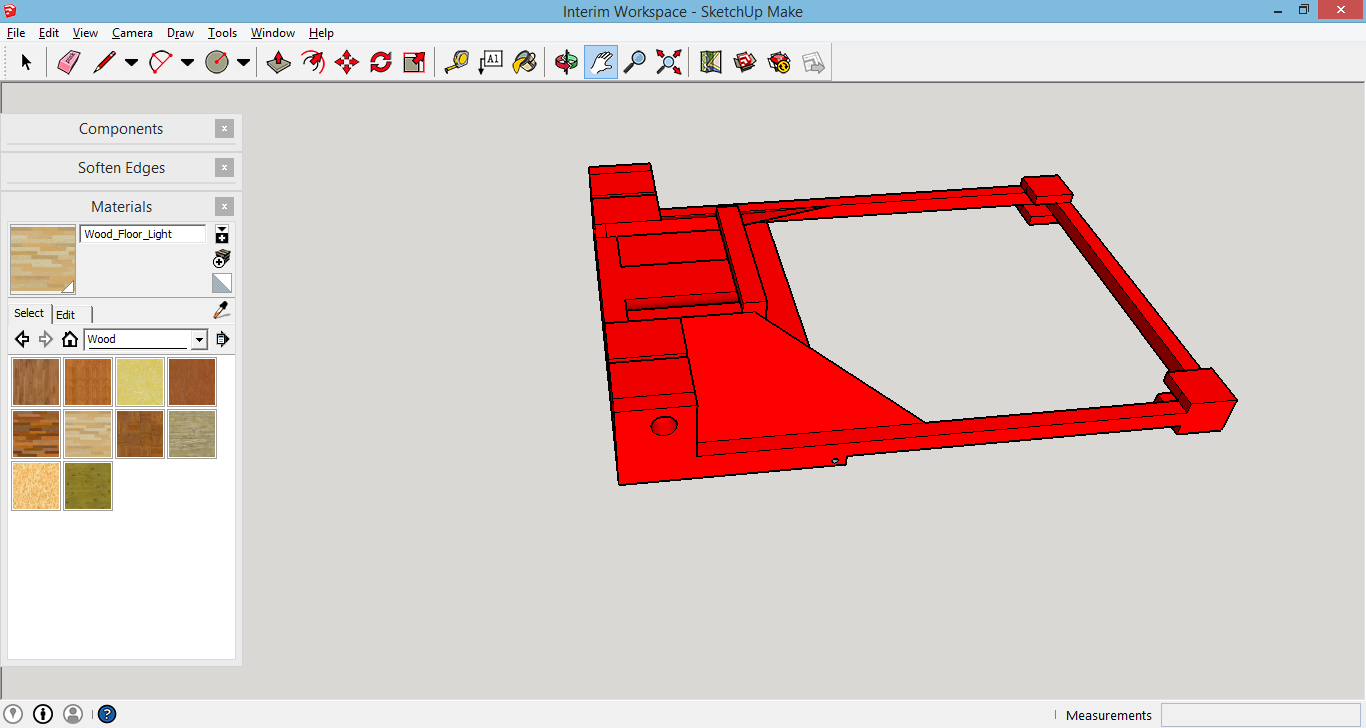


Figure 4: the mounting bracket for the ARM and LED boards

Figure 4 shows the basic design of the bracket to affix ARM and LED boards and the carbon brushes to be attached underneath after a suitable design is worked out to house them. The LED board will go on top of the ARM board slightly to save on material and reduce the maximum dimension of the board, the 3D printer is not very large and could struggle with the design if it were too big. Figure 5 shows (below) both boards mounted onto the bracket and figure 6 shows (below) both boards mounted and the method of affixing the LED board to the bracket. A thin sheet of steel is wrapped around the end of the PCB and fixed into the bracket with screws on either side, preventing the PCB flying out and breaking.

|  |  |
| --- | --- |
| Figure 5: with the Boards mounted | Figure 6: Boards mounted and fastening method |

The designs of the boards and the brackets are to scale so far the carbon brushes have not yet been added due to the delicate nature of aligning and no definite parts have been selected yet for this task.

Figure 7 shows the exploded view of the case assembly, standing at 480mmx480mmx107mm at the moment, showing the major components of the case minus the motor. The back of the case is reinforced with extra pieces of wood, to account for the extra strain and stress of having the motor mounted on it as well as holding up the fan blade. The frame of the case is made from 24mm thick wood, most likely pine. This frame will prevent people from getting things caught in the fan as well as allowing it to stand freely like a picture on a shelf. The acrylic in figure 7 will not be safety glass but a clear acrylic, 5mm thick to prevent things from getting caught in the fan as well as creating an aesthetically pleasing pseudo-screen effect.

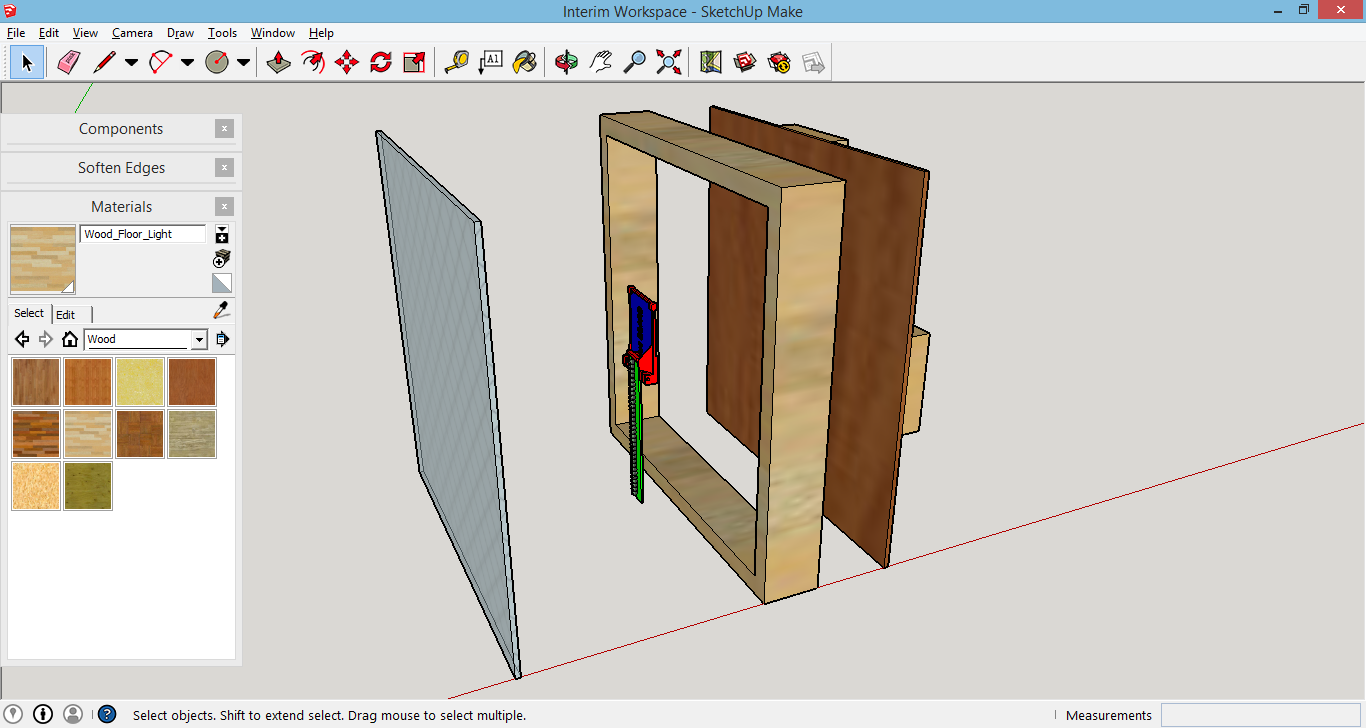


Figure 7: exploded view of the case assembly

# CHAPTER 4

# **This section was done by “All Members”**

# Conclusion

The design currently consists of, case assembly, which is 480mmx480mmx107mm, showing the major components of the case without the motor. The back of the case is reinforced with extra pieces of wood, to account for the extra strain and stress of having the motor mounted on it as well as holding up the fan blade. The frame of the case is made from 24mm thick wood, most likely pine. This frame will prevent people from getting things caught in the fan as well as allowing it to stand freely like a picture on a shelf. This was designed using SketchUp[1] was used to produce the models for the bracket to hold the boards together as a single fan blade, as well as the case for the fan.

It was considered that since interrupts are used in the drawing system of the clock, extra care must be taken to make sure that partial drawn/updated clock are not shown. The following approach was chosen so far:

1. Initially Clock Buffer 1 is a complete clock set to be drawn and Clock Buffer 2 is the one that is going to be updated.
2. This means that Draw Buffer is being sent through SPI to the LED driver while Update has it clock handle updated.
3. When the update buffer update is completed, it will set ‘*shouldSwapBuffers’* to true
4. The next time the buffer complete a circle past the hall effect sensor the buffer roles will be swapped

The PCB was designed and a prototype was run successfully which is a good sign that the project is heading in the positive direction. However the designed code is yet to be texted on the system and this can only be done when the circuit is built. With the current status the group has moved on to apply for more funding to produce a bigger clock as it is needed for the University display.

# References

1. *Andrew S. Tanenbaum (1990) Structured computer organization, Englewood Cliffs, N.J: Prentice-Hall.*
2. Casterns D.S*. et al* (2013) ‘Project management tools and Techniques: A practical Guide’ ,Taylor and Francis group LLC,London
3. Chopra A.*et al* (2013) ‘Introduction to Google SketchUp’ John Wiley & sons Inc, Toronto.
4. [*http://www.penguintutor.com/electronics/electrical-safety viewed on 18/10/2015*](http://www.penguintutor.com/electronics/electrical-safety%20viewed%20on%2018/10/2015)
5. *Health and Safety at Work Regulations 1999* [*http://www.legislation.gov.uk/uksi/1999/3242/contents/madeThe*](http://www.legislation.gov.uk/uksi/1999/3242/contents/madeThe) *Management of viewed on 18/10/2015*
6. *Joseph Yiu (2013) ‘The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors Newnes.’Elsevier Inc. Cambridge*
7. Semiconductors, NXP. 'SMPS EMC And Layout Guidlines'. *www.nxp.com*. N.p., 2011. <http://www.nxp.com/documents/application_note/AN10912.pdf> (viewed on 9 Dec. 2015.)
8. DATA sheet TLC5941 *Texas Instruments*. N.p., 2014.   
   <http://www.ti.com/lit/ds/symlink/tlc5941-q1.pdf> (viewed on 23 Nov. 2015)

# Bibliography

1. Hall Effect Sensor SS41 (<http://docs-europe.electrocomponents.com/webdocs/13e7/0900766b813e7972.pdf>
2. HC-06 (<https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf>)
3. Instruments, Texas. 'High Speed PCB La Yout Techniques'. *Texas Instruments*. N.p. http://www.ti.com/lit/ml/slyp173/slyp173.pdf

(viewed on 11/11/ 2015)

1. TLC5941PWP (<http://docs-europe.electrocomponents.com/webdocs/0a67/0900766b80a67c26.pdf>)
2. LM7805 (<http://docs-europe.electrocomponents.com/webdocs/137d/0900766b8137d675.pdf>
3. Majerle, T. (10 de October de 2014). *STM32F4 Discovery.com*. Obtido de STM32F4 Discovery.com: http://stm32f4-discovery.com/2014/07/library-21-read-sd-card-fatfs-stm32f4xx-devices/
4. Software, N. M. (s.d.). *Nelson Mandela Metropolitan University Control Software*. Obtido de Nelson Mandela Metropolitan University Control Software: http://controlsoft.nmmu.ac.za/STM32F4-Discovery-Board/Example-programs/HC-06-Bluetooth
5. STM32F411E (<http://docs-europe.electrocomponents.com/webdocs/1368/0900766b81368481.pdf>)
6. **Warren, H. (2013). *Hacker's delight*. Upper Saddle River, NJ: Addison-Wesley.**
7. Instruments, Texas. 'High Speed PCB La Yout Techniques'. *Texas*
8. *Instruments*. N.p., 2015.
9. http://www.ti.com/lit/ml/slyp173/slyp173.pdf
10. Web. 9 Dec. 2015.

# Appendices

# Appendix 1: Health and safety Product standards Document

**Health and Safety Executive**

Electricity

**How most accidents happen**

n Most accidents arise from contact with **live** conductors or equipment made **live** by faulty wiring and connections.

n Equipment using 240v AC may be as dangerous as that using 415v AC, depending on circumstances.

Each year, the use of electricity causes fatal and other injuries (eg burns) from electric shock and fire.

**Precautions** necessary to prevent accidents require everyone in the workshop (including the self-employed) to use equipment safely and co-operate with the employer where necessary.

***Employers in particular need to:***

n develop a suitable system of maintenance for both fixed installations and portable equipment – advice from a competent person may be required

n ensure that electrical contractors are competent for the work they are expected to do; for example, are they registered with such organisations as the National Inspection Council for Electrical Installation Contracting or equivalent?

n select equipment which is suitable for the job (using reduced low-voltage equipment lowers the risk of serious injury)

n check that wiring and equipment is sound and properly installed, especially so that protection equipment such as fuse and switchgear will operate adequately in the case of faults. Universally accepted standards are described in BS 7671 *Requirements for electrical installations*

n ensure any electricians working for you are competent. Levels of qualification established by the Electrical Joint Industries Board may be of help here, but check that qualifications are relevant; a fully qualified TV technician may not be competent to re-wire a building.

**The law\* on ‘live’ electrical work** *outlined*

No electrical work should be carried out ‘live’ where there is a possibility of contact with a potentially dangerous live conductor unless:

■■it is **unreasonable** in all the circumstances for the equipment to be dead; and

■■it is **reasonable** in all the circumstances for the work to be carried out on or near it while it is live; and

■■suitable precautions (including, where necessary, suitable protective equipment) have been taken to prevent injury, such as proper work planning and the use of adequately trained and supervised staff.

*\* See Key references.*

**Key references:** *Memorandum of guidance on the Electricity at Work*

*Regulations 1989* HSR25 HSE Books ISBN 0 7176 1602 9;

*Electrical safety and you* INDG231 (single copies free)Health and safety in engineering workshops Page 38 of 147 **Health and Safety Executive**

Portable electrical equipment

**Use this list to check whether you are managing the risks from portable electrical equipment. Have you:**

n prepared and implemented a system of maintenance for portable (and transportable) electrical equipment

n made sure that all items of portable electrical equipment are included in the maintenance system

n decided how to deal with ‘unauthorised’ equipment brought in by employees, eg portable, mains-operated radios

n collected information on where and how equipment is used. This information will help you decide what sort of checks/inspections/tests are required and at what frequencies

n provided straightforward training and information for all users (including yourself) to help them carry out user checks when equipment is used

n made sure that all equipment receives a formal visual inspection at appropriate intervals, carried out by someone who has been trained in what to look for

n considered preparing brief written guidance relating to visual inspection, what to look for, and procedures to follow when faults are found (and when unauthorised equipment is found)

n identified equipment which will need periodic combined inspection and testing

n established how often combined inspection and testing is appropriate (based on the type of equipment, how it is used, where it is used and the results of any previous checks/inspections/tests)

n appointed someone with the appropriate knowledge, training and experience to carry out the inspection and testing

n reviewed the results of checks/inspections/tests to identify any common trends and to confirm that the chosen inspection/test frequencies are appropriate

n made sure that the arrangements which you have made are being put into practice and that follow-up action is being taken?

**Remember that choosing the right equipment and carrying out visual inspections are perhaps the most important ways to minimise the risks.** Health and safety in engineering workshops Page 39 of 147 **Health and Safety Executive**

|  |  |
| --- | --- |
| **Activity** | **Preferred equipment** |
| Work inside metal tanks which may be damp or humid (through condensation, for example)  ‡ Work on apparatus where water may be present (on a metalworking fluid circulation system on a machine tool)  Work in close contact with metalwork in cool dry conditions  Ad hoc maintenance work and outside work in good weather | Use pneumatic power tools. Lighting, if essential, should be 25V dc CTE\* max and fixed out of reach.  Use equipment operating at voltages of 50V ac, 120V dc or less. Higher-voltage equipment should be built to waterproof standard, eg BS EN 60529: 1992 *Specification for degrees of protection provided by enclosures*.  The supply to earthed equipment should incorporate back-up protection which automatically disconnects the supply in the event of a fault, preferably earth monitoring.  Use equipment operating at voltages of 50v ac, 120v dc or less (110v ac CTE may be used in conjunction with all-insulated or double/insulated tools). Frequent cleaning of ventilation louvres is |

**Health and Safety Executive** Ergonomic machine operation, assembly and packing

To minimise risks

n avoid applying too much force for too long or too frequently using an awkward posture of the hand, wrist or arm.

Consider

***Reducing the force involved*** by, for example:

n keeping cutting tools sharp

n providing a larger gripping area

n supporting tools by means of balancers or tensioners (which may also help minimise vibration).

***Reducing highly repetitive movements*** by, for example:

n restructuring jobs so they contain more varied work

n automation.

***Reducing the need for awkward postures*** by, for example:

n changing the orientation of the part being worked upon to enable the wrist to be straight

n moving the operator so work is more comfortable.

***Training is vital***, particularly:

n to prevent the repetition of old, bad ways of working

n to protect newer employees from working too quickly too soon.

A modified handle design can lead to a more comfortable hand positionHealth and safety in engineering workshops Page 28 of 147 **Health and Safety Executive**

Most **ill health** arises from:

n prolonged forceful or repetitive gripping, twisting, reaching or other movement without enough rest or recovery. This causes pain, a restriction of joint movement and soft tissue swelling, leading sometimes to permanent disability, mostly in the hands, arms and shoulders

n lifting and handling goods unsafely by hand.

Consider workstations where both standing and sitting is possible. In general it is less physically stressful to sit, provided there are opportunities for moving around.

**Key reference:** *Upper limb disorders in the workplace*

HSG60 HSE Books ISBN 0 7176 1978 8 Health and safety in engineering workshops Page 29 of 147 **Health and Safety Executive**

Ergonomic aspects to consider

***For operating machinery***

n make frequently operated hand controls easy to reach and work

n if seats are provided leave sufficient room for knees and legs

n avoid bending and twisting to load/unload machines by having material/bins on racks which can be raised and lowered to keep them at waist height

n consider ‘lean-on’ or ‘sit-stand’, wheeled, sliding, suspended or fixed seats

n see pages 99–100 for normal chairs.

***For process or assembly work***

n arrange the work in a semi-circle

n provide a swivel chair

n bring the work to the worker

n consider chairs with forward-tilting seats and backrests to help workers reach further without loss of support, and consider work surfaces and component trays which slope towards the worker

n racks for material and finished items should be adjustable so that work is comfortable to reach and put away.

***For precision work***

n To avoid tense postures, a forward tilting seat and a work surface which slopes to the worker may be particularly helpful.

Sit-stand seat

Tilting seat

400

Zone of

convenient

reach

400

300

300

Normal

working

area

200

200

Foldaway seat

100

100

0

100

200

300

400

500

600

700

100

200

300

400

500

600

700

Horizontal reach –

distances in millimetres

**Key reference:** *Seating at work* HSG57 1997 HSE Books ISBN 0 7176 1231 7 Health and safety in engineering workshops Page 30 of 147

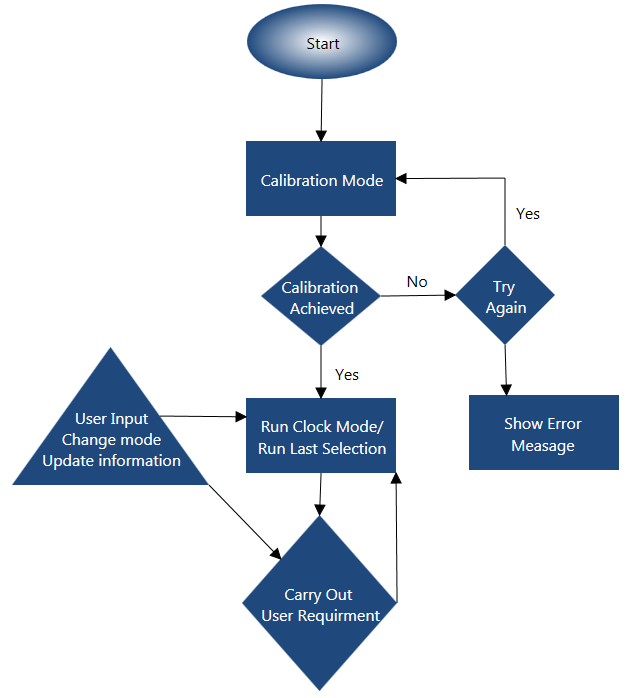
# Appendix 2 SD Card code (used for prototype Demo)

|  |
| --- |
|  |
|  | / FatFs - FAT file system module R0.11 (C)ChaN, 2015 |
|  | /-----------------------------------------------------------------------------/ |
|  | / FatFs module is a free software that opened under license policy of |
|  | / following conditions. |
|  | / |
|  | / Copyright (C) 2015, ChaN, all right reserved. |
|  | / |
|  | / 1. Redistributions of source code must retain the above copyright notice, |
|  | / this condition and the following disclaimer. |
|  | / |
|  | / This software is provided by the copyright holder and contributors "AS IS" |
|  | / and any warranties related to this software are DISCLAIMED. |
|  | / The copyright owner or contributors be NOT LIABLE for any damages caused |
|  | / by use of this software. |
|  | /----------------------------------------------------------------------------\*/ |
|  |  |
|  |  |
|  | #include "ff.h" /\* Declarations of FatFs API \*/ |
|  | #include "diskio.h" /\* Declarations of disk I/O functions \*/ |
|  |  |
|  |  |
|  | /\*-------------------------------------------------------------------------- |
|  |  |
|  | Module Private Definitions |
|  |  |
|  | ---------------------------------------------------------------------------\*/ |
|  |  |
|  | #if \_FATFS != 32020 /\* Revision ID \*/ |
|  | #error Wrong include file (ff.h). |
|  | #endif |
|  |  |
|  |  |
|  | /\* Reentrancy related \*/ |
|  | #if \_FS\_REENTRANT |
|  | #if \_USE\_LFN == 1 |
|  | #error Static LFN work area cannot be used at thread-safe configuration |
|  | #endif |
|  | #define ENTER\_FF(fs) { if (!lock\_fs(fs)) return FR\_TIMEOUT; } |
|  | #define LEAVE\_FF(fs, res) { unlock\_fs(fs, res); return res; } |
|  | #else |
|  | #define ENTER\_FF(fs) |
|  | #define LEAVE\_FF(fs, res) return res |
|  | #endif |
|  |  |
|  | #define ABORT(fs, res) { fp->err = (BYTE)(res); LEAVE\_FF(fs, res); } |
|  |  |
|  |  |
|  | /\* Definitions of sector size \*/ |
|  | #if (\_MAX\_SS < \_MIN\_SS) || (\_MAX\_SS != 512 && \_MAX\_SS != 1024 && \_MAX\_SS != 2048 && \_MAX\_SS != 4096) || (\_MIN\_SS != 512 && \_MIN\_SS != 1024 && \_MIN\_SS != 2048 && \_MIN\_SS != 4096) |
|  | #error Wrong sector size configuration |
|  | #endif |
|  | #if \_MAX\_SS == \_MIN\_SS |
|  | #define SS(fs) ((UINT)\_MAX\_SS) /\* Fixed sector size \*/ |
|  | #else |
|  | #define SS(fs) ((fs)->ssize) /\* Variable sector size \*/ |
|  | #endif |
|  |  |
|  |  |
|  | /\* Timestamp feature \*/ |
|  | #if \_FS\_NORTC == 1 |
|  | #if \_NORTC\_YEAR < 1980 || \_NORTC\_YEAR > 2107 || \_NORTC\_MON < 1 || \_NORTC\_MON > 12 || \_NORTC\_MDAY < 1 || \_NORTC\_MDAY > 31 |
|  | #error Invalid \_FS\_NORTC settings |
|  | #endif |
|  | #define GET\_FATTIME() ((DWORD)(\_NORTC\_YEAR - 1980) << 25 | (DWORD)\_NORTC\_MON << 21 | (DWORD)\_NORTC\_MDAY << 16) |
|  | #else |
|  | #define GET\_FATTIME() get\_fattime() |
|  | #endif |
|  |  |
|  |  |
|  | /\* File access control feature \*/ |
|  | #if \_FS\_LOCK |
|  | #if \_FS\_READONLY |
|  | #error \_FS\_LOCK must be 0 at read-only configuration |
|  | #endif |
|  | typedef struct { |
|  | FATFS \*fs; /\* Object ID 1, volume (NULL:blank entry) \*/ |
|  | DWORD clu; /\* Object ID 2, directory (0:root) \*/ |
|  | WORD idx; /\* Object ID 3, directory index \*/ |
|  | WORD ctr; /\* Object open counter, 0:none, 0x01..0xFF:read mode open count, 0x100:write mode \*/ |
|  | } FILESEM; |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  | /\* DBCS code ranges and SBCS extend character conversion table \*/ |
|  |  |
|  | #if \_CODE\_PAGE == 932 /\* Japanese Shift-JIS \*/ |
|  | #define \_DF1S 0x81 /\* DBC 1st byte range 1 start \*/ |
|  | #define \_DF1E 0x9F /\* DBC 1st byte range 1 end \*/ |
|  | #define \_DF2S 0xE0 /\* DBC 1st byte range 2 start \*/ |
|  | #define \_DF2E 0xFC /\* DBC 1st byte range 2 end \*/ |
|  | #define \_DS1S 0x40 /\* DBC 2nd byte range 1 start \*/ |
|  | #define \_DS1E 0x7E /\* DBC 2nd byte range 1 end \*/ |
|  | #define \_DS2S 0x80 /\* DBC 2nd byte range 2 start \*/ |
|  | #define \_DS2E 0xFC /\* DBC 2nd byte range 2 end \*/ |
|  |  |
|  | #elif \_CODE\_PAGE == 936 /\* Simplified Chinese GBK \*/ |
|  | #define \_DF1S 0x81 |
|  | #define \_DF1E 0xFE |
|  | #define \_DS1S 0x40 |
|  | #define \_DS1E 0x7E |
|  | #define \_DS2S 0x80 |
|  | #define \_DS2E 0xFE |
|  |  |
|  | #elif \_CODE\_PAGE == 949 /\* Korean \*/ |
|  | #define \_DF1S 0x81 |
|  | #define \_DF1E 0xFE |
|  | #define \_DS1S 0x41 |
|  | #define \_DS1E 0x5A |
|  | #define \_DS2S 0x61 |
|  | #define \_DS2E 0x7A |
|  | #define \_DS3S 0x81 |
|  | #define \_DS3E 0xFE |
|  |  |
|  | #elif \_CODE\_PAGE == 950 /\* Traditional Chinese Big5 \*/ |
|  | #define \_DF1S 0x81 |
|  | #define \_DF1E 0xFE |
|  | #define \_DS1S 0x40 |
|  | #define \_DS1E 0x7E |
|  | #define \_DS2S 0xA1 |
|  | #define \_DS2E 0xFE |
|  |  |
|  | #elif \_CODE\_PAGE == 437 /\* U.S. (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x9A,0x90,0x41,0x8E,0x41,0x8F,0x80,0x45,0x45,0x45,0x49,0x49,0x49,0x8E,0x8F,0x90,0x92,0x92,0x4F,0x99,0x4F,0x55,0x55,0x59,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0x41,0x49,0x4F,0x55,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 720 /\* Arabic (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x45,0x41,0x84,0x41,0x86,0x43,0x45,0x45,0x45,0x49,0x49,0x8D,0x8E,0x8F,0x90,0x92,0x92,0x93,0x94,0x95,0x49,0x49,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 737 /\* Greek (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x92,0x92,0x93,0x94,0x95,0x96,0x97,0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87, \ |
|  | 0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0xAA,0x92,0x93,0x94,0x95,0x96,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0x97,0xEA,0xEB,0xEC,0xE4,0xED,0xEE,0xE7,0xE8,0xF1,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 775 /\* Baltic (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x9A,0x91,0xA0,0x8E,0x95,0x8F,0x80,0xAD,0xED,0x8A,0x8A,0xA1,0x8D,0x8E,0x8F,0x90,0x92,0x92,0xE2,0x99,0x95,0x96,0x97,0x97,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0xA1,0xE0,0xA3,0xA3,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xB5,0xB6,0xB7,0xB8,0xBD,0xBE,0xC6,0xC7,0xA5,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE3,0xE8,0xE8,0xEA,0xEA,0xEE,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 850 /\* Multilingual Latin 1 (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0xDE,0x8E,0x8F,0x90,0x92,0x92,0xE2,0x99,0xE3,0xEA,0xEB,0x59,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE7,0xE9,0xEA,0xEB,0xED,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 852 /\* Latin 2 (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x9A,0x90,0xB6,0x8E,0xDE,0x8F,0x80,0x9D,0xD3,0x8A,0x8A,0xD7,0x8D,0x8E,0x8F,0x90,0x91,0x91,0xE2,0x99,0x95,0x95,0x97,0x97,0x99,0x9A,0x9B,0x9B,0x9D,0x9E,0x9F, \ |
|  | 0xB5,0xD6,0xE0,0xE9,0xA4,0xA4,0xA6,0xA6,0xA8,0xA8,0xAA,0x8D,0xAC,0xB8,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBD,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC6,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD1,0xD1,0xD2,0xD3,0xD2,0xD5,0xD6,0xD7,0xB7,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE3,0xD5,0xE6,0xE6,0xE8,0xE9,0xE8,0xEB,0xED,0xED,0xDD,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xEB,0xFC,0xFC,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 855 /\* Cyrillic (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x81,0x81,0x83,0x83,0x85,0x85,0x87,0x87,0x89,0x89,0x8B,0x8B,0x8D,0x8D,0x8F,0x8F,0x91,0x91,0x93,0x93,0x95,0x95,0x97,0x97,0x99,0x99,0x9B,0x9B,0x9D,0x9D,0x9F,0x9F, \ |
|  | 0xA1,0xA1,0xA3,0xA3,0xA5,0xA5,0xA7,0xA7,0xA9,0xA9,0xAB,0xAB,0xAD,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB6,0xB6,0xB8,0xB8,0xB9,0xBA,0xBB,0xBC,0xBE,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD1,0xD1,0xD3,0xD3,0xD5,0xD5,0xD7,0xD7,0xDD,0xD9,0xDA,0xDB,0xDC,0xDD,0xE0,0xDF, \ |
|  | 0xE0,0xE2,0xE2,0xE4,0xE4,0xE6,0xE6,0xE8,0xE8,0xEA,0xEA,0xEC,0xEC,0xEE,0xEE,0xEF,0xF0,0xF2,0xF2,0xF4,0xF4,0xF6,0xF6,0xF8,0xF8,0xFA,0xFA,0xFC,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 857 /\* Turkish (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0x98,0x8E,0x8F,0x90,0x92,0x92,0xE2,0x99,0xE3,0xEA,0xEB,0x98,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9E, \ |
|  | 0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA6,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xDE,0x59,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 858 /\* Multilingual Latin 1 + Euro (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0xDE,0x8E,0x8F,0x90,0x92,0x92,0xE2,0x99,0xE3,0xEA,0xEB,0x59,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD1,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE7,0xE9,0xEA,0xEB,0xED,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 862 /\* Hebrew (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0x41,0x49,0x4F,0x55,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 866 /\* Russian (OEM) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0x90,0x91,0x92,0x93,0x9d,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F,0xF0,0xF0,0xF2,0xF2,0xF4,0xF4,0xF6,0xF6,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 874 /\* Thai (OEM, Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 1250 /\* Central Europe (Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8C,0x8D,0x8E,0x8F, \ |
|  | 0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xA3,0xB4,0xB5,0xB6,0xB7,0xB8,0xA5,0xAA,0xBB,0xBC,0xBD,0xBC,0xAF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 1251 /\* Cyrillic (Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x82,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x80,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8C,0x8D,0x8E,0x8F, \ |
|  | 0xA0,0xA2,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB2,0xA5,0xB5,0xB6,0xB7,0xA8,0xB9,0xAA,0xBB,0xA3,0xBD,0xBD,0xAF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF} |
|  |  |
|  | #elif \_CODE\_PAGE == 1252 /\* Latin 1 (Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0xAd,0x9B,0x8C,0x9D,0xAE,0x9F, \ |
|  | 0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0x9F} |
|  |  |
|  | #elif \_CODE\_PAGE == 1253 /\* Greek (Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xA2,0xB8,0xB9,0xBA, \ |
|  | 0xE0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xF2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xFB,0xBC,0xFD,0xBF,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 1254 /\* Turkish (Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8C,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0x9F} |
|  |  |
|  | #elif \_CODE\_PAGE == 1255 /\* Hebrew (Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 1256 /\* Arabic (Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x8C,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0x41,0xE1,0x41,0xE3,0xE4,0xE5,0xE6,0x43,0x45,0x45,0x45,0x45,0xEC,0xED,0x49,0x49,0xF0,0xF1,0xF2,0xF3,0x4F,0xF5,0xF6,0xF7,0xF8,0x55,0xFA,0x55,0x55,0xFD,0xFE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 1257 /\* Baltic (Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xA8,0xB9,0xAA,0xBB,0xBC,0xBD,0xBE,0xAF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xFF} |
|  |  |
|  | #elif \_CODE\_PAGE == 1258 /\* Vietnam (OEM, Windows) \*/ |
|  | #define \_DF1S 0 |
|  | #define \_EXCVT {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0xAC,0x9D,0x9E,0x9F, \ |
|  | 0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF, \ |
|  | 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xEC,0xCD,0xCE,0xCF,0xD0,0xD1,0xF2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xFE,0x9F} |
|  |  |
|  | #elif \_CODE\_PAGE == 1 /\* ASCII (for only non-LFN cfg) \*/ |
|  | #if \_USE\_LFN |
|  | #error Cannot use LFN feature without valid code page. |
|  | #endif |
|  | #define \_DF1S 0 |
|  |  |
|  | #else |
|  | #error Unknown code page |
|  |  |
|  | #endif |
|  |  |
|  |  |
|  | /\* Character code support macros \*/ |
|  | #define IsUpper(c) (((c)>='A')&&((c)<='Z')) |
|  | #define IsLower(c) (((c)>='a')&&((c)<='z')) |
|  | #define IsDigit(c) (((c)>='0')&&((c)<='9')) |
|  |  |
|  | #if \_DF1S /\* Code page is DBCS \*/ |
|  |  |
|  | #ifdef \_DF2S /\* Two 1st byte areas \*/ |
|  | #define IsDBCS1(c) (((BYTE)(c) >= \_DF1S && (BYTE)(c) <= \_DF1E) || ((BYTE)(c) >= \_DF2S && (BYTE)(c) <= \_DF2E)) |
|  | #else /\* One 1st byte area \*/ |
|  | #define IsDBCS1(c) ((BYTE)(c) >= \_DF1S && (BYTE)(c) <= \_DF1E) |
|  | #endif |
|  |  |
|  | #ifdef \_DS3S /\* Three 2nd byte areas \*/ |
|  | #define IsDBCS2(c) (((BYTE)(c) >= \_DS1S && (BYTE)(c) <= \_DS1E) || ((BYTE)(c) >= \_DS2S && (BYTE)(c) <= \_DS2E) || ((BYTE)(c) >= \_DS3S && (BYTE)(c) <= \_DS3E)) |
|  | #else /\* Two 2nd byte areas \*/ |
|  | #define IsDBCS2(c) (((BYTE)(c) >= \_DS1S && (BYTE)(c) <= \_DS1E) || ((BYTE)(c) >= \_DS2S && (BYTE)(c) <= \_DS2E)) |
|  | #endif |
|  |  |
|  | #else /\* Code page is SBCS \*/ |
|  |  |
|  | #define IsDBCS1(c) 0 |
|  | #define IsDBCS2(c) 0 |
|  |  |
|  | #endif /\* \_DF1S \*/ |
|  |  |
|  |  |
|  | /\* Name status flags \*/ |
|  | #define NSFLAG 11 /\* Index of name status byte in fn[] \*/ |
|  | #define NS\_LOSS 0x01 /\* Out of 8.3 format \*/ |
|  | #define NS\_LFN 0x02 /\* Force to create LFN entry \*/ |
|  | #define NS\_LAST 0x04 /\* Last segment \*/ |
|  | #define NS\_BODY 0x08 /\* Lower case flag (body) \*/ |
|  | #define NS\_EXT 0x10 /\* Lower case flag (ext) \*/ |
|  | #define NS\_DOT 0x20 /\* Dot entry \*/ |
|  |  |
|  |  |
|  | /\* FAT sub-type boundaries (Differ from specs but correct for real DOS/Windows) \*/ |
|  | #define MIN\_FAT16 4086U /\* Minimum number of clusters as FAT16 \*/ |
|  | #define MIN\_FAT32 65526U /\* Minimum number of clusters as FAT32 \*/ |
|  |  |
|  |  |
|  | /\* FatFs refers the members in the FAT structures as byte array instead of |
|  | / structure member because the structure is not binary compatible between |
|  | / different platforms \*/ |
|  |  |
|  | #define BS\_jmpBoot 0 /\* x86 jump instruction (3) \*/ |
|  | #define BS\_OEMName 3 /\* OEM name (8) \*/ |
|  | #define BPB\_BytsPerSec 11 /\* Sector size [byte] (2) \*/ |
|  | #define BPB\_SecPerClus 13 /\* Cluster size [sector] (1) \*/ |
|  | #define BPB\_RsvdSecCnt 14 /\* Size of reserved area [sector] (2) \*/ |
|  | #define BPB\_NumFATs 16 /\* Number of FAT copies (1) \*/ |
|  | #define BPB\_RootEntCnt 17 /\* Number of root directory entries for FAT12/16 (2) \*/ |
|  | #define BPB\_TotSec16 19 /\* Volume size [sector] (2) \*/ |
|  | #define BPB\_Media 21 /\* Media descriptor (1) \*/ |
|  | #define BPB\_FATSz16 22 /\* FAT size [sector] (2) \*/ |
|  | #define BPB\_SecPerTrk 24 /\* Track size [sector] (2) \*/ |
|  | #define BPB\_NumHeads 26 /\* Number of heads (2) \*/ |
|  | #define BPB\_HiddSec 28 /\* Number of special hidden sectors (4) \*/ |
|  | #define BPB\_TotSec32 32 /\* Volume size [sector] (4) \*/ |
|  | #define BS\_DrvNum 36 /\* Physical drive number (2) \*/ |
|  | #define BS\_BootSig 38 /\* Extended boot signature (1) \*/ |
|  | #define BS\_VolID 39 /\* Volume serial number (4) \*/ |
|  | #define BS\_VolLab 43 /\* Volume label (8) \*/ |
|  | #define BS\_FilSysType 54 /\* File system type (1) \*/ |
|  | #define BPB\_FATSz32 36 /\* FAT size [sector] (4) \*/ |
|  | #define BPB\_ExtFlags 40 /\* Extended flags (2) \*/ |
|  | #define BPB\_FSVer 42 /\* File system version (2) \*/ |
|  | #define BPB\_RootClus 44 /\* Root directory first cluster (4) \*/ |
|  | #define BPB\_FSInfo 48 /\* Offset of FSINFO sector (2) \*/ |
|  | #define BPB\_BkBootSec 50 /\* Offset of backup boot sector (2) \*/ |
|  | #define BS\_DrvNum32 64 /\* Physical drive number (2) \*/ |
|  | #define BS\_BootSig32 66 /\* Extended boot signature (1) \*/ |
|  | #define BS\_VolID32 67 /\* Volume serial number (4) \*/ |
|  | #define BS\_VolLab32 71 /\* Volume label (8) \*/ |
|  | #define BS\_FilSysType32 82 /\* File system type (1) \*/ |
|  | #define FSI\_LeadSig 0 /\* FSI: Leading signature (4) \*/ |
|  | #define FSI\_StrucSig 484 /\* FSI: Structure signature (4) \*/ |
|  | #define FSI\_Free\_Count 488 /\* FSI: Number of free clusters (4) \*/ |
|  | #define FSI\_Nxt\_Free 492 /\* FSI: Last allocated cluster (4) \*/ |
|  | #define MBR\_Table 446 /\* MBR: Partition table offset (2) \*/ |
|  | #define SZ\_PTE 16 /\* MBR: Size of a partition table entry \*/ |
|  | #define BS\_55AA 510 /\* Signature word (2) \*/ |
|  |  |
|  | #define DIR\_Name 0 /\* Short file name (11) \*/ |
|  | #define DIR\_Attr 11 /\* Attribute (1) \*/ |
|  | #define DIR\_NTres 12 /\* Lower case flag (1) \*/ |
|  | #define DIR\_CrtTimeTenth 13 /\* Created time sub-second (1) \*/ |
|  | #define DIR\_CrtTime 14 /\* Created time (2) \*/ |
|  | #define DIR\_CrtDate 16 /\* Created date (2) \*/ |
|  | #define DIR\_LstAccDate 18 /\* Last accessed date (2) \*/ |
|  | #define DIR\_FstClusHI 20 /\* Higher 16-bit of first cluster (2) \*/ |
|  | #define DIR\_WrtTime 22 /\* Modified time (2) \*/ |
|  | #define DIR\_WrtDate 24 /\* Modified date (2) \*/ |
|  | #define DIR\_FstClusLO 26 /\* Lower 16-bit of first cluster (2) \*/ |
|  | #define DIR\_FileSize 28 /\* File size (4) \*/ |
|  | #define LDIR\_Ord 0 /\* LFN entry order and LLE flag (1) \*/ |
|  | #define LDIR\_Attr 11 /\* LFN attribute (1) \*/ |
|  | #define LDIR\_Type 12 /\* LFN type (1) \*/ |
|  | #define LDIR\_Chksum 13 /\* Sum of corresponding SFN entry \*/ |
|  | #define LDIR\_FstClusLO 26 /\* Must be zero (0) \*/ |
|  | #define SZ\_DIRE 32 /\* Size of a directory entry \*/ |
|  | #define LLEF 0x40 /\* Last long entry flag in LDIR\_Ord \*/ |
|  | #define DDEM 0xE5 /\* Deleted directory entry mark at DIR\_Name[0] \*/ |
|  | #define RDDEM 0x05 /\* Replacement of the character collides with DDEM \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*------------------------------------------------------------\*/ |
|  | /\* Module private work area \*/ |
|  | /\*------------------------------------------------------------\*/ |
|  | /\* Remark: Uninitialized variables with static duration are |
|  | / guaranteed zero/null at start-up. If not, either the linker |
|  | / or start-up routine being used is out of ANSI-C standard. |
|  | \*/ |
|  |  |
|  | #if \_VOLUMES < 1 || \_VOLUMES > 9 |
|  | #error Wrong \_VOLUMES setting |
|  | #endif |
|  | static FATFS \*FatFs[\_VOLUMES]; /\* Pointer to the file system objects (logical drives) \*/ |
|  | static WORD Fsid; /\* File system mount ID \*/ |
|  |  |
|  | #if \_FS\_RPATH && \_VOLUMES >= 2 |
|  | static BYTE CurrVol; /\* Current drive \*/ |
|  | #endif |
|  |  |
|  | #if \_FS\_LOCK |
|  | static FILESEM Files[\_FS\_LOCK]; /\* Open object lock semaphores \*/ |
|  | #endif |
|  |  |
|  | #if \_USE\_LFN == 0 /\* Non LFN feature \*/ |
|  | #define DEFINE\_NAMEBUF BYTE sfn[12] |
|  | #define INIT\_BUF(dobj) (dobj).fn = sfn |
|  | #define FREE\_BUF() |
|  | #else |
|  | #if \_MAX\_LFN < 12 || \_MAX\_LFN > 255 |
|  | #error Wrong \_MAX\_LFN setting |
|  | #endif |
|  | #if \_USE\_LFN == 1 /\* LFN feature with static working buffer \*/ |
|  | static WCHAR LfnBuf[\_MAX\_LFN + 1]; |
|  | #define DEFINE\_NAMEBUF BYTE sfn[12] |
|  | #define INIT\_BUF(dobj) { (dobj).fn = sfn; (dobj).lfn = LfnBuf; } |
|  | #define FREE\_BUF() |
|  | #elif \_USE\_LFN == 2 /\* LFN feature with dynamic working buffer on the stack \*/ |
|  | #define DEFINE\_NAMEBUF BYTE sfn[12]; WCHAR lbuf[\_MAX\_LFN + 1] |
|  | #define INIT\_BUF(dobj) { (dobj).fn = sfn; (dobj).lfn = lbuf; } |
|  | #define FREE\_BUF() |
|  | #elif \_USE\_LFN == 3 /\* LFN feature with dynamic working buffer on the heap \*/ |
|  | #define DEFINE\_NAMEBUF BYTE sfn[12]; WCHAR \*lfn |
|  | #define INIT\_BUF(dobj) { lfn = ff\_memalloc((\_MAX\_LFN + 1) \* 2); if (!lfn) LEAVE\_FF((dobj).fs, FR\_NOT\_ENOUGH\_CORE); (dobj).lfn = lfn; (dobj).fn = sfn; } |
|  | #define FREE\_BUF() ff\_memfree(lfn) |
|  | #else |
|  | #error Wrong \_USE\_LFN setting |
|  | #endif |
|  | #endif |
|  |  |
|  | #ifdef \_EXCVT |
|  | static const BYTE ExCvt[] = \_EXCVT; /\* Upper conversion table for extended characters \*/ |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-------------------------------------------------------------------------- |
|  |  |
|  | Module Private Functions |
|  |  |
|  | ---------------------------------------------------------------------------\*/ |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* String functions \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | /\* Copy memory to memory \*/ |
|  | static |
|  | void mem\_cpy (void\* dst, const void\* src, UINT cnt) { |
|  | BYTE \*d = (BYTE\*)dst; |
|  | const BYTE \*s = (const BYTE\*)src; |
|  |  |
|  | #if \_WORD\_ACCESS == 1 |
|  | while (cnt >= sizeof (int)) { |
|  | \*(int\*)d = \*(int\*)s; |
|  | d += sizeof (int); s += sizeof (int); |
|  | cnt -= sizeof (int); |
|  | } |
|  | #endif |
|  | while (cnt--) |
|  | \*d++ = \*s++; |
|  | } |
|  |  |
|  | /\* Fill memory \*/ |
|  | static |
|  | void mem\_set (void\* dst, int val, UINT cnt) { |
|  | BYTE \*d = (BYTE\*)dst; |
|  |  |
|  | while (cnt--) |
|  | \*d++ = (BYTE)val; |
|  | } |
|  |  |
|  | /\* Compare memory to memory \*/ |
|  | static |
|  | int mem\_cmp (const void\* dst, const void\* src, UINT cnt) { |
|  | const BYTE \*d = (const BYTE \*)dst, \*s = (const BYTE \*)src; |
|  | int r = 0; |
|  |  |
|  | while (cnt-- && (r = \*d++ - \*s++) == 0) ; |
|  | return r; |
|  | } |
|  |  |
|  | /\* Check if chr is contained in the string \*/ |
|  | static |
|  | int chk\_chr (const char\* str, int chr) { |
|  | while (\*str && \*str != chr) str++; |
|  | return \*str; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Request/Release grant to access the volume \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_FS\_REENTRANT |
|  | static |
|  | int lock\_fs ( |
|  | FATFS\* fs /\* File system object \*/ |
|  | ) |
|  | { |
|  | return ff\_req\_grant(fs->sobj); |
|  | } |
|  |  |
|  |  |
|  | static |
|  | void unlock\_fs ( |
|  | FATFS\* fs, /\* File system object \*/ |
|  | FRESULT res /\* Result code to be returned \*/ |
|  | ) |
|  | { |
|  | if (fs && |
|  | res != FR\_NOT\_ENABLED && |
|  | res != FR\_INVALID\_DRIVE && |
|  | res != FR\_INVALID\_OBJECT && |
|  | res != FR\_TIMEOUT) { |
|  | ff\_rel\_grant(fs->sobj); |
|  | } |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* File lock control functions \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_FS\_LOCK |
|  |  |
|  | static |
|  | FRESULT chk\_lock ( /\* Check if the file can be accessed \*/ |
|  | DIR\* dp, /\* Directory object pointing the file to be checked \*/ |
|  | int acc /\* Desired access type (0:Read, 1:Write, 2:Delete/Rename) \*/ |
|  | ) |
|  | { |
|  | UINT i, be; |
|  |  |
|  | /\* Search file semaphore table \*/ |
|  | for (i = be = 0; i < \_FS\_LOCK; i++) { |
|  | if (Files[i].fs) { /\* Existing entry \*/ |
|  | if (Files[i].fs == dp->fs && /\* Check if the object matched with an open object \*/ |
|  | Files[i].clu == dp->sclust && |
|  | Files[i].idx == dp->index) break; |
|  | } else { /\* Blank entry \*/ |
|  | be = 1; |
|  | } |
|  | } |
|  | if (i == \_FS\_LOCK) /\* The object is not opened \*/ |
|  | return (be || acc == 2) ? FR\_OK : FR\_TOO\_MANY\_OPEN\_FILES; /\* Is there a blank entry for new object? \*/ |
|  |  |
|  | /\* The object has been opened. Reject any open against writing file and all write mode open \*/ |
|  | return (acc || Files[i].ctr == 0x100) ? FR\_LOCKED : FR\_OK; |
|  | } |
|  |  |
|  |  |
|  | static |
|  | int enq\_lock (void) /\* Check if an entry is available for a new object \*/ |
|  | { |
|  | UINT i; |
|  |  |
|  | for (i = 0; i < \_FS\_LOCK && Files[i].fs; i++) ; |
|  | return (i == \_FS\_LOCK) ? 0 : 1; |
|  | } |
|  |  |
|  |  |
|  | static |
|  | UINT inc\_lock ( /\* Increment object open counter and returns its index (0:Internal error) \*/ |
|  | DIR\* dp, /\* Directory object pointing the file to register or increment \*/ |
|  | int acc /\* Desired access (0:Read, 1:Write, 2:Delete/Rename) \*/ |
|  | ) |
|  | { |
|  | UINT i; |
|  |  |
|  |  |
|  | for (i = 0; i < \_FS\_LOCK; i++) { /\* Find the object \*/ |
|  | if (Files[i].fs == dp->fs && |
|  | Files[i].clu == dp->sclust && |
|  | Files[i].idx == dp->index) break; |
|  | } |
|  |  |
|  | if (i == \_FS\_LOCK) { /\* Not opened. Register it as new. \*/ |
|  | for (i = 0; i < \_FS\_LOCK && Files[i].fs; i++) ; |
|  | if (i == \_FS\_LOCK) return 0; /\* No free entry to register (int err) \*/ |
|  | Files[i].fs = dp->fs; |
|  | Files[i].clu = dp->sclust; |
|  | Files[i].idx = dp->index; |
|  | Files[i].ctr = 0; |
|  | } |
|  |  |
|  | if (acc && Files[i].ctr) return 0; /\* Access violation (int err) \*/ |
|  |  |
|  | Files[i].ctr = acc ? 0x100 : Files[i].ctr + 1; /\* Set semaphore value \*/ |
|  |  |
|  | return i + 1; |
|  | } |
|  |  |
|  |  |
|  | static |
|  | FRESULT dec\_lock ( /\* Decrement object open counter \*/ |
|  | UINT i /\* Semaphore index (1..) \*/ |
|  | ) |
|  | { |
|  | WORD n; |
|  | FRESULT res; |
|  |  |
|  |  |
|  | if (--i < \_FS\_LOCK) { /\* Shift index number origin from 0 \*/ |
|  | n = Files[i].ctr; |
|  | if (n == 0x100) n = 0; /\* If write mode open, delete the entry \*/ |
|  | if (n) n--; /\* Decrement read mode open count \*/ |
|  | Files[i].ctr = n; |
|  | if (!n) Files[i].fs = 0; /\* Delete the entry if open count gets zero \*/ |
|  | res = FR\_OK; |
|  | } else { |
|  | res = FR\_INT\_ERR; /\* Invalid index nunber \*/ |
|  | } |
|  | return res; |
|  | } |
|  |  |
|  |  |
|  | static |
|  | void clear\_lock ( /\* Clear lock entries of the volume \*/ |
|  | FATFS \*fs |
|  | ) |
|  | { |
|  | UINT i; |
|  |  |
|  | for (i = 0; i < \_FS\_LOCK; i++) { |
|  | if (Files[i].fs == fs) Files[i].fs = 0; |
|  | } |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Move/Flush disk access window in the file system object \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if !\_FS\_READONLY |
|  | static |
|  | FRESULT sync\_window ( |
|  | FATFS\* fs /\* File system object \*/ |
|  | ) |
|  | { |
|  | DWORD wsect; |
|  | UINT nf; |
|  | FRESULT res = FR\_OK; |
|  |  |
|  |  |
|  | if (fs->wflag) { /\* Write back the sector if it is dirty \*/ |
|  | wsect = fs->winsect; /\* Current sector number \*/ |
|  | if (disk\_write(fs->drv, fs->win, wsect, 1) != RES\_OK) { |
|  | res = FR\_DISK\_ERR; |
|  | } else { |
|  | fs->wflag = 0; |
|  | if (wsect - fs->fatbase < fs->fsize) { /\* Is it in the FAT area? \*/ |
|  | for (nf = fs->n\_fats; nf >= 2; nf--) { /\* Reflect the change to all FAT copies \*/ |
|  | wsect += fs->fsize; |
|  | disk\_write(fs->drv, fs->win, wsect, 1); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | return res; |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  | static |
|  | FRESULT move\_window ( |
|  | FATFS\* fs, /\* File system object \*/ |
|  | DWORD sector /\* Sector number to make appearance in the fs->win[] \*/ |
|  | ) |
|  | { |
|  | FRESULT res = FR\_OK; |
|  |  |
|  |  |
|  | if (sector != fs->winsect) { /\* Window offset changed? \*/ |
|  | #if !\_FS\_READONLY |
|  | res = sync\_window(fs); /\* Write-back changes \*/ |
|  | #endif |
|  | if (res == FR\_OK) { /\* Fill sector window with new data \*/ |
|  | if (disk\_read(fs->drv, fs->win, sector, 1) != RES\_OK) { |
|  | sector = 0xFFFFFFFF; /\* Invalidate window if data is not reliable \*/ |
|  | res = FR\_DISK\_ERR; |
|  | } |
|  | fs->winsect = sector; |
|  | } |
|  | } |
|  | return res; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Synchronize file system and strage device \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if !\_FS\_READONLY |
|  | static |
|  | FRESULT sync\_fs ( /\* FR\_OK: successful, FR\_DISK\_ERR: failed \*/ |
|  | FATFS\* fs /\* File system object \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  |  |
|  |  |
|  | res = sync\_window(fs); |
|  | if (res == FR\_OK) { |
|  | /\* Update FSINFO sector if needed \*/ |
|  | if (fs->fs\_type == FS\_FAT32 && fs->fsi\_flag == 1) { |
|  | /\* Create FSINFO structure \*/ |
|  | mem\_set(fs->win, 0, SS(fs)); |
|  | ST\_WORD(fs->win + BS\_55AA, 0xAA55); |
|  | ST\_DWORD(fs->win + FSI\_LeadSig, 0x41615252); |
|  | ST\_DWORD(fs->win + FSI\_StrucSig, 0x61417272); |
|  | ST\_DWORD(fs->win + FSI\_Free\_Count, fs->free\_clust); |
|  | ST\_DWORD(fs->win + FSI\_Nxt\_Free, fs->last\_clust); |
|  | /\* Write it into the FSINFO sector \*/ |
|  | fs->winsect = fs->volbase + 1; |
|  | disk\_write(fs->drv, fs->win, fs->winsect, 1); |
|  | fs->fsi\_flag = 0; |
|  | } |
|  | /\* Make sure that no pending write process in the physical drive \*/ |
|  | if (disk\_ioctl(fs->drv, CTRL\_SYNC, 0) != RES\_OK) |
|  | res = FR\_DISK\_ERR; |
|  | } |
|  |  |
|  | return res; |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Get sector# from cluster# \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Hidden API for hacks and disk tools \*/ |
|  |  |
|  | DWORD clust2sect ( /\* !=0: Sector number, 0: Failed - invalid cluster# \*/ |
|  | FATFS\* fs, /\* File system object \*/ |
|  | DWORD clst /\* Cluster# to be converted \*/ |
|  | ) |
|  | { |
|  | clst -= 2; |
|  | if (clst >= fs->n\_fatent - 2) return 0; /\* Invalid cluster# \*/ |
|  | return clst \* fs->csize + fs->database; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* FAT access - Read value of a FAT entry \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Hidden API for hacks and disk tools \*/ |
|  |  |
|  | DWORD get\_fat ( /\* 0xFFFFFFFF:Disk error, 1:Internal error, 2..0x0FFFFFFF:Cluster status \*/ |
|  | FATFS\* fs, /\* File system object \*/ |
|  | DWORD clst /\* FAT index number (cluster number) to get the value \*/ |
|  | ) |
|  | { |
|  | UINT wc, bc; |
|  | BYTE \*p; |
|  | DWORD val; |
|  |  |
|  |  |
|  | if (clst < 2 || clst >= fs->n\_fatent) { /\* Check range \*/ |
|  | val = 1; /\* Internal error \*/ |
|  |  |
|  | } else { |
|  | val = 0xFFFFFFFF; /\* Default value falls on disk error \*/ |
|  |  |
|  | switch (fs->fs\_type) { |
|  | case FS\_FAT12 : |
|  | bc = (UINT)clst; bc += bc / 2; |
|  | if (move\_window(fs, fs->fatbase + (bc / SS(fs))) != FR\_OK) break; |
|  | wc = fs->win[bc++ % SS(fs)]; |
|  | if (move\_window(fs, fs->fatbase + (bc / SS(fs))) != FR\_OK) break; |
|  | wc |= fs->win[bc % SS(fs)] << 8; |
|  | val = clst & 1 ? wc >> 4 : (wc & 0xFFF); |
|  | break; |
|  |  |
|  | case FS\_FAT16 : |
|  | if (move\_window(fs, fs->fatbase + (clst / (SS(fs) / 2))) != FR\_OK) break; |
|  | p = &fs->win[clst \* 2 % SS(fs)]; |
|  | val = LD\_WORD(p); |
|  | break; |
|  |  |
|  | case FS\_FAT32 : |
|  | if (move\_window(fs, fs->fatbase + (clst / (SS(fs) / 4))) != FR\_OK) break; |
|  | p = &fs->win[clst \* 4 % SS(fs)]; |
|  | val = LD\_DWORD(p) & 0x0FFFFFFF; |
|  | break; |
|  |  |
|  | default: |
|  | val = 1; /\* Internal error \*/ |
|  | } |
|  | } |
|  |  |
|  | return val; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* FAT access - Change value of a FAT entry \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Hidden API for hacks and disk tools \*/ |
|  |  |
|  | #if !\_FS\_READONLY |
|  | FRESULT put\_fat ( |
|  | FATFS\* fs, /\* File system object \*/ |
|  | DWORD clst, /\* FAT index number (cluster number) to be changed \*/ |
|  | DWORD val /\* New value to be set to the entry \*/ |
|  | ) |
|  | { |
|  | UINT bc; |
|  | BYTE \*p; |
|  | FRESULT res; |
|  |  |
|  |  |
|  | if (clst < 2 || clst >= fs->n\_fatent) { /\* Check range \*/ |
|  | res = FR\_INT\_ERR; |
|  |  |
|  | } else { |
|  | switch (fs->fs\_type) { |
|  | case FS\_FAT12 : |
|  | bc = (UINT)clst; bc += bc / 2; |
|  | res = move\_window(fs, fs->fatbase + (bc / SS(fs))); |
|  | if (res != FR\_OK) break; |
|  | p = &fs->win[bc++ % SS(fs)]; |
|  | \*p = (clst & 1) ? ((\*p & 0x0F) | ((BYTE)val << 4)) : (BYTE)val; |
|  | fs->wflag = 1; |
|  | res = move\_window(fs, fs->fatbase + (bc / SS(fs))); |
|  | if (res != FR\_OK) break; |
|  | p = &fs->win[bc % SS(fs)]; |
|  | \*p = (clst & 1) ? (BYTE)(val >> 4) : ((\*p & 0xF0) | ((BYTE)(val >> 8) & 0x0F)); |
|  | fs->wflag = 1; |
|  | break; |
|  |  |
|  | case FS\_FAT16 : |
|  | res = move\_window(fs, fs->fatbase + (clst / (SS(fs) / 2))); |
|  | if (res != FR\_OK) break; |
|  | p = &fs->win[clst \* 2 % SS(fs)]; |
|  | ST\_WORD(p, (WORD)val); |
|  | fs->wflag = 1; |
|  | break; |
|  |  |
|  | case FS\_FAT32 : |
|  | res = move\_window(fs, fs->fatbase + (clst / (SS(fs) / 4))); |
|  | if (res != FR\_OK) break; |
|  | p = &fs->win[clst \* 4 % SS(fs)]; |
|  | val |= LD\_DWORD(p) & 0xF0000000; |
|  | ST\_DWORD(p, val); |
|  | fs->wflag = 1; |
|  | break; |
|  |  |
|  | default : |
|  | res = FR\_INT\_ERR; |
|  | } |
|  | } |
|  |  |
|  | return res; |
|  | } |
|  | #endif /\* !\_FS\_READONLY \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* FAT handling - Remove a cluster chain \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if !\_FS\_READONLY |
|  | static |
|  | FRESULT remove\_chain ( |
|  | FATFS\* fs, /\* File system object \*/ |
|  | DWORD clst /\* Cluster# to remove a chain from \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DWORD nxt; |
|  | #if \_USE\_TRIM |
|  | DWORD scl = clst, ecl = clst, rt[2]; |
|  | #endif |
|  |  |
|  | if (clst < 2 || clst >= fs->n\_fatent) { /\* Check range \*/ |
|  | res = FR\_INT\_ERR; |
|  |  |
|  | } else { |
|  | res = FR\_OK; |
|  | while (clst < fs->n\_fatent) { /\* Not a last link? \*/ |
|  | nxt = get\_fat(fs, clst); /\* Get cluster status \*/ |
|  | if (nxt == 0) break; /\* Empty cluster? \*/ |
|  | if (nxt == 1) { res = FR\_INT\_ERR; break; } /\* Internal error? \*/ |
|  | if (nxt == 0xFFFFFFFF) { res = FR\_DISK\_ERR; break; } /\* Disk error? \*/ |
|  | res = put\_fat(fs, clst, 0); /\* Mark the cluster "empty" \*/ |
|  | if (res != FR\_OK) break; |
|  | if (fs->free\_clust != 0xFFFFFFFF) { /\* Update FSINFO \*/ |
|  | fs->free\_clust++; |
|  | fs->fsi\_flag |= 1; |
|  | } |
|  | #if \_USE\_TRIM |
|  | if (ecl + 1 == nxt) { /\* Is next cluster contiguous? \*/ |
|  | ecl = nxt; |
|  | } else { /\* End of contiguous clusters \*/ |
|  | rt[0] = clust2sect(fs, scl); /\* Start sector \*/ |
|  | rt[1] = clust2sect(fs, ecl) + fs->csize - 1; /\* End sector \*/ |
|  | disk\_ioctl(fs->drv, CTRL\_TRIM, rt); /\* Erase the block \*/ |
|  | scl = ecl = nxt; |
|  | } |
|  | #endif |
|  | clst = nxt; /\* Next cluster \*/ |
|  | } |
|  | } |
|  |  |
|  | return res; |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* FAT handling - Stretch or Create a cluster chain \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if !\_FS\_READONLY |
|  | static |
|  | DWORD create\_chain ( /\* 0:No free cluster, 1:Internal error, 0xFFFFFFFF:Disk error, >=2:New cluster# \*/ |
|  | FATFS\* fs, /\* File system object \*/ |
|  | DWORD clst /\* Cluster# to stretch. 0 means create a new chain. \*/ |
|  | ) |
|  | { |
|  | DWORD cs, ncl, scl; |
|  | FRESULT res; |
|  |  |
|  |  |
|  | if (clst == 0) { /\* Create a new chain \*/ |
|  | scl = fs->last\_clust; /\* Get suggested start point \*/ |
|  | if (!scl || scl >= fs->n\_fatent) scl = 1; |
|  | } |
|  | else { /\* Stretch the current chain \*/ |
|  | cs = get\_fat(fs, clst); /\* Check the cluster status \*/ |
|  | if (cs < 2) return 1; /\* Invalid value \*/ |
|  | if (cs == 0xFFFFFFFF) return cs; /\* A disk error occurred \*/ |
|  | if (cs < fs->n\_fatent) return cs; /\* It is already followed by next cluster \*/ |
|  | scl = clst; |
|  | } |
|  |  |
|  | ncl = scl; /\* Start cluster \*/ |
|  | for (;;) { |
|  | ncl++; /\* Next cluster \*/ |
|  | if (ncl >= fs->n\_fatent) { /\* Check wrap around \*/ |
|  | ncl = 2; |
|  | if (ncl > scl) return 0; /\* No free cluster \*/ |
|  | } |
|  | cs = get\_fat(fs, ncl); /\* Get the cluster status \*/ |
|  | if (cs == 0) break; /\* Found a free cluster \*/ |
|  | if (cs == 0xFFFFFFFF || cs == 1)/\* An error occurred \*/ |
|  | return cs; |
|  | if (ncl == scl) return 0; /\* No free cluster \*/ |
|  | } |
|  |  |
|  | res = put\_fat(fs, ncl, 0x0FFFFFFF); /\* Mark the new cluster "last link" \*/ |
|  | if (res == FR\_OK && clst != 0) { |
|  | res = put\_fat(fs, clst, ncl); /\* Link it to the previous one if needed \*/ |
|  | } |
|  | if (res == FR\_OK) { |
|  | fs->last\_clust = ncl; /\* Update FSINFO \*/ |
|  | if (fs->free\_clust != 0xFFFFFFFF) { |
|  | fs->free\_clust--; |
|  | fs->fsi\_flag |= 1; |
|  | } |
|  | } else { |
|  | ncl = (res == FR\_DISK\_ERR) ? 0xFFFFFFFF : 1; |
|  | } |
|  |  |
|  | return ncl; /\* Return new cluster number or error code \*/ |
|  | } |
|  | #endif /\* !\_FS\_READONLY \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* FAT handling - Convert offset into cluster with link map table \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | #if \_USE\_FASTSEEK |
|  | static |
|  | DWORD clmt\_clust ( /\* <2:Error, >=2:Cluster number \*/ |
|  | FIL\* fp, /\* Pointer to the file object \*/ |
|  | DWORD ofs /\* File offset to be converted to cluster# \*/ |
|  | ) |
|  | { |
|  | DWORD cl, ncl, \*tbl; |
|  |  |
|  |  |
|  | tbl = fp->cltbl + 1; /\* Top of CLMT \*/ |
|  | cl = ofs / SS(fp->fs) / fp->fs->csize; /\* Cluster order from top of the file \*/ |
|  | for (;;) { |
|  | ncl = \*tbl++; /\* Number of cluters in the fragment \*/ |
|  | if (!ncl) return 0; /\* End of table? (error) \*/ |
|  | if (cl < ncl) break; /\* In this fragment? \*/ |
|  | cl -= ncl; tbl++; /\* Next fragment \*/ |
|  | } |
|  | return cl + \*tbl; /\* Return the cluster number \*/ |
|  | } |
|  | #endif /\* \_USE\_FASTSEEK \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Directory handling - Set directory index \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | FRESULT dir\_sdi ( |
|  | DIR\* dp, /\* Pointer to directory object \*/ |
|  | UINT idx /\* Index of directory table \*/ |
|  | ) |
|  | { |
|  | DWORD clst, sect; |
|  | UINT ic; |
|  |  |
|  |  |
|  | dp->index = (WORD)idx; /\* Current index \*/ |
|  | clst = dp->sclust; /\* Table start cluster (0:root) \*/ |
|  | if (clst == 1 || clst >= dp->fs->n\_fatent) /\* Check start cluster range \*/ |
|  | return FR\_INT\_ERR; |
|  | if (!clst && dp->fs->fs\_type == FS\_FAT32) /\* Replace cluster# 0 with root cluster# if in FAT32 \*/ |
|  | clst = dp->fs->dirbase; |
|  |  |
|  | if (clst == 0) { /\* Static table (root-directory in FAT12/16) \*/ |
|  | if (idx >= dp->fs->n\_rootdir) /\* Is index out of range? \*/ |
|  | return FR\_INT\_ERR; |
|  | sect = dp->fs->dirbase; |
|  | } |
|  | else { /\* Dynamic table (root-directory in FAT32 or sub-directory) \*/ |
|  | ic = SS(dp->fs) / SZ\_DIRE \* dp->fs->csize; /\* Entries per cluster \*/ |
|  | while (idx >= ic) { /\* Follow cluster chain \*/ |
|  | clst = get\_fat(dp->fs, clst); /\* Get next cluster \*/ |
|  | if (clst == 0xFFFFFFFF) return FR\_DISK\_ERR; /\* Disk error \*/ |
|  | if (clst < 2 || clst >= dp->fs->n\_fatent) /\* Reached to end of table or internal error \*/ |
|  | return FR\_INT\_ERR; |
|  | idx -= ic; |
|  | } |
|  | sect = clust2sect(dp->fs, clst); |
|  | } |
|  | dp->clust = clst; /\* Current cluster# \*/ |
|  | if (!sect) return FR\_INT\_ERR; |
|  | dp->sect = sect + idx / (SS(dp->fs) / SZ\_DIRE); /\* Sector# of the directory entry \*/ |
|  | dp->dir = dp->fs->win + (idx % (SS(dp->fs) / SZ\_DIRE)) \* SZ\_DIRE; /\* Ptr to the entry in the sector \*/ |
|  |  |
|  | return FR\_OK; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Directory handling - Move directory table index next \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | FRESULT dir\_next ( /\* FR\_OK:Succeeded, FR\_NO\_FILE:End of table, FR\_DENIED:Could not stretch \*/ |
|  | DIR\* dp, /\* Pointer to the directory object \*/ |
|  | int stretch /\* 0: Do not stretch table, 1: Stretch table if needed \*/ |
|  | ) |
|  | { |
|  | DWORD clst; |
|  | UINT i; |
|  | #if !\_FS\_READONLY |
|  | UINT c; |
|  | #endif |
|  |  |
|  |  |
|  | i = dp->index + 1; |
|  | if (!(i & 0xFFFF) || !dp->sect) /\* Report EOT when index has reached 65535 \*/ |
|  | return FR\_NO\_FILE; |
|  |  |
|  | if (!(i % (SS(dp->fs) / SZ\_DIRE))) { /\* Sector changed? \*/ |
|  | dp->sect++; /\* Next sector \*/ |
|  |  |
|  | if (!dp->clust) { /\* Static table \*/ |
|  | if (i >= dp->fs->n\_rootdir) /\* Report EOT if it reached end of static table \*/ |
|  | return FR\_NO\_FILE; |
|  | } |
|  | else { /\* Dynamic table \*/ |
|  | if (((i / (SS(dp->fs) / SZ\_DIRE)) & (dp->fs->csize - 1)) == 0) { /\* Cluster changed? \*/ |
|  | clst = get\_fat(dp->fs, dp->clust); /\* Get next cluster \*/ |
|  | if (clst <= 1) return FR\_INT\_ERR; |
|  | if (clst == 0xFFFFFFFF) return FR\_DISK\_ERR; |
|  | if (clst >= dp->fs->n\_fatent) { /\* If it reached end of dynamic table, \*/ |
|  | #if !\_FS\_READONLY |
|  | if (!stretch) return FR\_NO\_FILE; /\* If do not stretch, report EOT \*/ |
|  | clst = create\_chain(dp->fs, dp->clust); /\* Stretch cluster chain \*/ |
|  | if (clst == 0) return FR\_DENIED; /\* No free cluster \*/ |
|  | if (clst == 1) return FR\_INT\_ERR; |
|  | if (clst == 0xFFFFFFFF) return FR\_DISK\_ERR; |
|  | /\* Clean-up stretched table \*/ |
|  | if (sync\_window(dp->fs)) return FR\_DISK\_ERR;/\* Flush disk access window \*/ |
|  | mem\_set(dp->fs->win, 0, SS(dp->fs)); /\* Clear window buffer \*/ |
|  | dp->fs->winsect = clust2sect(dp->fs, clst); /\* Cluster start sector \*/ |
|  | for (c = 0; c < dp->fs->csize; c++) { /\* Fill the new cluster with 0 \*/ |
|  | dp->fs->wflag = 1; |
|  | if (sync\_window(dp->fs)) return FR\_DISK\_ERR; |
|  | dp->fs->winsect++; |
|  | } |
|  | dp->fs->winsect -= c; /\* Rewind window offset \*/ |
|  | #else |
|  | if (!stretch) return FR\_NO\_FILE; /\* If do not stretch, report EOT (this is to suppress warning) \*/ |
|  | return FR\_NO\_FILE; /\* Report EOT \*/ |
|  | #endif |
|  | } |
|  | dp->clust = clst; /\* Initialize data for new cluster \*/ |
|  | dp->sect = clust2sect(dp->fs, clst); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | dp->index = (WORD)i; /\* Current index \*/ |
|  | dp->dir = dp->fs->win + (i % (SS(dp->fs) / SZ\_DIRE)) \* SZ\_DIRE; /\* Current entry in the window \*/ |
|  |  |
|  | return FR\_OK; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Directory handling - Reserve directory entry \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | #if !\_FS\_READONLY |
|  | static |
|  | FRESULT dir\_alloc ( |
|  | DIR\* dp, /\* Pointer to the directory object \*/ |
|  | UINT nent /\* Number of contiguous entries to allocate (1-21) \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | UINT n; |
|  |  |
|  |  |
|  | res = dir\_sdi(dp, 0); |
|  | if (res == FR\_OK) { |
|  | n = 0; |
|  | do { |
|  | res = move\_window(dp->fs, dp->sect); |
|  | if (res != FR\_OK) break; |
|  | if (dp->dir[0] == DDEM || dp->dir[0] == 0) { /\* Is it a free entry? \*/ |
|  | if (++n == nent) break; /\* A block of contiguous free entries is found \*/ |
|  | } else { |
|  | n = 0; /\* Not a blank entry. Restart to search \*/ |
|  | } |
|  | res = dir\_next(dp, 1); /\* Next entry with table stretch enabled \*/ |
|  | } while (res == FR\_OK); |
|  | } |
|  | if (res == FR\_NO\_FILE) res = FR\_DENIED; /\* No directory entry to allocate \*/ |
|  | return res; |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Directory handling - Load/Store start cluster number \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | DWORD ld\_clust ( |
|  | FATFS\* fs, /\* Pointer to the fs object \*/ |
|  | BYTE\* dir /\* Pointer to the directory entry \*/ |
|  | ) |
|  | { |
|  | DWORD cl; |
|  |  |
|  | cl = LD\_WORD(dir + DIR\_FstClusLO); |
|  | if (fs->fs\_type == FS\_FAT32) |
|  | cl |= (DWORD)LD\_WORD(dir + DIR\_FstClusHI) << 16; |
|  |  |
|  | return cl; |
|  | } |
|  |  |
|  |  |
|  | #if !\_FS\_READONLY |
|  | static |
|  | void st\_clust ( |
|  | BYTE\* dir, /\* Pointer to the directory entry \*/ |
|  | DWORD cl /\* Value to be set \*/ |
|  | ) |
|  | { |
|  | ST\_WORD(dir + DIR\_FstClusLO, cl); |
|  | ST\_WORD(dir + DIR\_FstClusHI, cl >> 16); |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* LFN handling - Test/Pick/Fit an LFN segment from/to directory entry \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_USE\_LFN |
|  | static |
|  | const BYTE LfnOfs[] = {1,3,5,7,9,14,16,18,20,22,24,28,30}; /\* Offset of LFN characters in the directory entry \*/ |
|  |  |
|  |  |
|  | static |
|  | int cmp\_lfn ( /\* 1:Matched, 0:Not matched \*/ |
|  | WCHAR\* lfnbuf, /\* Pointer to the LFN to be compared \*/ |
|  | BYTE\* dir /\* Pointer to the directory entry containing a part of LFN \*/ |
|  | ) |
|  | { |
|  | UINT i, s; |
|  | WCHAR wc, uc; |
|  |  |
|  |  |
|  | i = ((dir[LDIR\_Ord] & ~LLEF) - 1) \* 13; /\* Get offset in the LFN buffer \*/ |
|  | s = 0; wc = 1; |
|  | do { |
|  | uc = LD\_WORD(dir + LfnOfs[s]); /\* Pick an LFN character from the entry \*/ |
|  | if (wc) { /\* Last character has not been processed \*/ |
|  | wc = ff\_wtoupper(uc); /\* Convert it to upper case \*/ |
|  | if (i >= \_MAX\_LFN || wc != ff\_wtoupper(lfnbuf[i++])) /\* Compare it \*/ |
|  | return 0; /\* Not matched \*/ |
|  | } else { |
|  | if (uc != 0xFFFF) return 0; /\* Check filler \*/ |
|  | } |
|  | } while (++s < 13); /\* Repeat until all characters in the entry are checked \*/ |
|  |  |
|  | if ((dir[LDIR\_Ord] & LLEF) && wc && lfnbuf[i]) /\* Last segment matched but different length \*/ |
|  | return 0; |
|  |  |
|  | return 1; /\* The part of LFN matched \*/ |
|  | } |
|  |  |
|  |  |
|  |  |
|  | static |
|  | int pick\_lfn ( /\* 1:Succeeded, 0:Buffer overflow \*/ |
|  | WCHAR\* lfnbuf, /\* Pointer to the Unicode-LFN buffer \*/ |
|  | BYTE\* dir /\* Pointer to the directory entry \*/ |
|  | ) |
|  | { |
|  | UINT i, s; |
|  | WCHAR wc, uc; |
|  |  |
|  |  |
|  | i = ((dir[LDIR\_Ord] & 0x3F) - 1) \* 13; /\* Offset in the LFN buffer \*/ |
|  |  |
|  | s = 0; wc = 1; |
|  | do { |
|  | uc = LD\_WORD(dir + LfnOfs[s]); /\* Pick an LFN character from the entry \*/ |
|  | if (wc) { /\* Last character has not been processed \*/ |
|  | if (i >= \_MAX\_LFN) return 0; /\* Buffer overflow? \*/ |
|  | lfnbuf[i++] = wc = uc; /\* Store it \*/ |
|  | } else { |
|  | if (uc != 0xFFFF) return 0; /\* Check filler \*/ |
|  | } |
|  | } while (++s < 13); /\* Read all character in the entry \*/ |
|  |  |
|  | if (dir[LDIR\_Ord] & LLEF) { /\* Put terminator if it is the last LFN part \*/ |
|  | if (i >= \_MAX\_LFN) return 0; /\* Buffer overflow? \*/ |
|  | lfnbuf[i] = 0; |
|  | } |
|  |  |
|  | return 1; |
|  | } |
|  |  |
|  |  |
|  | #if !\_FS\_READONLY |
|  | static |
|  | void fit\_lfn ( |
|  | const WCHAR\* lfnbuf, /\* Pointer to the LFN buffer \*/ |
|  | BYTE\* dir, /\* Pointer to the directory entry \*/ |
|  | BYTE ord, /\* LFN order (1-20) \*/ |
|  | BYTE sum /\* SFN sum \*/ |
|  | ) |
|  | { |
|  | UINT i, s; |
|  | WCHAR wc; |
|  |  |
|  |  |
|  | dir[LDIR\_Chksum] = sum; /\* Set check sum \*/ |
|  | dir[LDIR\_Attr] = AM\_LFN; /\* Set attribute. LFN entry \*/ |
|  | dir[LDIR\_Type] = 0; |
|  | ST\_WORD(dir + LDIR\_FstClusLO, 0); |
|  |  |
|  | i = (ord - 1) \* 13; /\* Get offset in the LFN buffer \*/ |
|  | s = wc = 0; |
|  | do { |
|  | if (wc != 0xFFFF) wc = lfnbuf[i++]; /\* Get an effective character \*/ |
|  | ST\_WORD(dir+LfnOfs[s], wc); /\* Put it \*/ |
|  | if (!wc) wc = 0xFFFF; /\* Padding characters following last character \*/ |
|  | } while (++s < 13); |
|  | if (wc == 0xFFFF || !lfnbuf[i]) ord |= LLEF; /\* Bottom LFN part is the start of LFN sequence \*/ |
|  | dir[LDIR\_Ord] = ord; /\* Set the LFN order \*/ |
|  | } |
|  |  |
|  | #endif |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Create numbered name \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_USE\_LFN |
|  | static |
|  | void gen\_numname ( |
|  | BYTE\* dst, /\* Pointer to the buffer to store numbered SFN \*/ |
|  | const BYTE\* src, /\* Pointer to SFN \*/ |
|  | const WCHAR\* lfn, /\* Pointer to LFN \*/ |
|  | UINT seq /\* Sequence number \*/ |
|  | ) |
|  | { |
|  | BYTE ns[8], c; |
|  | UINT i, j; |
|  | WCHAR wc; |
|  | DWORD sr; |
|  |  |
|  |  |
|  | mem\_cpy(dst, src, 11); |
|  |  |
|  | if (seq > 5) { /\* On many collisions, generate a hash number instead of sequential number \*/ |
|  | sr = seq; |
|  | while (\*lfn) { /\* Create a CRC \*/ |
|  | wc = \*lfn++; |
|  | for (i = 0; i < 16; i++) { |
|  | sr = (sr << 1) + (wc & 1); |
|  | wc >>= 1; |
|  | if (sr & 0x10000) sr ^= 0x11021; |
|  | } |
|  | } |
|  | seq = (UINT)sr; |
|  | } |
|  |  |
|  | /\* itoa (hexdecimal) \*/ |
|  | i = 7; |
|  | do { |
|  | c = (seq % 16) + '0'; |
|  | if (c > '9') c += 7; |
|  | ns[i--] = c; |
|  | seq /= 16; |
|  | } while (seq); |
|  | ns[i] = '~'; |
|  |  |
|  | /\* Append the number \*/ |
|  | for (j = 0; j < i && dst[j] != ' '; j++) { |
|  | if (IsDBCS1(dst[j])) { |
|  | if (j == i - 1) break; |
|  | j++; |
|  | } |
|  | } |
|  | do { |
|  | dst[j++] = (i < 8) ? ns[i++] : ' '; |
|  | } while (j < 8); |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Calculate sum of an SFN \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_USE\_LFN |
|  | static |
|  | BYTE sum\_sfn ( |
|  | const BYTE\* dir /\* Pointer to the SFN entry \*/ |
|  | ) |
|  | { |
|  | BYTE sum = 0; |
|  | UINT n = 11; |
|  |  |
|  | do sum = (sum >> 1) + (sum << 7) + \*dir++; while (--n); |
|  | return sum; |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Directory handling - Find an object in the directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | FRESULT dir\_find ( |
|  | DIR\* dp /\* Pointer to the directory object linked to the file name \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | BYTE c, \*dir; |
|  | #if \_USE\_LFN |
|  | BYTE a, ord, sum; |
|  | #endif |
|  |  |
|  | res = dir\_sdi(dp, 0); /\* Rewind directory object \*/ |
|  | if (res != FR\_OK) return res; |
|  |  |
|  | #if \_USE\_LFN |
|  | ord = sum = 0xFF; dp->lfn\_idx = 0xFFFF; /\* Reset LFN sequence \*/ |
|  | #endif |
|  | do { |
|  | res = move\_window(dp->fs, dp->sect); |
|  | if (res != FR\_OK) break; |
|  | dir = dp->dir; /\* Ptr to the directory entry of current index \*/ |
|  | c = dir[DIR\_Name]; |
|  | if (c == 0) { res = FR\_NO\_FILE; break; } /\* Reached to end of table \*/ |
|  | #if \_USE\_LFN /\* LFN configuration \*/ |
|  | a = dir[DIR\_Attr] & AM\_MASK; |
|  | if (c == DDEM || ((a & AM\_VOL) && a != AM\_LFN)) { /\* An entry without valid data \*/ |
|  | ord = 0xFF; dp->lfn\_idx = 0xFFFF; /\* Reset LFN sequence \*/ |
|  | } else { |
|  | if (a == AM\_LFN) { /\* An LFN entry is found \*/ |
|  | if (dp->lfn) { |
|  | if (c & LLEF) { /\* Is it start of LFN sequence? \*/ |
|  | sum = dir[LDIR\_Chksum]; |
|  | c &= ~LLEF; ord = c; /\* LFN start order \*/ |
|  | dp->lfn\_idx = dp->index; /\* Start index of LFN \*/ |
|  | } |
|  | /\* Check validity of the LFN entry and compare it with given name \*/ |
|  | ord = (c == ord && sum == dir[LDIR\_Chksum] && cmp\_lfn(dp->lfn, dir)) ? ord - 1 : 0xFF; |
|  | } |
|  | } else { /\* An SFN entry is found \*/ |
|  | if (!ord && sum == sum\_sfn(dir)) break; /\* LFN matched? \*/ |
|  | if (!(dp->fn[NSFLAG] & NS\_LOSS) && !mem\_cmp(dir, dp->fn, 11)) break; /\* SFN matched? \*/ |
|  | ord = 0xFF; dp->lfn\_idx = 0xFFFF; /\* Reset LFN sequence \*/ |
|  | } |
|  | } |
|  | #else /\* Non LFN configuration \*/ |
|  | if (!(dir[DIR\_Attr] & AM\_VOL) && !mem\_cmp(dir, dp->fn, 11)) /\* Is it a valid entry? \*/ |
|  | break; |
|  | #endif |
|  | res = dir\_next(dp, 0); /\* Next entry \*/ |
|  | } while (res == FR\_OK); |
|  |  |
|  | return res; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Read an object from the directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_FS\_MINIMIZE <= 1 || \_USE\_LABEL || \_FS\_RPATH >= 2 |
|  | static |
|  | FRESULT dir\_read ( |
|  | DIR\* dp, /\* Pointer to the directory object \*/ |
|  | int vol /\* Filtered by 0:file/directory or 1:volume label \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | BYTE a, c, \*dir; |
|  | #if \_USE\_LFN |
|  | BYTE ord = 0xFF, sum = 0xFF; |
|  | #endif |
|  |  |
|  | res = FR\_NO\_FILE; |
|  | while (dp->sect) { |
|  | res = move\_window(dp->fs, dp->sect); |
|  | if (res != FR\_OK) break; |
|  | dir = dp->dir; /\* Ptr to the directory entry of current index \*/ |
|  | c = dir[DIR\_Name]; |
|  | if (c == 0) { res = FR\_NO\_FILE; break; } /\* Reached to end of table \*/ |
|  | a = dir[DIR\_Attr] & AM\_MASK; |
|  | #if \_USE\_LFN /\* LFN configuration \*/ |
|  | if (c == DDEM || (!\_FS\_RPATH && c == '.') || (int)((a & ~AM\_ARC) == AM\_VOL) != vol) { /\* An entry without valid data \*/ |
|  | ord = 0xFF; |
|  | } else { |
|  | if (a == AM\_LFN) { /\* An LFN entry is found \*/ |
|  | if (c & LLEF) { /\* Is it start of LFN sequence? \*/ |
|  | sum = dir[LDIR\_Chksum]; |
|  | c &= ~LLEF; ord = c; |
|  | dp->lfn\_idx = dp->index; |
|  | } |
|  | /\* Check LFN validity and capture it \*/ |
|  | ord = (c == ord && sum == dir[LDIR\_Chksum] && pick\_lfn(dp->lfn, dir)) ? ord - 1 : 0xFF; |
|  | } else { /\* An SFN entry is found \*/ |
|  | if (ord || sum != sum\_sfn(dir)) /\* Is there a valid LFN? \*/ |
|  | dp->lfn\_idx = 0xFFFF; /\* It has no LFN. \*/ |
|  | break; |
|  | } |
|  | } |
|  | #else /\* Non LFN configuration \*/ |
|  | if (c != DDEM && (\_FS\_RPATH || c != '.') && a != AM\_LFN && (int)((a & ~AM\_ARC) == AM\_VOL) == vol) /\* Is it a valid entry? \*/ |
|  | break; |
|  | #endif |
|  | res = dir\_next(dp, 0); /\* Next entry \*/ |
|  | if (res != FR\_OK) break; |
|  | } |
|  |  |
|  | if (res != FR\_OK) dp->sect = 0; |
|  |  |
|  | return res; |
|  | } |
|  | #endif /\* \_FS\_MINIMIZE <= 1 || \_USE\_LABEL || \_FS\_RPATH >= 2 \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Register an object to the directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if !\_FS\_READONLY |
|  | static |
|  | FRESULT dir\_register ( /\* FR\_OK:Successful, FR\_DENIED:No free entry or too many SFN collision, FR\_DISK\_ERR:Disk error \*/ |
|  | DIR\* dp /\* Target directory with object name to be created \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | #if \_USE\_LFN /\* LFN configuration \*/ |
|  | UINT n, nent; |
|  | BYTE sn[12], \*fn, sum; |
|  | WCHAR \*lfn; |
|  |  |
|  |  |
|  | fn = dp->fn; lfn = dp->lfn; |
|  | mem\_cpy(sn, fn, 12); |
|  |  |
|  | if (\_FS\_RPATH && (sn[NSFLAG] & NS\_DOT)) /\* Cannot create dot entry \*/ |
|  | return FR\_INVALID\_NAME; |
|  |  |
|  | if (sn[NSFLAG] & NS\_LOSS) { /\* When LFN is out of 8.3 format, generate a numbered name \*/ |
|  | fn[NSFLAG] = 0; dp->lfn = 0; /\* Find only SFN \*/ |
|  | for (n = 1; n < 100; n++) { |
|  | gen\_numname(fn, sn, lfn, n); /\* Generate a numbered name \*/ |
|  | res = dir\_find(dp); /\* Check if the name collides with existing SFN \*/ |
|  | if (res != FR\_OK) break; |
|  | } |
|  | if (n == 100) return FR\_DENIED; /\* Abort if too many collisions \*/ |
|  | if (res != FR\_NO\_FILE) return res; /\* Abort if the result is other than 'not collided' \*/ |
|  | fn[NSFLAG] = sn[NSFLAG]; dp->lfn = lfn; |
|  | } |
|  |  |
|  | if (sn[NSFLAG] & NS\_LFN) { /\* When LFN is to be created, allocate entries for an SFN + LFNs. \*/ |
|  | for (n = 0; lfn[n]; n++) ; |
|  | nent = (n + 25) / 13; |
|  | } else { /\* Otherwise allocate an entry for an SFN \*/ |
|  | nent = 1; |
|  | } |
|  | res = dir\_alloc(dp, nent); /\* Allocate entries \*/ |
|  |  |
|  | if (res == FR\_OK && --nent) { /\* Set LFN entry if needed \*/ |
|  | res = dir\_sdi(dp, dp->index - nent); |
|  | if (res == FR\_OK) { |
|  | sum = sum\_sfn(dp->fn); /\* Sum value of the SFN tied to the LFN \*/ |
|  | do { /\* Store LFN entries in bottom first \*/ |
|  | res = move\_window(dp->fs, dp->sect); |
|  | if (res != FR\_OK) break; |
|  | fit\_lfn(dp->lfn, dp->dir, (BYTE)nent, sum); |
|  | dp->fs->wflag = 1; |
|  | res = dir\_next(dp, 0); /\* Next entry \*/ |
|  | } while (res == FR\_OK && --nent); |
|  | } |
|  | } |
|  | #else /\* Non LFN configuration \*/ |
|  | res = dir\_alloc(dp, 1); /\* Allocate an entry for SFN \*/ |
|  | #endif |
|  |  |
|  | if (res == FR\_OK) { /\* Set SFN entry \*/ |
|  | res = move\_window(dp->fs, dp->sect); |
|  | if (res == FR\_OK) { |
|  | mem\_set(dp->dir, 0, SZ\_DIRE); /\* Clean the entry \*/ |
|  | mem\_cpy(dp->dir, dp->fn, 11); /\* Put SFN \*/ |
|  | #if \_USE\_LFN |
|  | dp->dir[DIR\_NTres] = dp->fn[NSFLAG] & (NS\_BODY | NS\_EXT); /\* Put NT flag \*/ |
|  | #endif |
|  | dp->fs->wflag = 1; |
|  | } |
|  | } |
|  |  |
|  | return res; |
|  | } |
|  | #endif /\* !\_FS\_READONLY \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Remove an object from the directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if !\_FS\_READONLY && !\_FS\_MINIMIZE |
|  | static |
|  | FRESULT dir\_remove ( /\* FR\_OK: Successful, FR\_DISK\_ERR: A disk error \*/ |
|  | DIR\* dp /\* Directory object pointing the entry to be removed \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | #if \_USE\_LFN /\* LFN configuration \*/ |
|  | UINT i; |
|  |  |
|  | i = dp->index; /\* SFN index \*/ |
|  | res = dir\_sdi(dp, (dp->lfn\_idx == 0xFFFF) ? i : dp->lfn\_idx); /\* Goto the SFN or top of the LFN entries \*/ |
|  | if (res == FR\_OK) { |
|  | do { |
|  | res = move\_window(dp->fs, dp->sect); |
|  | if (res != FR\_OK) break; |
|  | mem\_set(dp->dir, 0, SZ\_DIRE); /\* Clear and mark the entry "deleted" \*/ |
|  | \*dp->dir = DDEM; |
|  | dp->fs->wflag = 1; |
|  | if (dp->index >= i) break; /\* When reached SFN, all entries of the object has been deleted. \*/ |
|  | res = dir\_next(dp, 0); /\* Next entry \*/ |
|  | } while (res == FR\_OK); |
|  | if (res == FR\_NO\_FILE) res = FR\_INT\_ERR; |
|  | } |
|  |  |
|  | #else /\* Non LFN configuration \*/ |
|  | res = dir\_sdi(dp, dp->index); |
|  | if (res == FR\_OK) { |
|  | res = move\_window(dp->fs, dp->sect); |
|  | if (res == FR\_OK) { |
|  | mem\_set(dp->dir, 0, SZ\_DIRE); /\* Clear and mark the entry "deleted" \*/ |
|  | \*dp->dir = DDEM; |
|  | dp->fs->wflag = 1; |
|  | } |
|  | } |
|  | #endif |
|  |  |
|  | return res; |
|  | } |
|  | #endif /\* !\_FS\_READONLY \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Get file information from directory entry \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_FS\_MINIMIZE <= 1 || \_FS\_RPATH >= 2 |
|  | static |
|  | void get\_fileinfo ( /\* No return code \*/ |
|  | DIR\* dp, /\* Pointer to the directory object \*/ |
|  | FILINFO\* fno /\* Pointer to the file information to be filled \*/ |
|  | ) |
|  | { |
|  | UINT i; |
|  | TCHAR \*p, c; |
|  | BYTE \*dir; |
|  | #if \_USE\_LFN |
|  | WCHAR w, \*lfn; |
|  | #endif |
|  |  |
|  | p = fno->fname; |
|  | if (dp->sect) { /\* Get SFN \*/ |
|  | dir = dp->dir; |
|  | i = 0; |
|  | while (i < 11) { /\* Copy name body and extension \*/ |
|  | c = (TCHAR)dir[i++]; |
|  | if (c == ' ') continue; /\* Skip padding spaces \*/ |
|  | if (c == RDDEM) c = (TCHAR)DDEM; /\* Restore replaced DDEM character \*/ |
|  | if (i == 9) \*p++ = '.'; /\* Insert a . if extension is exist \*/ |
|  | #if \_USE\_LFN |
|  | if (IsUpper(c) && (dir[DIR\_NTres] & (i >= 9 ? NS\_EXT : NS\_BODY))) |
|  | c += 0x20; /\* To lower \*/ |
|  | #if \_LFN\_UNICODE |
|  | if (IsDBCS1(c) && i != 8 && i != 11 && IsDBCS2(dir[i])) |
|  | c = c << 8 | dir[i++]; |
|  | c = ff\_convert(c, 1); /\* OEM -> Unicode \*/ |
|  | if (!c) c = '?'; |
|  | #endif |
|  | #endif |
|  | \*p++ = c; |
|  | } |
|  | fno->fattrib = dir[DIR\_Attr]; /\* Attribute \*/ |
|  | fno->fsize = LD\_DWORD(dir + DIR\_FileSize); /\* Size \*/ |
|  | fno->fdate = LD\_WORD(dir + DIR\_WrtDate); /\* Date \*/ |
|  | fno->ftime = LD\_WORD(dir + DIR\_WrtTime); /\* Time \*/ |
|  | } |
|  | \*p = 0; /\* Terminate SFN string by a \0 \*/ |
|  |  |
|  | #if \_USE\_LFN |
|  | if (fno->lfname) { |
|  | i = 0; p = fno->lfname; |
|  | if (dp->sect && fno->lfsize && dp->lfn\_idx != 0xFFFF) { /\* Get LFN if available \*/ |
|  | lfn = dp->lfn; |
|  | while ((w = \*lfn++) != 0) { /\* Get an LFN character \*/ |
|  | #if !\_LFN\_UNICODE |
|  | w = ff\_convert(w, 0); /\* Unicode -> OEM \*/ |
|  | if (!w) { i = 0; break; } /\* No LFN if it could not be converted \*/ |
|  | if (\_DF1S && w >= 0x100) /\* Put 1st byte if it is a DBC (always false on SBCS cfg) \*/ |
|  | p[i++] = (TCHAR)(w >> 8); |
|  | #endif |
|  | if (i >= fno->lfsize - 1) { i = 0; break; } /\* No LFN if buffer overflow \*/ |
|  | p[i++] = (TCHAR)w; |
|  | } |
|  | } |
|  | p[i] = 0; /\* Terminate LFN string by a \0 \*/ |
|  | } |
|  | #endif |
|  | } |
|  | #endif /\* \_FS\_MINIMIZE <= 1 || \_FS\_RPATH >= 2 \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Pattern matching \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_USE\_FIND && \_FS\_MINIMIZE <= 1 |
|  | static |
|  | WCHAR get\_achar ( /\* Get a character and advances ptr 1 or 2 \*/ |
|  | const TCHAR\*\* ptr /\* Pointer to pointer to the SBCS/DBCS/Unicode string \*/ |
|  | ) |
|  | { |
|  | WCHAR chr; |
|  |  |
|  | #if !\_LFN\_UNICODE |
|  | chr = (BYTE)\*(\*ptr)++; /\* Get a byte \*/ |
|  | if (IsLower(chr)) chr -= 0x20; /\* To upper ASCII char \*/ |
|  | if (IsDBCS1(chr) && IsDBCS2(\*\*ptr)) /\* Get DBC 2nd byte if needed \*/ |
|  | chr = chr << 8 | (BYTE)\*(\*ptr)++; |
|  | #ifdef \_EXCVT |
|  | if (chr >= 0x80) chr = ExCvt[chr - 0x80]; /\* To upper SBCS extended char \*/ |
|  | #endif |
|  | #else |
|  | chr = ff\_wtoupper(\*(\*ptr)++); /\* Get a word and to upper \*/ |
|  | #endif |
|  | return chr; |
|  | } |
|  |  |
|  |  |
|  | static |
|  | int pattern\_matching ( /\* Return value: 0:mismatched, 1:matched \*/ |
|  | const TCHAR\* pat, /\* Matching pattern \*/ |
|  | const TCHAR\* nam, /\* String to be tested \*/ |
|  | int skip, /\* Number of pre-skip chars (number of ?s) \*/ |
|  | int inf /\* Infinite search (\* specified) \*/ |
|  | ) |
|  | { |
|  | const TCHAR \*pp, \*np; |
|  | WCHAR pc, nc; |
|  | int nm, nx; |
|  |  |
|  |  |
|  | while (skip--) { /\* Pre-skip name chars \*/ |
|  | if (!get\_achar(&nam)) return 0; /\* Branch mismatched if less name chars \*/ |
|  | } |
|  | if (!\*pat && inf) return 1; /\* (short circuit) \*/ |
|  |  |
|  | do { |
|  | pp = pat; np = nam; /\* Top of pattern and name to match \*/ |
|  | for (;;) { |
|  | if (\*pp == '?' || \*pp == '\*') { /\* Wildcard? \*/ |
|  | nm = nx = 0; |
|  | do { /\* Analyze the wildcard chars \*/ |
|  | if (\*pp++ == '?') nm++; else nx = 1; |
|  | } while (\*pp == '?' || \*pp == '\*'); |
|  | if (pattern\_matching(pp, np, nm, nx)) return 1; /\* Test new branch (recurs upto number of wildcard blocks in the pattern) \*/ |
|  | nc = \*np; break; /\* Branch mismatched \*/ |
|  | } |
|  | pc = get\_achar(&pp); /\* Get a pattern char \*/ |
|  | nc = get\_achar(&np); /\* Get a name char \*/ |
|  | if (pc != nc) break; /\* Branch mismatched? \*/ |
|  | if (!pc) return 1; /\* Branch matched? (matched at end of both strings) \*/ |
|  | } |
|  | get\_achar(&nam); /\* nam++ \*/ |
|  | } while (inf && nc); /\* Retry until end of name if infinite search is specified \*/ |
|  |  |
|  | return 0; |
|  | } |
|  | #endif /\* \_USE\_FIND && \_FS\_MINIMIZE <= 1 \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Pick a segment and create the object name in directory form \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | FRESULT create\_name ( |
|  | DIR\* dp, /\* Pointer to the directory object \*/ |
|  | const TCHAR\*\* path /\* Pointer to pointer to the segment in the path string \*/ |
|  | ) |
|  | { |
|  | #if \_USE\_LFN /\* LFN configuration \*/ |
|  | BYTE b, cf; |
|  | WCHAR w, \*lfn; |
|  | UINT i, ni, si, di; |
|  | const TCHAR \*p; |
|  |  |
|  | /\* Create LFN in Unicode \*/ |
|  | for (p = \*path; \*p == '/' || \*p == '\\'; p++) ; /\* Strip duplicated separator \*/ |
|  | lfn = dp->lfn; |
|  | si = di = 0; |
|  | for (;;) { |
|  | w = p[si++]; /\* Get a character \*/ |
|  | if (w < ' ' || w == '/' || w == '\\') break; /\* Break on end of segment \*/ |
|  | if (di >= \_MAX\_LFN) /\* Reject too long name \*/ |
|  | return FR\_INVALID\_NAME; |
|  | #if !\_LFN\_UNICODE |
|  | w &= 0xFF; |
|  | if (IsDBCS1(w)) { /\* Check if it is a DBC 1st byte (always false on SBCS cfg) \*/ |
|  | b = (BYTE)p[si++]; /\* Get 2nd byte \*/ |
|  | w = (w << 8) + b; /\* Create a DBC \*/ |
|  | if (!IsDBCS2(b)) |
|  | return FR\_INVALID\_NAME; /\* Reject invalid sequence \*/ |
|  | } |
|  | w = ff\_convert(w, 1); /\* Convert ANSI/OEM to Unicode \*/ |
|  | if (!w) return FR\_INVALID\_NAME; /\* Reject invalid code \*/ |
|  | #endif |
|  | if (w < 0x80 && chk\_chr("\"\*:<>\?|\x7F", w)) /\* Reject illegal characters for LFN \*/ |
|  | return FR\_INVALID\_NAME; |
|  | lfn[di++] = w; /\* Store the Unicode character \*/ |
|  | } |
|  | \*path = &p[si]; /\* Return pointer to the next segment \*/ |
|  | cf = (w < ' ') ? NS\_LAST : 0; /\* Set last segment flag if end of path \*/ |
|  | #if \_FS\_RPATH |
|  | if ((di == 1 && lfn[di - 1] == '.') || /\* Is this a dot entry? \*/ |
|  | (di == 2 && lfn[di - 1] == '.' && lfn[di - 2] == '.')) { |
|  | lfn[di] = 0; |
|  | for (i = 0; i < 11; i++) |
|  | dp->fn[i] = (i < di) ? '.' : ' '; |
|  | dp->fn[i] = cf | NS\_DOT; /\* This is a dot entry \*/ |
|  | return FR\_OK; |
|  | } |
|  | #endif |
|  | while (di) { /\* Strip trailing spaces and dots \*/ |
|  | w = lfn[di - 1]; |
|  | if (w != ' ' && w != '.') break; |
|  | di--; |
|  | } |
|  | if (!di) return FR\_INVALID\_NAME; /\* Reject nul string \*/ |
|  |  |
|  | lfn[di] = 0; /\* LFN is created \*/ |
|  |  |
|  | /\* Create SFN in directory form \*/ |
|  | mem\_set(dp->fn, ' ', 11); |
|  | for (si = 0; lfn[si] == ' ' || lfn[si] == '.'; si++) ; /\* Strip leading spaces and dots \*/ |
|  | if (si) cf |= NS\_LOSS | NS\_LFN; |
|  | while (di && lfn[di - 1] != '.') di--; /\* Find extension (di<=si: no extension) \*/ |
|  |  |
|  | b = i = 0; ni = 8; |
|  | for (;;) { |
|  | w = lfn[si++]; /\* Get an LFN character \*/ |
|  | if (!w) break; /\* Break on end of the LFN \*/ |
|  | if (w == ' ' || (w == '.' && si != di)) { /\* Remove spaces and dots \*/ |
|  | cf |= NS\_LOSS | NS\_LFN; continue; |
|  | } |
|  |  |
|  | if (i >= ni || si == di) { /\* Extension or end of SFN \*/ |
|  | if (ni == 11) { /\* Long extension \*/ |
|  | cf |= NS\_LOSS | NS\_LFN; break; |
|  | } |
|  | if (si != di) cf |= NS\_LOSS | NS\_LFN; /\* Out of 8.3 format \*/ |
|  | if (si > di) break; /\* No extension \*/ |
|  | si = di; i = 8; ni = 11; /\* Enter extension section \*/ |
|  | b <<= 2; continue; |
|  | } |
|  |  |
|  | if (w >= 0x80) { /\* Non ASCII character \*/ |
|  | #ifdef \_EXCVT |
|  | w = ff\_convert(w, 0); /\* Unicode -> OEM code \*/ |
|  | if (w) w = ExCvt[w - 0x80]; /\* Convert extended character to upper (SBCS) \*/ |
|  | #else |
|  | w = ff\_convert(ff\_wtoupper(w), 0); /\* Upper converted Unicode -> OEM code \*/ |
|  | #endif |
|  | cf |= NS\_LFN; /\* Force create LFN entry \*/ |
|  | } |
|  |  |
|  | if (\_DF1S && w >= 0x100) { /\* DBC (always false at SBCS cfg) \*/ |
|  | if (i >= ni - 1) { |
|  | cf |= NS\_LOSS | NS\_LFN; i = ni; continue; |
|  | } |
|  | dp->fn[i++] = (BYTE)(w >> 8); |
|  | } else { /\* SBC \*/ |
|  | if (!w || chk\_chr("+,;=[]", w)) { /\* Replace illegal characters for SFN \*/ |
|  | w = '\_'; cf |= NS\_LOSS | NS\_LFN;/\* Lossy conversion \*/ |
|  | } else { |
|  | if (IsUpper(w)) { /\* ASCII large capital \*/ |
|  | b |= 2; |
|  | } else { |
|  | if (IsLower(w)) { /\* ASCII small capital \*/ |
|  | b |= 1; w -= 0x20; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | dp->fn[i++] = (BYTE)w; |
|  | } |
|  |  |
|  | if (dp->fn[0] == DDEM) dp->fn[0] = RDDEM; /\* If the first character collides with deleted mark, replace it with RDDEM \*/ |
|  |  |
|  | if (ni == 8) b <<= 2; |
|  | if ((b & 0x0C) == 0x0C || (b & 0x03) == 0x03) /\* Create LFN entry when there are composite capitals \*/ |
|  | cf |= NS\_LFN; |
|  | if (!(cf & NS\_LFN)) { /\* When LFN is in 8.3 format without extended character, NT flags are created \*/ |
|  | if ((b & 0x03) == 0x01) cf |= NS\_EXT; /\* NT flag (Extension has only small capital) \*/ |
|  | if ((b & 0x0C) == 0x04) cf |= NS\_BODY; /\* NT flag (Filename has only small capital) \*/ |
|  | } |
|  |  |
|  | dp->fn[NSFLAG] = cf; /\* SFN is created \*/ |
|  |  |
|  | return FR\_OK; |
|  |  |
|  |  |
|  | #else /\* Non-LFN configuration \*/ |
|  | BYTE b, c, d, \*sfn; |
|  | UINT ni, si, i; |
|  | const char \*p; |
|  |  |
|  | /\* Create file name in directory form \*/ |
|  | for (p = \*path; \*p == '/' || \*p == '\\'; p++) ; /\* Strip duplicated separator \*/ |
|  | sfn = dp->fn; |
|  | mem\_set(sfn, ' ', 11); |
|  | si = i = b = 0; ni = 8; |
|  | #if \_FS\_RPATH |
|  | if (p[si] == '.') { /\* Is this a dot entry? \*/ |
|  | for (;;) { |
|  | c = (BYTE)p[si++]; |
|  | if (c != '.' || si >= 3) break; |
|  | sfn[i++] = c; |
|  | } |
|  | if (c != '/' && c != '\\' && c > ' ') return FR\_INVALID\_NAME; |
|  | \*path = &p[si]; /\* Return pointer to the next segment \*/ |
|  | sfn[NSFLAG] = (c <= ' ') ? NS\_LAST | NS\_DOT : NS\_DOT; /\* Set last segment flag if end of path \*/ |
|  | return FR\_OK; |
|  | } |
|  | #endif |
|  | for (;;) { |
|  | c = (BYTE)p[si++]; |
|  | if (c <= ' ' || c == '/' || c == '\\') break; /\* Break on end of segment \*/ |
|  | if (c == '.' || i >= ni) { |
|  | if (ni != 8 || c != '.') return FR\_INVALID\_NAME; |
|  | i = 8; ni = 11; |
|  | b <<= 2; continue; |
|  | } |
|  | if (c >= 0x80) { /\* Extended character? \*/ |
|  | b |= 3; /\* Eliminate NT flag \*/ |
|  | #ifdef \_EXCVT |
|  | c = ExCvt[c - 0x80]; /\* To upper extended characters (SBCS cfg) \*/ |
|  | #else |
|  | #if !\_DF1S |
|  | return FR\_INVALID\_NAME; /\* Reject extended characters (ASCII cfg) \*/ |
|  | #endif |
|  | #endif |
|  | } |
|  | if (IsDBCS1(c)) { /\* Check if it is a DBC 1st byte (always false on SBCS cfg) \*/ |
|  | d = (BYTE)p[si++]; /\* Get 2nd byte \*/ |
|  | if (!IsDBCS2(d) || i >= ni - 1) /\* Reject invalid DBC \*/ |
|  | return FR\_INVALID\_NAME; |
|  | sfn[i++] = c; |
|  | sfn[i++] = d; |
|  | } else { /\* SBC \*/ |
|  | if (chk\_chr("\"\*+,:;<=>\?[]|\x7F", c)) /\* Reject illegal chrs for SFN \*/ |
|  | return FR\_INVALID\_NAME; |
|  | if (IsUpper(c)) { /\* ASCII large capital? \*/ |
|  | b |= 2; |
|  | } else { |
|  | if (IsLower(c)) { /\* ASCII small capital? \*/ |
|  | b |= 1; c -= 0x20; |
|  | } |
|  | } |
|  | sfn[i++] = c; |
|  | } |
|  | } |
|  | \*path = &p[si]; /\* Return pointer to the next segment \*/ |
|  | c = (c <= ' ') ? NS\_LAST : 0; /\* Set last segment flag if end of path \*/ |
|  |  |
|  | if (!i) return FR\_INVALID\_NAME; /\* Reject nul string \*/ |
|  | if (sfn[0] == DDEM) sfn[0] = RDDEM; /\* When first character collides with DDEM, replace it with RDDEM \*/ |
|  |  |
|  | if (ni == 8) b <<= 2; |
|  | if ((b & 0x03) == 0x01) c |= NS\_EXT; /\* NT flag (Name extension has only small capital) \*/ |
|  | if ((b & 0x0C) == 0x04) c |= NS\_BODY; /\* NT flag (Name body has only small capital) \*/ |
|  |  |
|  | sfn[NSFLAG] = c; /\* Store NT flag, File name is created \*/ |
|  |  |
|  | return FR\_OK; |
|  | #endif |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Follow a file path \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | FRESULT follow\_path ( /\* FR\_OK(0): successful, !=0: error code \*/ |
|  | DIR\* dp, /\* Directory object to return last directory and found object \*/ |
|  | const TCHAR\* path /\* Full-path string to find a file or directory \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | BYTE \*dir, ns; |
|  |  |
|  |  |
|  | #if \_FS\_RPATH |
|  | if (\*path == '/' || \*path == '\\') { /\* There is a heading separator \*/ |
|  | path++; dp->sclust = 0; /\* Strip it and start from the root directory \*/ |
|  | } else { /\* No heading separator \*/ |
|  | dp->sclust = dp->fs->cdir; /\* Start from the current directory \*/ |
|  | } |
|  | #else |
|  | if (\*path == '/' || \*path == '\\') /\* Strip heading separator if exist \*/ |
|  | path++; |
|  | dp->sclust = 0; /\* Always start from the root directory \*/ |
|  | #endif |
|  |  |
|  | if ((UINT)\*path < ' ') { /\* Null path name is the origin directory itself \*/ |
|  | res = dir\_sdi(dp, 0); |
|  | dp->dir = 0; |
|  | } else { /\* Follow path \*/ |
|  | for (;;) { |
|  | res = create\_name(dp, &path); /\* Get a segment name of the path \*/ |
|  | if (res != FR\_OK) break; |
|  | res = dir\_find(dp); /\* Find an object with the sagment name \*/ |
|  | ns = dp->fn[NSFLAG]; |
|  | if (res != FR\_OK) { /\* Failed to find the object \*/ |
|  | if (res == FR\_NO\_FILE) { /\* Object is not found \*/ |
|  | if (\_FS\_RPATH && (ns & NS\_DOT)) { /\* If dot entry is not exist, \*/ |
|  | dp->sclust = 0; dp->dir = 0; /\* it is the root directory and stay there \*/ |
|  | if (!(ns & NS\_LAST)) continue; /\* Continue to follow if not last segment \*/ |
|  | res = FR\_OK; /\* Ended at the root directroy. Function completed. \*/ |
|  | } else { /\* Could not find the object \*/ |
|  | if (!(ns & NS\_LAST)) res = FR\_NO\_PATH; /\* Adjust error code if not last segment \*/ |
|  | } |
|  | } |
|  | break; |
|  | } |
|  | if (ns & NS\_LAST) break; /\* Last segment matched. Function completed. \*/ |
|  | dir = dp->dir; /\* Follow the sub-directory \*/ |
|  | if (!(dir[DIR\_Attr] & AM\_DIR)) { /\* It is not a sub-directory and cannot follow \*/ |
|  | res = FR\_NO\_PATH; break; |
|  | } |
|  | dp->sclust = ld\_clust(dp->fs, dir); |
|  | } |
|  | } |
|  |  |
|  | return res; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Get logical drive number from path name \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | int get\_ldnumber ( /\* Returns logical drive number (-1:invalid drive) \*/ |
|  | const TCHAR\*\* path /\* Pointer to pointer to the path name \*/ |
|  | ) |
|  | { |
|  | const TCHAR \*tp, \*tt; |
|  | UINT i; |
|  | int vol = -1; |
|  | #if \_STR\_VOLUME\_ID /\* Find string drive id \*/ |
|  | static const char\* const str[] = {\_VOLUME\_STRS}; |
|  | const char \*sp; |
|  | char c; |
|  | TCHAR tc; |
|  | #endif |
|  |  |
|  |  |
|  | if (\*path) { /\* If the pointer is not a null \*/ |
|  | for (tt = \*path; (UINT)\*tt >= (\_USE\_LFN ? ' ' : '!') && \*tt != ':'; tt++) ; /\* Find ':' in the path \*/ |
|  | if (\*tt == ':') { /\* If a ':' is exist in the path name \*/ |
|  | tp = \*path; |
|  | i = \*tp++ - '0'; |
|  | if (i < 10 && tp == tt) { /\* Is there a numeric drive id? \*/ |
|  | if (i < \_VOLUMES) { /\* If a drive id is found, get the value and strip it \*/ |
|  | vol = (int)i; |
|  | \*path = ++tt; |
|  | } |
|  | } |
|  | #if \_STR\_VOLUME\_ID |
|  | else { /\* No numeric drive number, find string drive id \*/ |
|  | i = 0; tt++; |
|  | do { |
|  | sp = str[i]; tp = \*path; |
|  | do { /\* Compare a string drive id with path name \*/ |
|  | c = \*sp++; tc = \*tp++; |
|  | if (IsLower(tc)) tc -= 0x20; |
|  | } while (c && (TCHAR)c == tc); |
|  | } while ((c || tp != tt) && ++i < \_VOLUMES); /\* Repeat for each id until pattern match \*/ |
|  | if (i < \_VOLUMES) { /\* If a drive id is found, get the value and strip it \*/ |
|  | vol = (int)i; |
|  | \*path = tt; |
|  | } |
|  | } |
|  | #endif |
|  | return vol; |
|  | } |
|  | #if \_FS\_RPATH && \_VOLUMES >= 2 |
|  | vol = CurrVol; /\* Current drive \*/ |
|  | #else |
|  | vol = 0; /\* Drive 0 \*/ |
|  | #endif |
|  | } |
|  | return vol; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Load a sector and check if it is an FAT boot sector \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | BYTE check\_fs ( /\* 0:FAT boor sector, 1:Valid boor sector but not FAT, 2:Not a boot sector, 3:Disk error \*/ |
|  | FATFS\* fs, /\* File system object \*/ |
|  | DWORD sect /\* Sector# (lba) to check if it is an FAT boot record or not \*/ |
|  | ) |
|  | { |
|  | fs->wflag = 0; fs->winsect = 0xFFFFFFFF; /\* Invaidate window \*/ |
|  | if (move\_window(fs, sect) != FR\_OK) /\* Load boot record \*/ |
|  | return 3; |
|  |  |
|  | if (LD\_WORD(&fs->win[BS\_55AA]) != 0xAA55) /\* Check boot record signature (always placed at offset 510 even if the sector size is >512) \*/ |
|  | return 2; |
|  |  |
|  | if ((LD\_DWORD(&fs->win[BS\_FilSysType]) & 0xFFFFFF) == 0x544146) /\* Check "FAT" string \*/ |
|  | return 0; |
|  | if ((LD\_DWORD(&fs->win[BS\_FilSysType32]) & 0xFFFFFF) == 0x544146) /\* Check "FAT" string \*/ |
|  | return 0; |
|  |  |
|  | return 1; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Find logical drive and check if the volume is mounted \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | FRESULT find\_volume ( /\* FR\_OK(0): successful, !=0: any error occurred \*/ |
|  | FATFS\*\* rfs, /\* Pointer to pointer to the found file system object \*/ |
|  | const TCHAR\*\* path, /\* Pointer to pointer to the path name (drive number) \*/ |
|  | BYTE wmode /\* !=0: Check write protection for write access \*/ |
|  | ) |
|  | { |
|  | BYTE fmt, \*pt; |
|  | int vol; |
|  | DSTATUS stat; |
|  | DWORD bsect, fasize, tsect, sysect, nclst, szbfat, br[4]; |
|  | WORD nrsv; |
|  | FATFS \*fs; |
|  | UINT i; |
|  |  |
|  |  |
|  | /\* Get logical drive number from the path name \*/ |
|  | \*rfs = 0; |
|  | vol = get\_ldnumber(path); |
|  | if (vol < 0) return FR\_INVALID\_DRIVE; |
|  |  |
|  | /\* Check if the file system object is valid or not \*/ |
|  | fs = FatFs[vol]; /\* Get pointer to the file system object \*/ |
|  | if (!fs) return FR\_NOT\_ENABLED; /\* Is the file system object available? \*/ |
|  |  |
|  | ENTER\_FF(fs); /\* Lock the volume \*/ |
|  | \*rfs = fs; /\* Return pointer to the file system object \*/ |
|  |  |
|  | if (fs->fs\_type) { /\* If the volume has been mounted \*/ |
|  | stat = disk\_status(fs->drv); |
|  | if (!(stat & STA\_NOINIT)) { /\* and the physical drive is kept initialized \*/ |
|  | if (!\_FS\_READONLY && wmode && (stat & STA\_PROTECT)) /\* Check write protection if needed \*/ |
|  | return FR\_WRITE\_PROTECTED; |
|  | return FR\_OK; /\* The file system object is valid \*/ |
|  | } |
|  | } |
|  |  |
|  | /\* The file system object is not valid. \*/ |
|  | /\* Following code attempts to mount the volume. (analyze BPB and initialize the fs object) \*/ |
|  |  |
|  | fs->fs\_type = 0; /\* Clear the file system object \*/ |
|  | fs->drv = LD2PD(vol); /\* Bind the logical drive and a physical drive \*/ |
|  | stat = disk\_initialize(fs->drv); /\* Initialize the physical drive \*/ |
|  | if (stat & STA\_NOINIT) /\* Check if the initialization succeeded \*/ |
|  | return FR\_NOT\_READY; /\* Failed to initialize due to no medium or hard error \*/ |
|  | if (!\_FS\_READONLY && wmode && (stat & STA\_PROTECT)) /\* Check disk write protection if needed \*/ |
|  | return FR\_WRITE\_PROTECTED; |
|  | #if \_MAX\_SS != \_MIN\_SS /\* Get sector size (multiple sector size cfg only) \*/ |
|  | if (disk\_ioctl(fs->drv, GET\_SECTOR\_SIZE, &SS(fs)) != RES\_OK |
|  | || SS(fs) < \_MIN\_SS || SS(fs) > \_MAX\_SS) return FR\_DISK\_ERR; |
|  | #endif |
|  | /\* Find an FAT partition on the drive. Supports only generic partitioning, FDISK and SFD. \*/ |
|  | bsect = 0; |
|  | fmt = check\_fs(fs, bsect); /\* Load sector 0 and check if it is an FAT boot sector as SFD \*/ |
|  | if (fmt == 1 || (!fmt && (LD2PT(vol)))) { /\* Not an FAT boot sector or forced partition number \*/ |
|  | for (i = 0; i < 4; i++) { /\* Get partition offset \*/ |
|  | pt = fs->win + MBR\_Table + i \* SZ\_PTE; |
|  | br[i] = pt[4] ? LD\_DWORD(&pt[8]) : 0; |
|  | } |
|  | i = LD2PT(vol); /\* Partition number: 0:auto, 1-4:forced \*/ |
|  | if (i) i--; |
|  | do { /\* Find an FAT volume \*/ |
|  | bsect = br[i]; |
|  | fmt = bsect ? check\_fs(fs, bsect) : 2; /\* Check the partition \*/ |
|  | } while (!LD2PT(vol) && fmt && ++i < 4); |
|  | } |
|  | if (fmt == 3) return FR\_DISK\_ERR; /\* An error occured in the disk I/O layer \*/ |
|  | if (fmt) return FR\_NO\_FILESYSTEM; /\* No FAT volume is found \*/ |
|  |  |
|  | /\* An FAT volume is found. Following code initializes the file system object \*/ |
|  |  |
|  | if (LD\_WORD(fs->win + BPB\_BytsPerSec) != SS(fs)) /\* (BPB\_BytsPerSec must be equal to the physical sector size) \*/ |
|  | return FR\_NO\_FILESYSTEM; |
|  |  |
|  | fasize = LD\_WORD(fs->win + BPB\_FATSz16); /\* Number of sectors per FAT \*/ |
|  | if (!fasize) fasize = LD\_DWORD(fs->win + BPB\_FATSz32); |
|  | fs->fsize = fasize; |
|  |  |
|  | fs->n\_fats = fs->win[BPB\_NumFATs]; /\* Number of FAT copies \*/ |
|  | if (fs->n\_fats != 1 && fs->n\_fats != 2) /\* (Must be 1 or 2) \*/ |
|  | return FR\_NO\_FILESYSTEM; |
|  | fasize \*= fs->n\_fats; /\* Number of sectors for FAT area \*/ |
|  |  |
|  | fs->csize = fs->win[BPB\_SecPerClus]; /\* Number of sectors per cluster \*/ |
|  | if (!fs->csize || (fs->csize & (fs->csize - 1))) /\* (Must be power of 2) \*/ |
|  | return FR\_NO\_FILESYSTEM; |
|  |  |
|  | fs->n\_rootdir = LD\_WORD(fs->win + BPB\_RootEntCnt); /\* Number of root directory entries \*/ |
|  | if (fs->n\_rootdir % (SS(fs) / SZ\_DIRE)) /\* (Must be sector aligned) \*/ |
|  | return FR\_NO\_FILESYSTEM; |
|  |  |
|  | tsect = LD\_WORD(fs->win + BPB\_TotSec16); /\* Number of sectors on the volume \*/ |
|  | if (!tsect) tsect = LD\_DWORD(fs->win + BPB\_TotSec32); |
|  |  |
|  | nrsv = LD\_WORD(fs->win + BPB\_RsvdSecCnt); /\* Number of reserved sectors \*/ |
|  | if (!nrsv) return FR\_NO\_FILESYSTEM; /\* (Must not be 0) \*/ |
|  |  |
|  | /\* Determine the FAT sub type \*/ |
|  | sysect = nrsv + fasize + fs->n\_rootdir / (SS(fs) / SZ\_DIRE); /\* RSV + FAT + DIR \*/ |
|  | if (tsect < sysect) return FR\_NO\_FILESYSTEM; /\* (Invalid volume size) \*/ |
|  | nclst = (tsect - sysect) / fs->csize; /\* Number of clusters \*/ |
|  | if (!nclst) return FR\_NO\_FILESYSTEM; /\* (Invalid volume size) \*/ |
|  | fmt = FS\_FAT12; |
|  | if (nclst >= MIN\_FAT16) fmt = FS\_FAT16; |
|  | if (nclst >= MIN\_FAT32) fmt = FS\_FAT32; |
|  |  |
|  | /\* Boundaries and Limits \*/ |
|  | fs->n\_fatent = nclst + 2; /\* Number of FAT entries \*/ |
|  | fs->volbase = bsect; /\* Volume start sector \*/ |
|  | fs->fatbase = bsect + nrsv; /\* FAT start sector \*/ |
|  | fs->database = bsect + sysect; /\* Data start sector \*/ |
|  | if (fmt == FS\_FAT32) { |
|  | if (fs->n\_rootdir) return FR\_NO\_FILESYSTEM; /\* (BPB\_RootEntCnt must be 0) \*/ |
|  | fs->dirbase = LD\_DWORD(fs->win + BPB\_RootClus); /\* Root directory start cluster \*/ |
|  | szbfat = fs->n\_fatent \* 4; /\* (Needed FAT size) \*/ |
|  | } else { |
|  | if (!fs->n\_rootdir) return FR\_NO\_FILESYSTEM; /\* (BPB\_RootEntCnt must not be 0) \*/ |
|  | fs->dirbase = fs->fatbase + fasize; /\* Root directory start sector \*/ |
|  | szbfat = (fmt == FS\_FAT16) ? /\* (Needed FAT size) \*/ |
|  | fs->n\_fatent \* 2 : fs->n\_fatent \* 3 / 2 + (fs->n\_fatent & 1); |
|  | } |
|  | if (fs->fsize < (szbfat + (SS(fs) - 1)) / SS(fs)) /\* (BPB\_FATSz must not be less than the size needed) \*/ |
|  | return FR\_NO\_FILESYSTEM; |
|  |  |
|  | #if !\_FS\_READONLY |
|  | /\* Initialize cluster allocation information \*/ |
|  | fs->last\_clust = fs->free\_clust = 0xFFFFFFFF; |
|  |  |
|  | /\* Get fsinfo if available \*/ |
|  | fs->fsi\_flag = 0x80; |
|  | #if (\_FS\_NOFSINFO & 3) != 3 |
|  | if (fmt == FS\_FAT32 /\* Enable FSINFO only if FAT32 and BPB\_FSInfo is 1 \*/ |
|  | && LD\_WORD(fs->win + BPB\_FSInfo) == 1 |
|  | && move\_window(fs, bsect + 1) == FR\_OK) |
|  | { |
|  | fs->fsi\_flag = 0; |
|  | if (LD\_WORD(fs->win + BS\_55AA) == 0xAA55 /\* Load FSINFO data if available \*/ |
|  | && LD\_DWORD(fs->win + FSI\_LeadSig) == 0x41615252 |
|  | && LD\_DWORD(fs->win + FSI\_StrucSig) == 0x61417272) |
|  | { |
|  | #if (\_FS\_NOFSINFO & 1) == 0 |
|  | fs->free\_clust = LD\_DWORD(fs->win + FSI\_Free\_Count); |
|  | #endif |
|  | #if (\_FS\_NOFSINFO & 2) == 0 |
|  | fs->last\_clust = LD\_DWORD(fs->win + FSI\_Nxt\_Free); |
|  | #endif |
|  | } |
|  | } |
|  | #endif |
|  | #endif |
|  | fs->fs\_type = fmt; /\* FAT sub-type \*/ |
|  | fs->id = ++Fsid; /\* File system mount ID \*/ |
|  | #if \_FS\_RPATH |
|  | fs->cdir = 0; /\* Set current directory to root \*/ |
|  | #endif |
|  | #if \_FS\_LOCK /\* Clear file lock semaphores \*/ |
|  | clear\_lock(fs); |
|  | #endif |
|  |  |
|  | return FR\_OK; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Check if the file/directory object is valid or not \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | static |
|  | FRESULT validate ( /\* FR\_OK(0): The object is valid, !=0: Invalid \*/ |
|  | void\* obj /\* Pointer to the object FIL/DIR to check validity \*/ |
|  | ) |
|  | { |
|  | FIL \*fil = (FIL\*)obj; /\* Assuming offset of .fs and .id in the FIL/DIR structure is identical \*/ |
|  |  |
|  |  |
|  | if (!fil || !fil->fs || !fil->fs->fs\_type || fil->fs->id != fil->id) |
|  | return FR\_INVALID\_OBJECT; |
|  |  |
|  | ENTER\_FF(fil->fs); /\* Lock file system \*/ |
|  |  |
|  | if (disk\_status(fil->fs->drv) & STA\_NOINIT) |
|  | return FR\_NOT\_READY; |
|  |  |
|  | return FR\_OK; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-------------------------------------------------------------------------- |
|  |  |
|  | Public Functions |
|  |  |
|  | --------------------------------------------------------------------------\*/ |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Mount/Unmount a Logical Drive \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_mount ( |
|  | FATFS\* fs, /\* Pointer to the file system object (NULL:unmount)\*/ |
|  | const TCHAR\* path, /\* Logical drive number to be mounted/unmounted \*/ |
|  | BYTE opt /\* 0:Do not mount (delayed mount), 1:Mount immediately \*/ |
|  | ) |
|  | { |
|  | FATFS \*cfs; |
|  | int vol; |
|  | FRESULT res; |
|  | const TCHAR \*rp = path; |
|  |  |
|  |  |
|  | vol = get\_ldnumber(&rp); |
|  | if (vol < 0) return FR\_INVALID\_DRIVE; |
|  | cfs = FatFs[vol]; /\* Pointer to fs object \*/ |
|  |  |
|  | if (cfs) { |
|  | #if \_FS\_LOCK |
|  | clear\_lock(cfs); |
|  | #endif |
|  | #if \_FS\_REENTRANT /\* Discard sync object of the current volume \*/ |
|  | if (!ff\_del\_syncobj(cfs->sobj)) return FR\_INT\_ERR; |
|  | #endif |
|  | cfs->fs\_type = 0; /\* Clear old fs object \*/ |
|  | } |
|  |  |
|  | if (fs) { |
|  | fs->fs\_type = 0; /\* Clear new fs object \*/ |
|  | #if \_FS\_REENTRANT /\* Create sync object for the new volume \*/ |
|  | if (!ff\_cre\_syncobj((BYTE)vol, &fs->sobj)) return FR\_INT\_ERR; |
|  | #endif |
|  | } |
|  | FatFs[vol] = fs; /\* Register new fs object \*/ |
|  |  |
|  | if (!fs || opt != 1) return FR\_OK; /\* Do not mount now, it will be mounted later \*/ |
|  |  |
|  | res = find\_volume(&fs, &path, 0); /\* Force mounted the volume \*/ |
|  | LEAVE\_FF(fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Open or Create a File \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_open ( |
|  | FIL\* fp, /\* Pointer to the blank file object \*/ |
|  | const TCHAR\* path, /\* Pointer to the file name \*/ |
|  | BYTE mode /\* Access mode and file open mode flags \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | BYTE \*dir; |
|  | DEFINE\_NAMEBUF; |
|  | #if !\_FS\_READONLY |
|  | DWORD dw, cl; |
|  | #endif |
|  |  |
|  |  |
|  | if (!fp) return FR\_INVALID\_OBJECT; |
|  | fp->fs = 0; /\* Clear file object \*/ |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | #if !\_FS\_READONLY |
|  | mode &= FA\_READ | FA\_WRITE | FA\_CREATE\_ALWAYS | FA\_OPEN\_ALWAYS | FA\_CREATE\_NEW; |
|  | res = find\_volume(&dj.fs, &path, (BYTE)(mode & ~FA\_READ)); |
|  | #else |
|  | mode &= FA\_READ; |
|  | res = find\_volume(&dj.fs, &path, 0); |
|  | #endif |
|  | if (res == FR\_OK) { |
|  | INIT\_BUF(dj); |
|  | res = follow\_path(&dj, path); /\* Follow the file path \*/ |
|  | dir = dj.dir; |
|  | #if !\_FS\_READONLY /\* R/W configuration \*/ |
|  | if (res == FR\_OK) { |
|  | if (!dir) /\* Default directory itself \*/ |
|  | res = FR\_INVALID\_NAME; |
|  | #if \_FS\_LOCK |
|  | else |
|  | res = chk\_lock(&dj, (mode & ~FA\_READ) ? 1 : 0); |
|  | #endif |
|  | } |
|  | /\* Create or Open a file \*/ |
|  | if (mode & (FA\_CREATE\_ALWAYS | FA\_OPEN\_ALWAYS | FA\_CREATE\_NEW)) { |
|  | if (res != FR\_OK) { /\* No file, create new \*/ |
|  | if (res == FR\_NO\_FILE) /\* There is no file to open, create a new entry \*/ |
|  | #if \_FS\_LOCK |
|  | res = enq\_lock() ? dir\_register(&dj) : FR\_TOO\_MANY\_OPEN\_FILES; |
|  | #else |
|  | res = dir\_register(&dj); |
|  | #endif |
|  | mode |= FA\_CREATE\_ALWAYS; /\* File is created \*/ |
|  | dir = dj.dir; /\* New entry \*/ |
|  | } |
|  | else { /\* Any object is already existing \*/ |
|  | if (dir[DIR\_Attr] & (AM\_RDO | AM\_DIR)) { /\* Cannot overwrite it (R/O or DIR) \*/ |
|  | res = FR\_DENIED; |
|  | } else { |
|  | if (mode & FA\_CREATE\_NEW) /\* Cannot create as new file \*/ |
|  | res = FR\_EXIST; |
|  | } |
|  | } |
|  | if (res == FR\_OK && (mode & FA\_CREATE\_ALWAYS)) { /\* Truncate it if overwrite mode \*/ |
|  | dw = GET\_FATTIME(); /\* Created time \*/ |
|  | ST\_DWORD(dir + DIR\_CrtTime, dw); |
|  | dir[DIR\_Attr] = 0; /\* Reset attribute \*/ |
|  | ST\_DWORD(dir + DIR\_FileSize, 0);/\* size = 0 \*/ |
|  | cl = ld\_clust(dj.fs, dir); /\* Get start cluster \*/ |
|  | st\_clust(dir, 0); /\* cluster = 0 \*/ |
|  | dj.fs->wflag = 1; |
|  | if (cl) { /\* Remove the cluster chain if exist \*/ |
|  | dw = dj.fs->winsect; |
|  | res = remove\_chain(dj.fs, cl); |
|  | if (res == FR\_OK) { |
|  | dj.fs->last\_clust = cl - 1; /\* Reuse the cluster hole \*/ |
|  | res = move\_window(dj.fs, dw); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | else { /\* Open an existing file \*/ |
|  | if (res == FR\_OK) { /\* Follow succeeded \*/ |
|  | if (dir[DIR\_Attr] & AM\_DIR) { /\* It is a directory \*/ |
|  | res = FR\_NO\_FILE; |
|  | } else { |
|  | if ((mode & FA\_WRITE) && (dir[DIR\_Attr] & AM\_RDO)) /\* R/O violation \*/ |
|  | res = FR\_DENIED; |
|  | } |
|  | } |
|  | } |
|  | if (res == FR\_OK) { |
|  | if (mode & FA\_CREATE\_ALWAYS) /\* Set file change flag if created or overwritten \*/ |
|  | mode |= FA\_\_WRITTEN; |
|  | fp->dir\_sect = dj.fs->winsect; /\* Pointer to the directory entry \*/ |
|  | fp->dir\_ptr = dir; |
|  | #if \_FS\_LOCK |
|  | fp->lockid = inc\_lock(&dj, (mode & ~FA\_READ) ? 1 : 0); |
|  | if (!fp->lockid) res = FR\_INT\_ERR; |
|  | #endif |
|  | } |
|  |  |
|  | #else /\* R/O configuration \*/ |
|  | if (res == FR\_OK) { /\* Follow succeeded \*/ |
|  | dir = dj.dir; |
|  | if (!dir) { /\* Current directory itself \*/ |
|  | res = FR\_INVALID\_NAME; |
|  | } else { |
|  | if (dir[DIR\_Attr] & AM\_DIR) /\* It is a directory \*/ |
|  | res = FR\_NO\_FILE; |
|  | } |
|  | } |
|  | #endif |
|  | FREE\_BUF(); |
|  |  |
|  | if (res == FR\_OK) { |
|  | fp->flag = mode; /\* File access mode \*/ |
|  | fp->err = 0; /\* Clear error flag \*/ |
|  | fp->sclust = ld\_clust(dj.fs, dir); /\* File start cluster \*/ |
|  | fp->fsize = LD\_DWORD(dir + DIR\_FileSize); /\* File size \*/ |
|  | fp->fptr = 0; /\* File pointer \*/ |
|  | fp->dsect = 0; |
|  | #if \_USE\_FASTSEEK |
|  | fp->cltbl = 0; /\* Normal seek mode \*/ |
|  | #endif |
|  | fp->fs = dj.fs; /\* Validate file object \*/ |
|  | fp->id = fp->fs->id; |
|  | } |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Read File \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_read ( |
|  | FIL\* fp, /\* Pointer to the file object \*/ |
|  | void\* buff, /\* Pointer to data buffer \*/ |
|  | UINT btr, /\* Number of bytes to read \*/ |
|  | UINT\* br /\* Pointer to number of bytes read \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DWORD clst, sect, remain; |
|  | UINT rcnt, cc; |
|  | BYTE csect, \*rbuff = (BYTE\*)buff; |
|  |  |
|  |  |
|  | \*br = 0; /\* Clear read byte counter \*/ |
|  |  |
|  | res = validate(fp); /\* Check validity \*/ |
|  | if (res != FR\_OK) LEAVE\_FF(fp->fs, res); |
|  | if (fp->err) /\* Check error \*/ |
|  | LEAVE\_FF(fp->fs, (FRESULT)fp->err); |
|  | if (!(fp->flag & FA\_READ)) /\* Check access mode \*/ |
|  | LEAVE\_FF(fp->fs, FR\_DENIED); |
|  | remain = fp->fsize - fp->fptr; |
|  | if (btr > remain) btr = (UINT)remain; /\* Truncate btr by remaining bytes \*/ |
|  |  |
|  | for ( ; btr; /\* Repeat until all data read \*/ |
|  | rbuff += rcnt, fp->fptr += rcnt, \*br += rcnt, btr -= rcnt) { |
|  | if ((fp->fptr % SS(fp->fs)) == 0) { /\* On the sector boundary? \*/ |
|  | csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1)); /\* Sector offset in the cluster \*/ |
|  | if (!csect) { /\* On the cluster boundary? \*/ |
|  | if (fp->fptr == 0) { /\* On the top of the file? \*/ |
|  | clst = fp->sclust; /\* Follow from the origin \*/ |
|  | } else { /\* Middle or end of the file \*/ |
|  | #if \_USE\_FASTSEEK |
|  | if (fp->cltbl) |
|  | clst = clmt\_clust(fp, fp->fptr); /\* Get cluster# from the CLMT \*/ |
|  | else |
|  | #endif |
|  | clst = get\_fat(fp->fs, fp->clust); /\* Follow cluster chain on the FAT \*/ |
|  | } |
|  | if (clst < 2) ABORT(fp->fs, FR\_INT\_ERR); |
|  | if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->clust = clst; /\* Update current cluster \*/ |
|  | } |
|  | sect = clust2sect(fp->fs, fp->clust); /\* Get current sector \*/ |
|  | if (!sect) ABORT(fp->fs, FR\_INT\_ERR); |
|  | sect += csect; |
|  | cc = btr / SS(fp->fs); /\* When remaining bytes >= sector size, \*/ |
|  | if (cc) { /\* Read maximum contiguous sectors directly \*/ |
|  | if (csect + cc > fp->fs->csize) /\* Clip at cluster boundary \*/ |
|  | cc = fp->fs->csize - csect; |
|  | if (disk\_read(fp->fs->drv, rbuff, sect, cc) != RES\_OK) |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | #if !\_FS\_READONLY && \_FS\_MINIMIZE <= 2 /\* Replace one of the read sectors with cached data if it contains a dirty sector \*/ |
|  | #if \_FS\_TINY |
|  | if (fp->fs->wflag && fp->fs->winsect - sect < cc) |
|  | mem\_cpy(rbuff + ((fp->fs->winsect - sect) \* SS(fp->fs)), fp->fs->win, SS(fp->fs)); |
|  | #else |
|  | if ((fp->flag & FA\_\_DIRTY) && fp->dsect - sect < cc) |
|  | mem\_cpy(rbuff + ((fp->dsect - sect) \* SS(fp->fs)), fp->buf, SS(fp->fs)); |
|  | #endif |
|  | #endif |
|  | rcnt = SS(fp->fs) \* cc; /\* Number of bytes transferred \*/ |
|  | continue; |
|  | } |
|  | #if !\_FS\_TINY |
|  | if (fp->dsect != sect) { /\* Load data sector if not in cache \*/ |
|  | #if !\_FS\_READONLY |
|  | if (fp->flag & FA\_\_DIRTY) { /\* Write-back dirty sector cache \*/ |
|  | if (disk\_write(fp->fs->drv, fp->buf, fp->dsect, 1) != RES\_OK) |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->flag &= ~FA\_\_DIRTY; |
|  | } |
|  | #endif |
|  | if (disk\_read(fp->fs->drv, fp->buf, sect, 1) != RES\_OK) /\* Fill sector cache \*/ |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | } |
|  | #endif |
|  | fp->dsect = sect; |
|  | } |
|  | rcnt = SS(fp->fs) - ((UINT)fp->fptr % SS(fp->fs)); /\* Get partial sector data from sector buffer \*/ |
|  | if (rcnt > btr) rcnt = btr; |
|  | #if \_FS\_TINY |
|  | if (move\_window(fp->fs, fp->dsect) != FR\_OK) /\* Move sector window \*/ |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | mem\_cpy(rbuff, &fp->fs->win[fp->fptr % SS(fp->fs)], rcnt); /\* Pick partial sector \*/ |
|  | #else |
|  | mem\_cpy(rbuff, &fp->buf[fp->fptr % SS(fp->fs)], rcnt); /\* Pick partial sector \*/ |
|  | #endif |
|  | } |
|  |  |
|  | LEAVE\_FF(fp->fs, FR\_OK); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | #if !\_FS\_READONLY |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Write File \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_write ( |
|  | FIL\* fp, /\* Pointer to the file object \*/ |
|  | const void \*buff, /\* Pointer to the data to be written \*/ |
|  | UINT btw, /\* Number of bytes to write \*/ |
|  | UINT\* bw /\* Pointer to number of bytes written \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DWORD clst, sect; |
|  | UINT wcnt, cc; |
|  | const BYTE \*wbuff = (const BYTE\*)buff; |
|  | BYTE csect; |
|  |  |
|  |  |
|  | \*bw = 0; /\* Clear write byte counter \*/ |
|  |  |
|  | res = validate(fp); /\* Check validity \*/ |
|  | if (res != FR\_OK) LEAVE\_FF(fp->fs, res); |
|  | if (fp->err) /\* Check error \*/ |
|  | LEAVE\_FF(fp->fs, (FRESULT)fp->err); |
|  | if (!(fp->flag & FA\_WRITE)) /\* Check access mode \*/ |
|  | LEAVE\_FF(fp->fs, FR\_DENIED); |
|  | if (fp->fptr + btw < fp->fptr) btw = 0; /\* File size cannot reach 4GB \*/ |
|  |  |
|  | for ( ; btw; /\* Repeat until all data written \*/ |
|  | wbuff += wcnt, fp->fptr += wcnt, \*bw += wcnt, btw -= wcnt) { |
|  | if ((fp->fptr % SS(fp->fs)) == 0) { /\* On the sector boundary? \*/ |
|  | csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1)); /\* Sector offset in the cluster \*/ |
|  | if (!csect) { /\* On the cluster boundary? \*/ |
|  | if (fp->fptr == 0) { /\* On the top of the file? \*/ |
|  | clst = fp->sclust; /\* Follow from the origin \*/ |
|  | if (clst == 0) /\* When no cluster is allocated, \*/ |
|  | clst = create\_chain(fp->fs, 0); /\* Create a new cluster chain \*/ |
|  | } else { /\* Middle or end of the file \*/ |
|  | #if \_USE\_FASTSEEK |
|  | if (fp->cltbl) |
|  | clst = clmt\_clust(fp, fp->fptr); /\* Get cluster# from the CLMT \*/ |
|  | else |
|  | #endif |
|  | clst = create\_chain(fp->fs, fp->clust); /\* Follow or stretch cluster chain on the FAT \*/ |
|  | } |
|  | if (clst == 0) break; /\* Could not allocate a new cluster (disk full) \*/ |
|  | if (clst == 1) ABORT(fp->fs, FR\_INT\_ERR); |
|  | if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->clust = clst; /\* Update current cluster \*/ |
|  | if (fp->sclust == 0) fp->sclust = clst; /\* Set start cluster if the first write \*/ |
|  | } |
|  | #if \_FS\_TINY |
|  | if (fp->fs->winsect == fp->dsect && sync\_window(fp->fs)) /\* Write-back sector cache \*/ |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | #else |
|  | if (fp->flag & FA\_\_DIRTY) { /\* Write-back sector cache \*/ |
|  | if (disk\_write(fp->fs->drv, fp->buf, fp->dsect, 1) != RES\_OK) |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->flag &= ~FA\_\_DIRTY; |
|  | } |
|  | #endif |
|  | sect = clust2sect(fp->fs, fp->clust); /\* Get current sector \*/ |
|  | if (!sect) ABORT(fp->fs, FR\_INT\_ERR); |
|  | sect += csect; |
|  | cc = btw / SS(fp->fs); /\* When remaining bytes >= sector size, \*/ |
|  | if (cc) { /\* Write maximum contiguous sectors directly \*/ |
|  | if (csect + cc > fp->fs->csize) /\* Clip at cluster boundary \*/ |
|  | cc = fp->fs->csize - csect; |
|  | if (disk\_write(fp->fs->drv, wbuff, sect, cc) != RES\_OK) |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | #if \_FS\_MINIMIZE <= 2 |
|  | #if \_FS\_TINY |
|  | if (fp->fs->winsect - sect < cc) { /\* Refill sector cache if it gets invalidated by the direct write \*/ |
|  | mem\_cpy(fp->fs->win, wbuff + ((fp->fs->winsect - sect) \* SS(fp->fs)), SS(fp->fs)); |
|  | fp->fs->wflag = 0; |
|  | } |
|  | #else |
|  | if (fp->dsect - sect < cc) { /\* Refill sector cache if it gets invalidated by the direct write \*/ |
|  | mem\_cpy(fp->buf, wbuff + ((fp->dsect - sect) \* SS(fp->fs)), SS(fp->fs)); |
|  | fp->flag &= ~FA\_\_DIRTY; |
|  | } |
|  | #endif |
|  | #endif |
|  | wcnt = SS(fp->fs) \* cc; /\* Number of bytes transferred \*/ |
|  | continue; |
|  | } |
|  | #if \_FS\_TINY |
|  | if (fp->fptr >= fp->fsize) { /\* Avoid silly cache filling at growing edge \*/ |
|  | if (sync\_window(fp->fs)) ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->fs->winsect = sect; |
|  | } |
|  | #else |
|  | if (fp->dsect != sect) { /\* Fill sector cache with file data \*/ |
|  | if (fp->fptr < fp->fsize && |
|  | disk\_read(fp->fs->drv, fp->buf, sect, 1) != RES\_OK) |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | } |
|  | #endif |
|  | fp->dsect = sect; |
|  | } |
|  | wcnt = SS(fp->fs) - ((UINT)fp->fptr % SS(fp->fs));/\* Put partial sector into file I/O buffer \*/ |
|  | if (wcnt > btw) wcnt = btw; |
|  | #if \_FS\_TINY |
|  | if (move\_window(fp->fs, fp->dsect) != FR\_OK) /\* Move sector window \*/ |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | mem\_cpy(&fp->fs->win[fp->fptr % SS(fp->fs)], wbuff, wcnt); /\* Fit partial sector \*/ |
|  | fp->fs->wflag = 1; |
|  | #else |
|  | mem\_cpy(&fp->buf[fp->fptr % SS(fp->fs)], wbuff, wcnt); /\* Fit partial sector \*/ |
|  | fp->flag |= FA\_\_DIRTY; |
|  | #endif |
|  | } |
|  |  |
|  | if (fp->fptr > fp->fsize) fp->fsize = fp->fptr; /\* Update file size if needed \*/ |
|  | fp->flag |= FA\_\_WRITTEN; /\* Set file change flag \*/ |
|  |  |
|  | LEAVE\_FF(fp->fs, FR\_OK); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Synchronize the File \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_sync ( |
|  | FIL\* fp /\* Pointer to the file object \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DWORD tm; |
|  | BYTE \*dir; |
|  |  |
|  |  |
|  | res = validate(fp); /\* Check validity of the object \*/ |
|  | if (res == FR\_OK) { |
|  | if (fp->flag & FA\_\_WRITTEN) { /\* Has the file been written? \*/ |
|  | /\* Write-back dirty buffer \*/ |
|  | #if !\_FS\_TINY |
|  | if (fp->flag & FA\_\_DIRTY) { |
|  | if (disk\_write(fp->fs->drv, fp->buf, fp->dsect, 1) != RES\_OK) |
|  | LEAVE\_FF(fp->fs, FR\_DISK\_ERR); |
|  | fp->flag &= ~FA\_\_DIRTY; |
|  | } |
|  | #endif |
|  | /\* Update the directory entry \*/ |
|  | res = move\_window(fp->fs, fp->dir\_sect); |
|  | if (res == FR\_OK) { |
|  | dir = fp->dir\_ptr; |
|  | dir[DIR\_Attr] |= AM\_ARC; /\* Set archive bit \*/ |
|  | ST\_DWORD(dir + DIR\_FileSize, fp->fsize); /\* Update file size \*/ |
|  | st\_clust(dir, fp->sclust); /\* Update start cluster \*/ |
|  | tm = GET\_FATTIME(); /\* Update updated time \*/ |
|  | ST\_DWORD(dir + DIR\_WrtTime, tm); |
|  | ST\_WORD(dir + DIR\_LstAccDate, 0); |
|  | fp->flag &= ~FA\_\_WRITTEN; |
|  | fp->fs->wflag = 1; |
|  | res = sync\_fs(fp->fs); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | LEAVE\_FF(fp->fs, res); |
|  | } |
|  |  |
|  | #endif /\* !\_FS\_READONLY \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Close File \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_close ( |
|  | FIL \*fp /\* Pointer to the file object to be closed \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  |  |
|  |  |
|  | #if !\_FS\_READONLY |
|  | res = f\_sync(fp); /\* Flush cached data \*/ |
|  | if (res == FR\_OK) |
|  | #endif |
|  | { |
|  | res = validate(fp); /\* Lock volume \*/ |
|  | if (res == FR\_OK) { |
|  | #if \_FS\_REENTRANT |
|  | FATFS \*fs = fp->fs; |
|  | #endif |
|  | #if \_FS\_LOCK |
|  | res = dec\_lock(fp->lockid); /\* Decrement file open counter \*/ |
|  | if (res == FR\_OK) |
|  | #endif |
|  | fp->fs = 0; /\* Invalidate file object \*/ |
|  | #if \_FS\_REENTRANT |
|  | unlock\_fs(fs, FR\_OK); /\* Unlock volume \*/ |
|  | #endif |
|  | } |
|  | } |
|  | return res; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Change Current Directory or Current Drive, Get Current Directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | #if \_FS\_RPATH >= 1 |
|  | #if \_VOLUMES >= 2 |
|  | FRESULT f\_chdrive ( |
|  | const TCHAR\* path /\* Drive number \*/ |
|  | ) |
|  | { |
|  | int vol; |
|  |  |
|  |  |
|  | vol = get\_ldnumber(&path); |
|  | if (vol < 0) return FR\_INVALID\_DRIVE; |
|  |  |
|  | CurrVol = (BYTE)vol; |
|  |  |
|  | return FR\_OK; |
|  | } |
|  | #endif |
|  |  |
|  |  |
|  | FRESULT f\_chdir ( |
|  | const TCHAR\* path /\* Pointer to the directory path \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, &path, 0); |
|  | if (res == FR\_OK) { |
|  | INIT\_BUF(dj); |
|  | res = follow\_path(&dj, path); /\* Follow the path \*/ |
|  | FREE\_BUF(); |
|  | if (res == FR\_OK) { /\* Follow completed \*/ |
|  | if (!dj.dir) { |
|  | dj.fs->cdir = dj.sclust; /\* Start directory itself \*/ |
|  | } else { |
|  | if (dj.dir[DIR\_Attr] & AM\_DIR) /\* Reached to the directory \*/ |
|  | dj.fs->cdir = ld\_clust(dj.fs, dj.dir); |
|  | else |
|  | res = FR\_NO\_PATH; /\* Reached but a file \*/ |
|  | } |
|  | } |
|  | if (res == FR\_NO\_FILE) res = FR\_NO\_PATH; |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  |  |
|  | #if \_FS\_RPATH >= 2 |
|  | FRESULT f\_getcwd ( |
|  | TCHAR\* buff, /\* Pointer to the directory path \*/ |
|  | UINT len /\* Size of path \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | UINT i, n; |
|  | DWORD ccl; |
|  | TCHAR \*tp; |
|  | FILINFO fno; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | \*buff = 0; |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, (const TCHAR\*\*)&buff, 0); /\* Get current volume \*/ |
|  | if (res == FR\_OK) { |
|  | INIT\_BUF(dj); |
|  | i = len; /\* Bottom of buffer (directory stack base) \*/ |
|  | dj.sclust = dj.fs->cdir; /\* Start to follow upper directory from current directory \*/ |
|  | while ((ccl = dj.sclust) != 0) { /\* Repeat while current directory is a sub-directory \*/ |
|  | res = dir\_sdi(&dj, 1); /\* Get parent directory \*/ |
|  | if (res != FR\_OK) break; |
|  | res = dir\_read(&dj, 0); |
|  | if (res != FR\_OK) break; |
|  | dj.sclust = ld\_clust(dj.fs, dj.dir); /\* Goto parent directory \*/ |
|  | res = dir\_sdi(&dj, 0); |
|  | if (res != FR\_OK) break; |
|  | do { /\* Find the entry links to the child directory \*/ |
|  | res = dir\_read(&dj, 0); |
|  | if (res != FR\_OK) break; |
|  | if (ccl == ld\_clust(dj.fs, dj.dir)) break; /\* Found the entry \*/ |
|  | res = dir\_next(&dj, 0); |
|  | } while (res == FR\_OK); |
|  | if (res == FR\_NO\_FILE) res = FR\_INT\_ERR;/\* It cannot be 'not found'. \*/ |
|  | if (res != FR\_OK) break; |
|  | #if \_USE\_LFN |
|  | fno.lfname = buff; |
|  | fno.lfsize = i; |
|  | #endif |
|  | get\_fileinfo(&dj, &fno); /\* Get the directory name and push it to the buffer \*/ |
|  | tp = fno.fname; |
|  | #if \_USE\_LFN |
|  | if (\*buff) tp = buff; |
|  | #endif |
|  | for (n = 0; tp[n]; n++) ; |
|  | if (i < n + 3) { |
|  | res = FR\_NOT\_ENOUGH\_CORE; break; |
|  | } |
|  | while (n) buff[--i] = tp[--n]; |
|  | buff[--i] = '/'; |
|  | } |
|  | tp = buff; |
|  | if (res == FR\_OK) { |
|  | #if \_VOLUMES >= 2 |
|  | \*tp++ = '0' + CurrVol; /\* Put drive number \*/ |
|  | \*tp++ = ':'; |
|  | #endif |
|  | if (i == len) { /\* Root-directory \*/ |
|  | \*tp++ = '/'; |
|  | } else { /\* Sub-directroy \*/ |
|  | do /\* Add stacked path str \*/ |
|  | \*tp++ = buff[i++]; |
|  | while (i < len); |
|  | } |
|  | } |
|  | \*tp = 0; |
|  | FREE\_BUF(); |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  | #endif /\* \_FS\_RPATH >= 2 \*/ |
|  | #endif /\* \_FS\_RPATH >= 1 \*/ |
|  |  |
|  |  |
|  |  |
|  | #if \_FS\_MINIMIZE <= 2 |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Seek File R/W Pointer \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_lseek ( |
|  | FIL\* fp, /\* Pointer to the file object \*/ |
|  | DWORD ofs /\* File pointer from top of file \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DWORD clst, bcs, nsect, ifptr; |
|  | #if \_USE\_FASTSEEK |
|  | DWORD cl, pcl, ncl, tcl, dsc, tlen, ulen, \*tbl; |
|  | #endif |
|  |  |
|  |  |
|  | res = validate(fp); /\* Check validity of the object \*/ |
|  | if (res != FR\_OK) LEAVE\_FF(fp->fs, res); |
|  | if (fp->err) /\* Check error \*/ |
|  | LEAVE\_FF(fp->fs, (FRESULT)fp->err); |
|  |  |
|  | #if \_USE\_FASTSEEK |
|  | if (fp->cltbl) { /\* Fast seek \*/ |
|  | if (ofs == CREATE\_LINKMAP) { /\* Create CLMT \*/ |
|  | tbl = fp->cltbl; |
|  | tlen = \*tbl++; ulen = 2; /\* Given table size and required table size \*/ |
|  | cl = fp->sclust; /\* Top of the chain \*/ |
|  | if (cl) { |
|  | do { |
|  | /\* Get a fragment \*/ |
|  | tcl = cl; ncl = 0; ulen += 2; /\* Top, length and used items \*/ |
|  | do { |
|  | pcl = cl; ncl++; |
|  | cl = get\_fat(fp->fs, cl); |
|  | if (cl <= 1) ABORT(fp->fs, FR\_INT\_ERR); |
|  | if (cl == 0xFFFFFFFF) ABORT(fp->fs, FR\_DISK\_ERR); |
|  | } while (cl == pcl + 1); |
|  | if (ulen <= tlen) { /\* Store the length and top of the fragment \*/ |
|  | \*tbl++ = ncl; \*tbl++ = tcl; |
|  | } |
|  | } while (cl < fp->fs->n\_fatent); /\* Repeat until end of chain \*/ |
|  | } |
|  | \*fp->cltbl = ulen; /\* Number of items used \*/ |
|  | if (ulen <= tlen) |
|  | \*tbl = 0; /\* Terminate table \*/ |
|  | else |
|  | res = FR\_NOT\_ENOUGH\_CORE; /\* Given table size is smaller than required \*/ |
|  |  |
|  | } else { /\* Fast seek \*/ |
|  | if (ofs > fp->fsize) /\* Clip offset at the file size \*/ |
|  | ofs = fp->fsize; |
|  | fp->fptr = ofs; /\* Set file pointer \*/ |
|  | if (ofs) { |
|  | fp->clust = clmt\_clust(fp, ofs - 1); |
|  | dsc = clust2sect(fp->fs, fp->clust); |
|  | if (!dsc) ABORT(fp->fs, FR\_INT\_ERR); |
|  | dsc += (ofs - 1) / SS(fp->fs) & (fp->fs->csize - 1); |
|  | if (fp->fptr % SS(fp->fs) && dsc != fp->dsect) { /\* Refill sector cache if needed \*/ |
|  | #if !\_FS\_TINY |
|  | #if !\_FS\_READONLY |
|  | if (fp->flag & FA\_\_DIRTY) { /\* Write-back dirty sector cache \*/ |
|  | if (disk\_write(fp->fs->drv, fp->buf, fp->dsect, 1) != RES\_OK) |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->flag &= ~FA\_\_DIRTY; |
|  | } |
|  | #endif |
|  | if (disk\_read(fp->fs->drv, fp->buf, dsc, 1) != RES\_OK) /\* Load current sector \*/ |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | #endif |
|  | fp->dsect = dsc; |
|  | } |
|  | } |
|  | } |
|  | } else |
|  | #endif |
|  |  |
|  | /\* Normal Seek \*/ |
|  | { |
|  | if (ofs > fp->fsize /\* In read-only mode, clip offset with the file size \*/ |
|  | #if !\_FS\_READONLY |
|  | && !(fp->flag & FA\_WRITE) |
|  | #endif |
|  | ) ofs = fp->fsize; |
|  |  |
|  | ifptr = fp->fptr; |
|  | fp->fptr = nsect = 0; |
|  | if (ofs) { |
|  | bcs = (DWORD)fp->fs->csize \* SS(fp->fs); /\* Cluster size (byte) \*/ |
|  | if (ifptr > 0 && |
|  | (ofs - 1) / bcs >= (ifptr - 1) / bcs) { /\* When seek to same or following cluster, \*/ |
|  | fp->fptr = (ifptr - 1) & ~(bcs - 1); /\* start from the current cluster \*/ |
|  | ofs -= fp->fptr; |
|  | clst = fp->clust; |
|  | } else { /\* When seek to back cluster, \*/ |
|  | clst = fp->sclust; /\* start from the first cluster \*/ |
|  | #if !\_FS\_READONLY |
|  | if (clst == 0) { /\* If no cluster chain, create a new chain \*/ |
|  | clst = create\_chain(fp->fs, 0); |
|  | if (clst == 1) ABORT(fp->fs, FR\_INT\_ERR); |
|  | if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->sclust = clst; |
|  | } |
|  | #endif |
|  | fp->clust = clst; |
|  | } |
|  | if (clst != 0) { |
|  | while (ofs > bcs) { /\* Cluster following loop \*/ |
|  | #if !\_FS\_READONLY |
|  | if (fp->flag & FA\_WRITE) { /\* Check if in write mode or not \*/ |
|  | clst = create\_chain(fp->fs, clst); /\* Force stretch if in write mode \*/ |
|  | if (clst == 0) { /\* When disk gets full, clip file size \*/ |
|  | ofs = bcs; break; |
|  | } |
|  | } else |
|  | #endif |
|  | clst = get\_fat(fp->fs, clst); /\* Follow cluster chain if not in write mode \*/ |
|  | if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR\_DISK\_ERR); |
|  | if (clst <= 1 || clst >= fp->fs->n\_fatent) ABORT(fp->fs, FR\_INT\_ERR); |
|  | fp->clust = clst; |
|  | fp->fptr += bcs; |
|  | ofs -= bcs; |
|  | } |
|  | fp->fptr += ofs; |
|  | if (ofs % SS(fp->fs)) { |
|  | nsect = clust2sect(fp->fs, clst); /\* Current sector \*/ |
|  | if (!nsect) ABORT(fp->fs, FR\_INT\_ERR); |
|  | nsect += ofs / SS(fp->fs); |
|  | } |
|  | } |
|  | } |
|  | if (fp->fptr % SS(fp->fs) && nsect != fp->dsect) { /\* Fill sector cache if needed \*/ |
|  | #if !\_FS\_TINY |
|  | #if !\_FS\_READONLY |
|  | if (fp->flag & FA\_\_DIRTY) { /\* Write-back dirty sector cache \*/ |
|  | if (disk\_write(fp->fs->drv, fp->buf, fp->dsect, 1) != RES\_OK) |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->flag &= ~FA\_\_DIRTY; |
|  | } |
|  | #endif |
|  | if (disk\_read(fp->fs->drv, fp->buf, nsect, 1) != RES\_OK) /\* Fill sector cache \*/ |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | #endif |
|  | fp->dsect = nsect; |
|  | } |
|  | #if !\_FS\_READONLY |
|  | if (fp->fptr > fp->fsize) { /\* Set file change flag if the file size is extended \*/ |
|  | fp->fsize = fp->fptr; |
|  | fp->flag |= FA\_\_WRITTEN; |
|  | } |
|  | #endif |
|  | } |
|  |  |
|  | LEAVE\_FF(fp->fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | #if \_FS\_MINIMIZE <= 1 |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Create a Directory Object \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_opendir ( |
|  | DIR\* dp, /\* Pointer to directory object to create \*/ |
|  | const TCHAR\* path /\* Pointer to the directory path \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | FATFS\* fs; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | if (!dp) return FR\_INVALID\_OBJECT; |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&fs, &path, 0); |
|  | if (res == FR\_OK) { |
|  | dp->fs = fs; |
|  | INIT\_BUF(\*dp); |
|  | res = follow\_path(dp, path); /\* Follow the path to the directory \*/ |
|  | FREE\_BUF(); |
|  | if (res == FR\_OK) { /\* Follow completed \*/ |
|  | if (dp->dir) { /\* It is not the origin directory itself \*/ |
|  | if (dp->dir[DIR\_Attr] & AM\_DIR) /\* The object is a sub directory \*/ |
|  | dp->sclust = ld\_clust(fs, dp->dir); |
|  | else /\* The object is a file \*/ |
|  | res = FR\_NO\_PATH; |
|  | } |
|  | if (res == FR\_OK) { |
|  | dp->id = fs->id; |
|  | res = dir\_sdi(dp, 0); /\* Rewind directory \*/ |
|  | #if \_FS\_LOCK |
|  | if (res == FR\_OK) { |
|  | if (dp->sclust) { |
|  | dp->lockid = inc\_lock(dp, 0); /\* Lock the sub directory \*/ |
|  | if (!dp->lockid) |
|  | res = FR\_TOO\_MANY\_OPEN\_FILES; |
|  | } else { |
|  | dp->lockid = 0; /\* Root directory need not to be locked \*/ |
|  | } |
|  | } |
|  | #endif |
|  | } |
|  | } |
|  | if (res == FR\_NO\_FILE) res = FR\_NO\_PATH; |
|  | } |
|  | if (res != FR\_OK) dp->fs = 0; /\* Invalidate the directory object if function faild \*/ |
|  |  |
|  | LEAVE\_FF(fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Close Directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_closedir ( |
|  | DIR \*dp /\* Pointer to the directory object to be closed \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  |  |
|  |  |
|  | res = validate(dp); |
|  | if (res == FR\_OK) { |
|  | #if \_FS\_REENTRANT |
|  | FATFS \*fs = dp->fs; |
|  | #endif |
|  | #if \_FS\_LOCK |
|  | if (dp->lockid) /\* Decrement sub-directory open counter \*/ |
|  | res = dec\_lock(dp->lockid); |
|  | if (res == FR\_OK) |
|  | #endif |
|  | dp->fs = 0; /\* Invalidate directory object \*/ |
|  | #if \_FS\_REENTRANT |
|  | unlock\_fs(fs, FR\_OK); /\* Unlock volume \*/ |
|  | #endif |
|  | } |
|  | return res; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Read Directory Entries in Sequence \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_readdir ( |
|  | DIR\* dp, /\* Pointer to the open directory object \*/ |
|  | FILINFO\* fno /\* Pointer to file information to return \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | res = validate(dp); /\* Check validity of the object \*/ |
|  | if (res == FR\_OK) { |
|  | if (!fno) { |
|  | res = dir\_sdi(dp, 0); /\* Rewind the directory object \*/ |
|  | } else { |
|  | INIT\_BUF(\*dp); |
|  | res = dir\_read(dp, 0); /\* Read an item \*/ |
|  | if (res == FR\_NO\_FILE) { /\* Reached end of directory \*/ |
|  | dp->sect = 0; |
|  | res = FR\_OK; |
|  | } |
|  | if (res == FR\_OK) { /\* A valid entry is found \*/ |
|  | get\_fileinfo(dp, fno); /\* Get the object information \*/ |
|  | res = dir\_next(dp, 0); /\* Increment index for next \*/ |
|  | if (res == FR\_NO\_FILE) { |
|  | dp->sect = 0; |
|  | res = FR\_OK; |
|  | } |
|  | } |
|  | FREE\_BUF(); |
|  | } |
|  | } |
|  |  |
|  | LEAVE\_FF(dp->fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | #if \_USE\_FIND |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Find next file \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_findnext ( |
|  | DIR\* dp, /\* Pointer to the open directory object \*/ |
|  | FILINFO\* fno /\* Pointer to the file information structure \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  |  |
|  |  |
|  | for (;;) { |
|  | res = f\_readdir(dp, fno); /\* Get a directory item \*/ |
|  | if (res != FR\_OK || !fno || !fno->fname[0]) break; /\* Terminate if any error or end of directory \*/ |
|  | #if \_USE\_LFN |
|  | if (fno->lfname && pattern\_matching(dp->pat, fno->lfname, 0, 0)) break; /\* Test for LFN if exist \*/ |
|  | #endif |
|  | if (pattern\_matching(dp->pat, fno->fname, 0, 0)) break; /\* Test for SFN \*/ |
|  | } |
|  | return res; |
|  |  |
|  | } |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Find first file \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_findfirst ( |
|  | DIR\* dp, /\* Pointer to the blank directory object \*/ |
|  | FILINFO\* fno, /\* Pointer to the file information structure \*/ |
|  | const TCHAR\* path, /\* Pointer to the directory to open \*/ |
|  | const TCHAR\* pattern /\* Pointer to the matching pattern \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  |  |
|  |  |
|  | dp->pat = pattern; /\* Save pointer to pattern string \*/ |
|  | res = f\_opendir(dp, path); /\* Open the target directory \*/ |
|  | if (res == FR\_OK) |
|  | res = f\_findnext(dp, fno); /\* Find the first item \*/ |
|  | return res; |
|  | } |
|  |  |
|  | #endif /\* \_USE\_FIND \*/ |
|  |  |
|  |  |
|  |  |
|  | #if \_FS\_MINIMIZE == 0 |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Get File Status \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_stat ( |
|  | const TCHAR\* path, /\* Pointer to the file path \*/ |
|  | FILINFO\* fno /\* Pointer to file information to return \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, &path, 0); |
|  | if (res == FR\_OK) { |
|  | INIT\_BUF(dj); |
|  | res = follow\_path(&dj, path); /\* Follow the file path \*/ |
|  | if (res == FR\_OK) { /\* Follow completed \*/ |
|  | if (dj.dir) { /\* Found an object \*/ |
|  | if (fno) get\_fileinfo(&dj, fno); |
|  | } else { /\* It is root directory \*/ |
|  | res = FR\_INVALID\_NAME; |
|  | } |
|  | } |
|  | FREE\_BUF(); |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | #if !\_FS\_READONLY |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Get Number of Free Clusters \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_getfree ( |
|  | const TCHAR\* path, /\* Path name of the logical drive number \*/ |
|  | DWORD\* nclst, /\* Pointer to a variable to return number of free clusters \*/ |
|  | FATFS\*\* fatfs /\* Pointer to return pointer to corresponding file system object \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | FATFS \*fs; |
|  | DWORD n, clst, sect, stat; |
|  | UINT i; |
|  | BYTE fat, \*p; |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(fatfs, &path, 0); |
|  | fs = \*fatfs; |
|  | if (res == FR\_OK) { |
|  | /\* If free\_clust is valid, return it without full cluster scan \*/ |
|  | if (fs->free\_clust <= fs->n\_fatent - 2) { |
|  | \*nclst = fs->free\_clust; |
|  | } else { |
|  | /\* Get number of free clusters \*/ |
|  | fat = fs->fs\_type; |
|  | n = 0; |
|  | if (fat == FS\_FAT12) { |
|  | clst = 2; |
|  | do { |
|  | stat = get\_fat(fs, clst); |
|  | if (stat == 0xFFFFFFFF) { res = FR\_DISK\_ERR; break; } |
|  | if (stat == 1) { res = FR\_INT\_ERR; break; } |
|  | if (stat == 0) n++; |
|  | } while (++clst < fs->n\_fatent); |
|  | } else { |
|  | clst = fs->n\_fatent; |
|  | sect = fs->fatbase; |
|  | i = 0; p = 0; |
|  | do { |
|  | if (!i) { |
|  | res = move\_window(fs, sect++); |
|  | if (res != FR\_OK) break; |
|  | p = fs->win; |
|  | i = SS(fs); |
|  | } |
|  | if (fat == FS\_FAT16) { |
|  | if (LD\_WORD(p) == 0) n++; |
|  | p += 2; i -= 2; |
|  | } else { |
|  | if ((LD\_DWORD(p) & 0x0FFFFFFF) == 0) n++; |
|  | p += 4; i -= 4; |
|  | } |
|  | } while (--clst); |
|  | } |
|  | fs->free\_clust = n; |
|  | fs->fsi\_flag |= 1; |
|  | \*nclst = n; |
|  | } |
|  | } |
|  | LEAVE\_FF(fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Truncate File \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_truncate ( |
|  | FIL\* fp /\* Pointer to the file object \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DWORD ncl; |
|  |  |
|  |  |
|  | res = validate(fp); /\* Check validity of the object \*/ |
|  | if (res == FR\_OK) { |
|  | if (fp->err) { /\* Check error \*/ |
|  | res = (FRESULT)fp->err; |
|  | } else { |
|  | if (!(fp->flag & FA\_WRITE)) /\* Check access mode \*/ |
|  | res = FR\_DENIED; |
|  | } |
|  | } |
|  | if (res == FR\_OK) { |
|  | if (fp->fsize > fp->fptr) { |
|  | fp->fsize = fp->fptr; /\* Set file size to current R/W point \*/ |
|  | fp->flag |= FA\_\_WRITTEN; |
|  | if (fp->fptr == 0) { /\* When set file size to zero, remove entire cluster chain \*/ |
|  | res = remove\_chain(fp->fs, fp->sclust); |
|  | fp->sclust = 0; |
|  | } else { /\* When truncate a part of the file, remove remaining clusters \*/ |
|  | ncl = get\_fat(fp->fs, fp->clust); |
|  | res = FR\_OK; |
|  | if (ncl == 0xFFFFFFFF) res = FR\_DISK\_ERR; |
|  | if (ncl == 1) res = FR\_INT\_ERR; |
|  | if (res == FR\_OK && ncl < fp->fs->n\_fatent) { |
|  | res = put\_fat(fp->fs, fp->clust, 0x0FFFFFFF); |
|  | if (res == FR\_OK) res = remove\_chain(fp->fs, ncl); |
|  | } |
|  | } |
|  | #if !\_FS\_TINY |
|  | if (res == FR\_OK && (fp->flag & FA\_\_DIRTY)) { |
|  | if (disk\_write(fp->fs->drv, fp->buf, fp->dsect, 1) != RES\_OK) |
|  | res = FR\_DISK\_ERR; |
|  | else |
|  | fp->flag &= ~FA\_\_DIRTY; |
|  | } |
|  | #endif |
|  | } |
|  | if (res != FR\_OK) fp->err = (FRESULT)res; |
|  | } |
|  |  |
|  | LEAVE\_FF(fp->fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Delete a File or Directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_unlink ( |
|  | const TCHAR\* path /\* Pointer to the file or directory path \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj, sdj; |
|  | BYTE \*dir; |
|  | DWORD dclst = 0; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, &path, 1); |
|  | if (res == FR\_OK) { |
|  | INIT\_BUF(dj); |
|  | res = follow\_path(&dj, path); /\* Follow the file path \*/ |
|  | if (\_FS\_RPATH && res == FR\_OK && (dj.fn[NSFLAG] & NS\_DOT)) |
|  | res = FR\_INVALID\_NAME; /\* Cannot remove dot entry \*/ |
|  | #if \_FS\_LOCK |
|  | if (res == FR\_OK) res = chk\_lock(&dj, 2); /\* Cannot remove open object \*/ |
|  | #endif |
|  | if (res == FR\_OK) { /\* The object is accessible \*/ |
|  | dir = dj.dir; |
|  | if (!dir) { |
|  | res = FR\_INVALID\_NAME; /\* Cannot remove the origin directory \*/ |
|  | } else { |
|  | if (dir[DIR\_Attr] & AM\_RDO) |
|  | res = FR\_DENIED; /\* Cannot remove R/O object \*/ |
|  | } |
|  | if (res == FR\_OK) { |
|  | dclst = ld\_clust(dj.fs, dir); |
|  | if (dclst && (dir[DIR\_Attr] & AM\_DIR)) { /\* Is it a sub-directory ? \*/ |
|  | #if \_FS\_RPATH |
|  | if (dclst == dj.fs->cdir) { /\* Is it the current directory? \*/ |
|  | res = FR\_DENIED; |
|  | } else |
|  | #endif |
|  | { |
|  | mem\_cpy(&sdj, &dj, sizeof (DIR)); /\* Open the sub-directory \*/ |
|  | sdj.sclust = dclst; |
|  | res = dir\_sdi(&sdj, 2); |
|  | if (res == FR\_OK) { |
|  | res = dir\_read(&sdj, 0); /\* Read an item (excluding dot entries) \*/ |
|  | if (res == FR\_OK) res = FR\_DENIED; /\* Not empty? (cannot remove) \*/ |
|  | if (res == FR\_NO\_FILE) res = FR\_OK; /\* Empty? (can remove) \*/ |
|  | } |
|  | } |
|  | } |
|  | } |
|  | if (res == FR\_OK) { |
|  | res = dir\_remove(&dj); /\* Remove the directory entry \*/ |
|  | if (res == FR\_OK && dclst) /\* Remove the cluster chain if exist \*/ |
|  | res = remove\_chain(dj.fs, dclst); |
|  | if (res == FR\_OK) res = sync\_fs(dj.fs); |
|  | } |
|  | } |
|  | FREE\_BUF(); |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Create a Directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_mkdir ( |
|  | const TCHAR\* path /\* Pointer to the directory path \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | BYTE \*dir, n; |
|  | DWORD dsc, dcl, pcl, tm = GET\_FATTIME(); |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, &path, 1); |
|  | if (res == FR\_OK) { |
|  | INIT\_BUF(dj); |
|  | res = follow\_path(&dj, path); /\* Follow the file path \*/ |
|  | if (res == FR\_OK) res = FR\_EXIST; /\* Any object with same name is already existing \*/ |
|  | if (\_FS\_RPATH && res == FR\_NO\_FILE && (dj.fn[NSFLAG] & NS\_DOT)) |
|  | res = FR\_INVALID\_NAME; |
|  | if (res == FR\_NO\_FILE) { /\* Can create a new directory \*/ |
|  | dcl = create\_chain(dj.fs, 0); /\* Allocate a cluster for the new directory table \*/ |
|  | res = FR\_OK; |
|  | if (dcl == 0) res = FR\_DENIED; /\* No space to allocate a new cluster \*/ |
|  | if (dcl == 1) res = FR\_INT\_ERR; |
|  | if (dcl == 0xFFFFFFFF) res = FR\_DISK\_ERR; |
|  | if (res == FR\_OK) /\* Flush FAT \*/ |
|  | res = sync\_window(dj.fs); |
|  | if (res == FR\_OK) { /\* Initialize the new directory table \*/ |
|  | dsc = clust2sect(dj.fs, dcl); |
|  | dir = dj.fs->win; |
|  | mem\_set(dir, 0, SS(dj.fs)); |
|  | mem\_set(dir + DIR\_Name, ' ', 11); /\* Create "." entry \*/ |
|  | dir[DIR\_Name] = '.'; |
|  | dir[DIR\_Attr] = AM\_DIR; |
|  | ST\_DWORD(dir + DIR\_WrtTime, tm); |
|  | st\_clust(dir, dcl); |
|  | mem\_cpy(dir + SZ\_DIRE, dir, SZ\_DIRE); /\* Create ".." entry \*/ |
|  | dir[SZ\_DIRE + 1] = '.'; pcl = dj.sclust; |
|  | if (dj.fs->fs\_type == FS\_FAT32 && pcl == dj.fs->dirbase) |
|  | pcl = 0; |
|  | st\_clust(dir + SZ\_DIRE, pcl); |
|  | for (n = dj.fs->csize; n; n--) { /\* Write dot entries and clear following sectors \*/ |
|  | dj.fs->winsect = dsc++; |
|  | dj.fs->wflag = 1; |
|  | res = sync\_window(dj.fs); |
|  | if (res != FR\_OK) break; |
|  | mem\_set(dir, 0, SS(dj.fs)); |
|  | } |
|  | } |
|  | if (res == FR\_OK) res = dir\_register(&dj); /\* Register the object to the directoy \*/ |
|  | if (res != FR\_OK) { |
|  | remove\_chain(dj.fs, dcl); /\* Could not register, remove cluster chain \*/ |
|  | } else { |
|  | dir = dj.dir; |
|  | dir[DIR\_Attr] = AM\_DIR; /\* Attribute \*/ |
|  | ST\_DWORD(dir + DIR\_WrtTime, tm); /\* Created time \*/ |
|  | st\_clust(dir, dcl); /\* Table start cluster \*/ |
|  | dj.fs->wflag = 1; |
|  | res = sync\_fs(dj.fs); |
|  | } |
|  | } |
|  | FREE\_BUF(); |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Change Attribute \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_chmod ( |
|  | const TCHAR\* path, /\* Pointer to the file path \*/ |
|  | BYTE attr, /\* Attribute bits \*/ |
|  | BYTE mask /\* Attribute mask to change \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | BYTE \*dir; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, &path, 1); |
|  | if (res == FR\_OK) { |
|  | INIT\_BUF(dj); |
|  | res = follow\_path(&dj, path); /\* Follow the file path \*/ |
|  | FREE\_BUF(); |
|  | if (\_FS\_RPATH && res == FR\_OK && (dj.fn[NSFLAG] & NS\_DOT)) |
|  | res = FR\_INVALID\_NAME; |
|  | if (res == FR\_OK) { |
|  | dir = dj.dir; |
|  | if (!dir) { /\* Is it a root directory? \*/ |
|  | res = FR\_INVALID\_NAME; |
|  | } else { /\* File or sub directory \*/ |
|  | mask &= AM\_RDO|AM\_HID|AM\_SYS|AM\_ARC; /\* Valid attribute mask \*/ |
|  | dir[DIR\_Attr] = (attr & mask) | (dir[DIR\_Attr] & (BYTE)~mask); /\* Apply attribute change \*/ |
|  | dj.fs->wflag = 1; |
|  | res = sync\_fs(dj.fs); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Rename File/Directory \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_rename ( |
|  | const TCHAR\* path\_old, /\* Pointer to the object to be renamed \*/ |
|  | const TCHAR\* path\_new /\* Pointer to the new name \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR djo, djn; |
|  | BYTE buf[21], \*dir; |
|  | DWORD dw; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | /\* Get logical drive number of the source object \*/ |
|  | res = find\_volume(&djo.fs, &path\_old, 1); |
|  | if (res == FR\_OK) { |
|  | djn.fs = djo.fs; |
|  | INIT\_BUF(djo); |
|  | res = follow\_path(&djo, path\_old); /\* Check old object \*/ |
|  | if (\_FS\_RPATH && res == FR\_OK && (djo.fn[NSFLAG] & NS\_DOT)) |
|  | res = FR\_INVALID\_NAME; |
|  | #if \_FS\_LOCK |
|  | if (res == FR\_OK) res = chk\_lock(&djo, 2); |
|  | #endif |
|  | if (res == FR\_OK) { /\* Old object is found \*/ |
|  | if (!djo.dir) { /\* Is root dir? \*/ |
|  | res = FR\_NO\_FILE; |
|  | } else { |
|  | mem\_cpy(buf, djo.dir + DIR\_Attr, 21); /\* Save information about object except name \*/ |
|  | mem\_cpy(&djn, &djo, sizeof (DIR)); /\* Duplicate the directory object \*/ |
|  | if (get\_ldnumber(&path\_new) >= 0) /\* Snip drive number off and ignore it \*/ |
|  | res = follow\_path(&djn, path\_new); /\* and make sure if new object name is not conflicting \*/ |
|  | else |
|  | res = FR\_INVALID\_DRIVE; |
|  | if (res == FR\_OK) res = FR\_EXIST; /\* The new object name is already existing \*/ |
|  | if (res == FR\_NO\_FILE) { /\* It is a valid path and no name collision \*/ |
|  | res = dir\_register(&djn); /\* Register the new entry \*/ |
|  | if (res == FR\_OK) { |
|  | /\* Start of critical section where any interruption can cause a cross-link \*/ |
|  | dir = djn.dir; /\* Copy information about object except name \*/ |
|  | mem\_cpy(dir + 13, buf + 2, 19); |
|  | dir[DIR\_Attr] = buf[0] | AM\_ARC; |
|  | djo.fs->wflag = 1; |
|  | if ((dir[DIR\_Attr] & AM\_DIR) && djo.sclust != djn.sclust) { /\* Update .. entry in the sub-directory if needed \*/ |
|  | dw = clust2sect(djo.fs, ld\_clust(djo.fs, dir)); |
|  | if (!dw) { |
|  | res = FR\_INT\_ERR; |
|  | } else { |
|  | res = move\_window(djo.fs, dw); |
|  | dir = djo.fs->win + SZ\_DIRE \* 1; /\* Ptr to .. entry \*/ |
|  | if (res == FR\_OK && dir[1] == '.') { |
|  | st\_clust(dir, djn.sclust); |
|  | djo.fs->wflag = 1; |
|  | } |
|  | } |
|  | } |
|  | if (res == FR\_OK) { |
|  | res = dir\_remove(&djo); /\* Remove old entry \*/ |
|  | if (res == FR\_OK) |
|  | res = sync\_fs(djo.fs); |
|  | } |
|  | /\* End of critical section \*/ |
|  | } |
|  | } |
|  | } |
|  | } |
|  | FREE\_BUF(); |
|  | } |
|  |  |
|  | LEAVE\_FF(djo.fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Change Timestamp \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_utime ( |
|  | const TCHAR\* path, /\* Pointer to the file/directory name \*/ |
|  | const FILINFO\* fno /\* Pointer to the time stamp to be set \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | BYTE \*dir; |
|  | DEFINE\_NAMEBUF; |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, &path, 1); |
|  | if (res == FR\_OK) { |
|  | INIT\_BUF(dj); |
|  | res = follow\_path(&dj, path); /\* Follow the file path \*/ |
|  | FREE\_BUF(); |
|  | if (\_FS\_RPATH && res == FR\_OK && (dj.fn[NSFLAG] & NS\_DOT)) |
|  | res = FR\_INVALID\_NAME; |
|  | if (res == FR\_OK) { |
|  | dir = dj.dir; |
|  | if (!dir) { /\* Root directory \*/ |
|  | res = FR\_INVALID\_NAME; |
|  | } else { /\* File or sub-directory \*/ |
|  | ST\_WORD(dir + DIR\_WrtTime, fno->ftime); |
|  | ST\_WORD(dir + DIR\_WrtDate, fno->fdate); |
|  | dj.fs->wflag = 1; |
|  | res = sync\_fs(dj.fs); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  | #endif /\* !\_FS\_READONLY \*/ |
|  | #endif /\* \_FS\_MINIMIZE == 0 \*/ |
|  | #endif /\* \_FS\_MINIMIZE <= 1 \*/ |
|  | #endif /\* \_FS\_MINIMIZE <= 2 \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | #if \_USE\_LABEL |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Get volume label \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_getlabel ( |
|  | const TCHAR\* path, /\* Path name of the logical drive number \*/ |
|  | TCHAR\* label, /\* Pointer to a buffer to return the volume label \*/ |
|  | DWORD\* vsn /\* Pointer to a variable to return the volume serial number \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | UINT i, j; |
|  | #if \_USE\_LFN && \_LFN\_UNICODE |
|  | WCHAR w; |
|  | #endif |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, &path, 0); |
|  |  |
|  | /\* Get volume label \*/ |
|  | if (res == FR\_OK && label) { |
|  | dj.sclust = 0; /\* Open root directory \*/ |
|  | res = dir\_sdi(&dj, 0); |
|  | if (res == FR\_OK) { |
|  | res = dir\_read(&dj, 1); /\* Get an entry with AM\_VOL \*/ |
|  | if (res == FR\_OK) { /\* A volume label is exist \*/ |
|  | #if \_USE\_LFN && \_LFN\_UNICODE |
|  | i = j = 0; |
|  | do { |
|  | w = (i < 11) ? dj.dir[i++] : ' '; |
|  | if (IsDBCS1(w) && i < 11 && IsDBCS2(dj.dir[i])) |
|  | w = w << 8 | dj.dir[i++]; |
|  | label[j++] = ff\_convert(w, 1); /\* OEM -> Unicode \*/ |
|  | } while (j < 11); |
|  | #else |
|  | mem\_cpy(label, dj.dir, 11); |
|  | #endif |
|  | j = 11; |
|  | do { |
|  | label[j] = 0; |
|  | if (!j) break; |
|  | } while (label[--j] == ' '); |
|  | } |
|  | if (res == FR\_NO\_FILE) { /\* No label, return nul string \*/ |
|  | label[0] = 0; |
|  | res = FR\_OK; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | /\* Get volume serial number \*/ |
|  | if (res == FR\_OK && vsn) { |
|  | res = move\_window(dj.fs, dj.fs->volbase); |
|  | if (res == FR\_OK) { |
|  | i = dj.fs->fs\_type == FS\_FAT32 ? BS\_VolID32 : BS\_VolID; |
|  | \*vsn = LD\_DWORD(&dj.fs->win[i]); |
|  | } |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  |  |
|  |  |
|  | #if !\_FS\_READONLY |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Set volume label \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_setlabel ( |
|  | const TCHAR\* label /\* Pointer to the volume label to set \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DIR dj; |
|  | BYTE vn[11]; |
|  | UINT i, j, sl; |
|  | WCHAR w; |
|  | DWORD tm; |
|  |  |
|  |  |
|  | /\* Get logical drive number \*/ |
|  | res = find\_volume(&dj.fs, &label, 1); |
|  | if (res) LEAVE\_FF(dj.fs, res); |
|  |  |
|  | /\* Create a volume label in directory form \*/ |
|  | vn[0] = 0; |
|  | for (sl = 0; label[sl]; sl++) ; /\* Get name length \*/ |
|  | for ( ; sl && label[sl - 1] == ' '; sl--) ; /\* Remove trailing spaces \*/ |
|  | if (sl) { /\* Create volume label in directory form \*/ |
|  | i = j = 0; |
|  | do { |
|  | #if \_USE\_LFN && \_LFN\_UNICODE |
|  | w = ff\_convert(ff\_wtoupper(label[i++]), 0); |
|  | #else |
|  | w = (BYTE)label[i++]; |
|  | if (IsDBCS1(w)) |
|  | w = (j < 10 && i < sl && IsDBCS2(label[i])) ? w << 8 | (BYTE)label[i++] : 0; |
|  | #if \_USE\_LFN |
|  | w = ff\_convert(ff\_wtoupper(ff\_convert(w, 1)), 0); |
|  | #else |
|  | if (IsLower(w)) w -= 0x20; /\* To upper ASCII characters \*/ |
|  | #ifdef \_EXCVT |
|  | if (w >= 0x80) w = ExCvt[w - 0x80]; /\* To upper extended characters (SBCS cfg) \*/ |
|  | #else |
|  | if (!\_DF1S && w >= 0x80) w = 0; /\* Reject extended characters (ASCII cfg) \*/ |
|  | #endif |
|  | #endif |
|  | #endif |
|  | if (!w || chk\_chr("\"\*+,.:;<=>\?[]|\x7F", w) || j >= (UINT)((w >= 0x100) ? 10 : 11)) /\* Reject invalid characters for volume label \*/ |
|  | LEAVE\_FF(dj.fs, FR\_INVALID\_NAME); |
|  | if (w >= 0x100) vn[j++] = (BYTE)(w >> 8); |
|  | vn[j++] = (BYTE)w; |
|  | } while (i < sl); |
|  | while (j < 11) vn[j++] = ' '; /\* Fill remaining name field \*/ |
|  | if (vn[0] == DDEM) LEAVE\_FF(dj.fs, FR\_INVALID\_NAME); /\* Reject illegal name (heading DDEM) \*/ |
|  | } |
|  |  |
|  | /\* Set volume label \*/ |
|  | dj.sclust = 0; /\* Open root directory \*/ |
|  | res = dir\_sdi(&dj, 0); |
|  | if (res == FR\_OK) { |
|  | res = dir\_read(&dj, 1); /\* Get an entry with AM\_VOL \*/ |
|  | if (res == FR\_OK) { /\* A volume label is found \*/ |
|  | if (vn[0]) { |
|  | mem\_cpy(dj.dir, vn, 11); /\* Change the volume label name \*/ |
|  | tm = GET\_FATTIME(); |
|  | ST\_DWORD(dj.dir + DIR\_WrtTime, tm); |
|  | } else { |
|  | dj.dir[0] = DDEM; /\* Remove the volume label \*/ |
|  | } |
|  | dj.fs->wflag = 1; |
|  | res = sync\_fs(dj.fs); |
|  | } else { /\* No volume label is found or error \*/ |
|  | if (res == FR\_NO\_FILE) { |
|  | res = FR\_OK; |
|  | if (vn[0]) { /\* Create volume label as new \*/ |
|  | res = dir\_alloc(&dj, 1); /\* Allocate an entry for volume label \*/ |
|  | if (res == FR\_OK) { |
|  | mem\_set(dj.dir, 0, SZ\_DIRE); /\* Set volume label \*/ |
|  | mem\_cpy(dj.dir, vn, 11); |
|  | dj.dir[DIR\_Attr] = AM\_VOL; |
|  | tm = GET\_FATTIME(); |
|  | ST\_DWORD(dj.dir + DIR\_WrtTime, tm); |
|  | dj.fs->wflag = 1; |
|  | res = sync\_fs(dj.fs); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | LEAVE\_FF(dj.fs, res); |
|  | } |
|  |  |
|  | #endif /\* !\_FS\_READONLY \*/ |
|  | #endif /\* \_USE\_LABEL \*/ |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Forward data to the stream directly (available on only tiny cfg) \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #if \_USE\_FORWARD && \_FS\_TINY |
|  |  |
|  | FRESULT f\_forward ( |
|  | FIL\* fp, /\* Pointer to the file object \*/ |
|  | UINT (\*func)(const BYTE\*,UINT), /\* Pointer to the streaming function \*/ |
|  | UINT btf, /\* Number of bytes to forward \*/ |
|  | UINT\* bf /\* Pointer to number of bytes forwarded \*/ |
|  | ) |
|  | { |
|  | FRESULT res; |
|  | DWORD remain, clst, sect; |
|  | UINT rcnt; |
|  | BYTE csect; |
|  |  |
|  |  |
|  | \*bf = 0; /\* Clear transfer byte counter \*/ |
|  |  |
|  | res = validate(fp); /\* Check validity of the object \*/ |
|  | if (res != FR\_OK) LEAVE\_FF(fp->fs, res); |
|  | if (fp->err) /\* Check error \*/ |
|  | LEAVE\_FF(fp->fs, (FRESULT)fp->err); |
|  | if (!(fp->flag & FA\_READ)) /\* Check access mode \*/ |
|  | LEAVE\_FF(fp->fs, FR\_DENIED); |
|  |  |
|  | remain = fp->fsize - fp->fptr; |
|  | if (btf > remain) btf = (UINT)remain; /\* Truncate btf by remaining bytes \*/ |
|  |  |
|  | for ( ; btf && (\*func)(0, 0); /\* Repeat until all data transferred or stream becomes busy \*/ |
|  | fp->fptr += rcnt, \*bf += rcnt, btf -= rcnt) { |
|  | csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1)); /\* Sector offset in the cluster \*/ |
|  | if ((fp->fptr % SS(fp->fs)) == 0) { /\* On the sector boundary? \*/ |
|  | if (!csect) { /\* On the cluster boundary? \*/ |
|  | clst = (fp->fptr == 0) ? /\* On the top of the file? \*/ |
|  | fp->sclust : get\_fat(fp->fs, fp->clust); |
|  | if (clst <= 1) ABORT(fp->fs, FR\_INT\_ERR); |
|  | if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->clust = clst; /\* Update current cluster \*/ |
|  | } |
|  | } |
|  | sect = clust2sect(fp->fs, fp->clust); /\* Get current data sector \*/ |
|  | if (!sect) ABORT(fp->fs, FR\_INT\_ERR); |
|  | sect += csect; |
|  | if (move\_window(fp->fs, sect) != FR\_OK) /\* Move sector window \*/ |
|  | ABORT(fp->fs, FR\_DISK\_ERR); |
|  | fp->dsect = sect; |
|  | rcnt = SS(fp->fs) - (WORD)(fp->fptr % SS(fp->fs)); /\* Forward data from sector window \*/ |
|  | if (rcnt > btf) rcnt = btf; |
|  | rcnt = (\*func)(&fp->fs->win[(WORD)fp->fptr % SS(fp->fs)], rcnt); |
|  | if (!rcnt) ABORT(fp->fs, FR\_INT\_ERR); |
|  | } |
|  |  |
|  | LEAVE\_FF(fp->fs, FR\_OK); |
|  | } |
|  | #endif /\* \_USE\_FORWARD \*/ |
|  |  |
|  |  |
|  |  |
|  | #if \_USE\_MKFS && !\_FS\_READONLY |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Create file system on the logical drive \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | #define N\_ROOTDIR 512 /\* Number of root directory entries for FAT12/16 \*/ |
|  | #define N\_FATS 1 /\* Number of FATs (1 or 2) \*/ |
|  |  |
|  |  |
|  | FRESULT f\_mkfs ( |
|  | const TCHAR\* path, /\* Logical drive number \*/ |
|  | BYTE sfd, /\* Partitioning rule 0:FDISK, 1:SFD \*/ |
|  | UINT au /\* Size of allocation unit in unit of byte or sector \*/ |
|  | ) |
|  | { |
|  | static const WORD vst[] = { 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 0}; |
|  | static const WORD cst[] = {32768, 16384, 8192, 4096, 2048, 16384, 8192, 4096, 2048, 1024, 512}; |
|  | int vol; |
|  | BYTE fmt, md, sys, \*tbl, pdrv, part; |
|  | DWORD n\_clst, vs, n, wsect; |
|  | UINT i; |
|  | DWORD b\_vol, b\_fat, b\_dir, b\_data; /\* LBA \*/ |
|  | DWORD n\_vol, n\_rsv, n\_fat, n\_dir; /\* Size \*/ |
|  | FATFS \*fs; |
|  | DSTATUS stat; |
|  | #if \_USE\_TRIM |
|  | DWORD eb[2]; |
|  | #endif |
|  |  |
|  |  |
|  | /\* Check mounted drive and clear work area \*/ |
|  | if (sfd > 1) return FR\_INVALID\_PARAMETER; |
|  | vol = get\_ldnumber(&path); |
|  | if (vol < 0) return FR\_INVALID\_DRIVE; |
|  | fs = FatFs[vol]; |
|  | if (!fs) return FR\_NOT\_ENABLED; |
|  | fs->fs\_type = 0; |
|  | pdrv = LD2PD(vol); /\* Physical drive \*/ |
|  | part = LD2PT(vol); /\* Partition (0:auto detect, 1-4:get from partition table)\*/ |
|  |  |
|  | /\* Get disk statics \*/ |
|  | stat = disk\_initialize(pdrv); |
|  | if (stat & STA\_NOINIT) return FR\_NOT\_READY; |
|  | if (stat & STA\_PROTECT) return FR\_WRITE\_PROTECTED; |
|  | #if \_MAX\_SS != \_MIN\_SS /\* Get disk sector size \*/ |
|  | if (disk\_ioctl(pdrv, GET\_SECTOR\_SIZE, &SS(fs)) != RES\_OK || SS(fs) > \_MAX\_SS || SS(fs) < \_MIN\_SS) |
|  | return FR\_DISK\_ERR; |
|  | #endif |
|  | if (\_MULTI\_PARTITION && part) { |
|  | /\* Get partition information from partition table in the MBR \*/ |
|  | if (disk\_read(pdrv, fs->win, 0, 1) != RES\_OK) return FR\_DISK\_ERR; |
|  | if (LD\_WORD(fs->win + BS\_55AA) != 0xAA55) return FR\_MKFS\_ABORTED; |
|  | tbl = &fs->win[MBR\_Table + (part - 1) \* SZ\_PTE]; |
|  | if (!tbl[4]) return FR\_MKFS\_ABORTED; /\* No partition? \*/ |
|  | b\_vol = LD\_DWORD(tbl + 8); /\* Volume start sector \*/ |
|  | n\_vol = LD\_DWORD(tbl + 12); /\* Volume size \*/ |
|  | } else { |
|  | /\* Create a partition in this function \*/ |
|  | if (disk\_ioctl(pdrv, GET\_SECTOR\_COUNT, &n\_vol) != RES\_OK || n\_vol < 128) |
|  | return FR\_DISK\_ERR; |
|  | b\_vol = (sfd) ? 0 : 63; /\* Volume start sector \*/ |
|  | n\_vol -= b\_vol; /\* Volume size \*/ |
|  | } |
|  |  |
|  | if (au & (au - 1)) au = 0; |
|  | if (!au) { /\* AU auto selection \*/ |
|  | vs = n\_vol / (2000 / (SS(fs) / 512)); |
|  | for (i = 0; vs < vst[i]; i++) ; |
|  | au = cst[i]; |
|  | } |
|  | if (au >= \_MIN\_SS) au /= SS(fs); /\* Number of sectors per cluster \*/ |
|  | if (!au) au = 1; |
|  | if (au > 128) au = 128; |
|  |  |
|  | /\* Pre-compute number of clusters and FAT sub-type \*/ |
|  | n\_clst = n\_vol / au; |
|  | fmt = FS\_FAT12; |
|  | if (n\_clst >= MIN\_FAT16) fmt = FS\_FAT16; |
|  | if (n\_clst >= MIN\_FAT32) fmt = FS\_FAT32; |
|  |  |
|  | /\* Determine offset and size of FAT structure \*/ |
|  | if (fmt == FS\_FAT32) { |
|  | n\_fat = ((n\_clst \* 4) + 8 + SS(fs) - 1) / SS(fs); |
|  | n\_rsv = 32; |
|  | n\_dir = 0; |
|  | } else { |
|  | n\_fat = (fmt == FS\_FAT12) ? (n\_clst \* 3 + 1) / 2 + 3 : (n\_clst \* 2) + 4; |
|  | n\_fat = (n\_fat + SS(fs) - 1) / SS(fs); |
|  | n\_rsv = 1; |
|  | n\_dir = (DWORD)N\_ROOTDIR \* SZ\_DIRE / SS(fs); |
|  | } |
|  | b\_fat = b\_vol + n\_rsv; /\* FAT area start sector \*/ |
|  | b\_dir = b\_fat + n\_fat \* N\_FATS; /\* Directory area start sector \*/ |
|  | b\_data = b\_dir + n\_dir; /\* Data area start sector \*/ |
|  | if (n\_vol < b\_data + au - b\_vol) return FR\_MKFS\_ABORTED; /\* Too small volume \*/ |
|  |  |
|  | /\* Align data start sector to erase block boundary (for flash memory media) \*/ |
|  | if (disk\_ioctl(pdrv, GET\_BLOCK\_SIZE, &n) != RES\_OK || !n || n > 32768) n = 1; |
|  | n = (b\_data + n - 1) & ~(n - 1); /\* Next nearest erase block from current data start \*/ |
|  | n = (n - b\_data) / N\_FATS; |
|  | if (fmt == FS\_FAT32) { /\* FAT32: Move FAT offset \*/ |
|  | n\_rsv += n; |
|  | b\_fat += n; |
|  | } else { /\* FAT12/16: Expand FAT size \*/ |
|  | n\_fat += n; |
|  | } |
|  |  |
|  | /\* Determine number of clusters and final check of validity of the FAT sub-type \*/ |
|  | n\_clst = (n\_vol - n\_rsv - n\_fat \* N\_FATS - n\_dir) / au; |
|  | if ( (fmt == FS\_FAT16 && n\_clst < MIN\_FAT16) |
|  | || (fmt == FS\_FAT32 && n\_clst < MIN\_FAT32)) |
|  | return FR\_MKFS\_ABORTED; |
|  |  |
|  | /\* Determine system ID in the partition table \*/ |
|  | if (fmt == FS\_FAT32) { |
|  | sys = 0x0C; /\* FAT32X \*/ |
|  | } else { |
|  | if (fmt == FS\_FAT12 && n\_vol < 0x10000) { |
|  | sys = 0x01; /\* FAT12(<65536) \*/ |
|  | } else { |
|  | sys = (n\_vol < 0x10000) ? 0x04 : 0x06; /\* FAT16(<65536) : FAT12/16(>=65536) \*/ |
|  | } |
|  | } |
|  |  |
|  | if (\_MULTI\_PARTITION && part) { |
|  | /\* Update system ID in the partition table \*/ |
|  | tbl = &fs->win[MBR\_Table + (part - 1) \* SZ\_PTE]; |
|  | tbl[4] = sys; |
|  | if (disk\_write(pdrv, fs->win, 0, 1) != RES\_OK) /\* Write it to teh MBR \*/ |
|  | return FR\_DISK\_ERR; |
|  | md = 0xF8; |
|  | } else { |
|  | if (sfd) { /\* No partition table (SFD) \*/ |
|  | md = 0xF0; |
|  | } else { /\* Create partition table (FDISK) \*/ |
|  | mem\_set(fs->win, 0, SS(fs)); |
|  | tbl = fs->win + MBR\_Table; /\* Create partition table for single partition in the drive \*/ |
|  | tbl[1] = 1; /\* Partition start head \*/ |
|  | tbl[2] = 1; /\* Partition start sector \*/ |
|  | tbl[3] = 0; /\* Partition start cylinder \*/ |
|  | tbl[4] = sys; /\* System type \*/ |
|  | tbl[5] = 254; /\* Partition end head \*/ |
|  | n = (b\_vol + n\_vol) / 63 / 255; |
|  | tbl[6] = (BYTE)(n >> 2 | 63); /\* Partition end sector \*/ |
|  | tbl[7] = (BYTE)n; /\* End cylinder \*/ |
|  | ST\_DWORD(tbl + 8, 63); /\* Partition start in LBA \*/ |
|  | ST\_DWORD(tbl + 12, n\_vol); /\* Partition size in LBA \*/ |
|  | ST\_WORD(fs->win + BS\_55AA, 0xAA55); /\* MBR signature \*/ |
|  | if (disk\_write(pdrv, fs->win, 0, 1) != RES\_OK) /\* Write it to the MBR \*/ |
|  | return FR\_DISK\_ERR; |
|  | md = 0xF8; |
|  | } |
|  | } |
|  |  |
|  | /\* Create BPB in the VBR \*/ |
|  | tbl = fs->win; /\* Clear sector \*/ |
|  | mem\_set(tbl, 0, SS(fs)); |
|  | mem\_cpy(tbl, "\xEB\xFE\x90" "MSDOS5.0", 11);/\* Boot jump code, OEM name \*/ |
|  | i = SS(fs); /\* Sector size \*/ |
|  | ST\_WORD(tbl + BPB\_BytsPerSec, i); |
|  | tbl[BPB\_SecPerClus] = (BYTE)au; /\* Sectors per cluster \*/ |
|  | ST\_WORD(tbl + BPB\_RsvdSecCnt, n\_rsv); /\* Reserved sectors \*/ |
|  | tbl[BPB\_NumFATs] = N\_FATS; /\* Number of FATs \*/ |
|  | i = (fmt == FS\_FAT32) ? 0 : N\_ROOTDIR; /\* Number of root directory entries \*/ |
|  | ST\_WORD(tbl + BPB\_RootEntCnt, i); |
|  | if (n\_vol < 0x10000) { /\* Number of total sectors \*/ |
|  | ST\_WORD(tbl + BPB\_TotSec16, n\_vol); |
|  | } else { |
|  | ST\_DWORD(tbl + BPB\_TotSec32, n\_vol); |
|  | } |
|  | tbl[BPB\_Media] = md; /\* Media descriptor \*/ |
|  | ST\_WORD(tbl + BPB\_SecPerTrk, 63); /\* Number of sectors per track \*/ |
|  | ST\_WORD(tbl + BPB\_NumHeads, 255); /\* Number of heads \*/ |
|  | ST\_DWORD(tbl + BPB\_HiddSec, b\_vol); /\* Hidden sectors \*/ |
|  | n = GET\_FATTIME(); /\* Use current time as VSN \*/ |
|  | if (fmt == FS\_FAT32) { |
|  | ST\_DWORD(tbl + BS\_VolID32, n); /\* VSN \*/ |
|  | ST\_DWORD(tbl + BPB\_FATSz32, n\_fat); /\* Number of sectors per FAT \*/ |
|  | ST\_DWORD(tbl + BPB\_RootClus, 2); /\* Root directory start cluster (2) \*/ |
|  | ST\_WORD(tbl + BPB\_FSInfo, 1); /\* FSINFO record offset (VBR + 1) \*/ |
|  | ST\_WORD(tbl + BPB\_BkBootSec, 6); /\* Backup boot record offset (VBR + 6) \*/ |
|  | tbl[BS\_DrvNum32] = 0x80; /\* Drive number \*/ |
|  | tbl[BS\_BootSig32] = 0x29; /\* Extended boot signature \*/ |
|  | mem\_cpy(tbl + BS\_VolLab32, "NO NAME " "FAT32 ", 19); /\* Volume label, FAT signature \*/ |
|  | } else { |
|  | ST\_DWORD(tbl + BS\_VolID, n); /\* VSN \*/ |
|  | ST\_WORD(tbl + BPB\_FATSz16, n\_fat); /\* Number of sectors per FAT \*/ |
|  | tbl[BS\_DrvNum] = 0x80; /\* Drive number \*/ |
|  | tbl[BS\_BootSig] = 0x29; /\* Extended boot signature \*/ |
|  | mem\_cpy(tbl + BS\_VolLab, "NO NAME " "FAT ", 19); /\* Volume label, FAT signature \*/ |
|  | } |
|  | ST\_WORD(tbl + BS\_55AA, 0xAA55); /\* Signature (Offset is fixed here regardless of sector size) \*/ |
|  | if (disk\_write(pdrv, tbl, b\_vol, 1) != RES\_OK) /\* Write it to the VBR sector \*/ |
|  | return FR\_DISK\_ERR; |
|  | if (fmt == FS\_FAT32) /\* Write backup VBR if needed (VBR + 6) \*/ |
|  | disk\_write(pdrv, tbl, b\_vol + 6, 1); |
|  |  |
|  | /\* Initialize FAT area \*/ |
|  | wsect = b\_fat; |
|  | for (i = 0; i < N\_FATS; i++) { /\* Initialize each FAT copy \*/ |
|  | mem\_set(tbl, 0, SS(fs)); /\* 1st sector of the FAT \*/ |
|  | n = md; /\* Media descriptor byte \*/ |
|  | if (fmt != FS\_FAT32) { |
|  | n |= (fmt == FS\_FAT12) ? 0x00FFFF00 : 0xFFFFFF00; |
|  | ST\_DWORD(tbl + 0, n); /\* Reserve cluster #0-1 (FAT12/16) \*/ |
|  | } else { |
|  | n |= 0xFFFFFF00; |
|  | ST\_DWORD(tbl + 0, n); /\* Reserve cluster #0-1 (FAT32) \*/ |
|  | ST\_DWORD(tbl + 4, 0xFFFFFFFF); |
|  | ST\_DWORD(tbl + 8, 0x0FFFFFFF); /\* Reserve cluster #2 for root directory \*/ |
|  | } |
|  | if (disk\_write(pdrv, tbl, wsect++, 1) != RES\_OK) |
|  | return FR\_DISK\_ERR; |
|  | mem\_set(tbl, 0, SS(fs)); /\* Fill following FAT entries with zero \*/ |
|  | for (n = 1; n < n\_fat; n++) { /\* This loop may take a time on FAT32 volume due to many single sector writes \*/ |
|  | if (disk\_write(pdrv, tbl, wsect++, 1) != RES\_OK) |
|  | return FR\_DISK\_ERR; |
|  | } |
|  | } |
|  |  |
|  | /\* Initialize root directory \*/ |
|  | i = (fmt == FS\_FAT32) ? au : (UINT)n\_dir; |
|  | do { |
|  | if (disk\_write(pdrv, tbl, wsect++, 1) != RES\_OK) |
|  | return FR\_DISK\_ERR; |
|  | } while (--i); |
|  |  |
|  | #if \_USE\_TRIM /\* Erase data area if needed \*/ |
|  | { |
|  | eb[0] = wsect; eb[1] = wsect + (n\_clst - ((fmt == FS\_FAT32) ? 1 : 0)) \* au - 1; |
|  | disk\_ioctl(pdrv, CTRL\_TRIM, eb); |
|  | } |
|  | #endif |
|  |  |
|  | /\* Create FSINFO if needed \*/ |
|  | if (fmt == FS\_FAT32) { |
|  | ST\_DWORD(tbl + FSI\_LeadSig, 0x41615252); |
|  | ST\_DWORD(tbl + FSI\_StrucSig, 0x61417272); |
|  | ST\_DWORD(tbl + FSI\_Free\_Count, n\_clst - 1); /\* Number of free clusters \*/ |
|  | ST\_DWORD(tbl + FSI\_Nxt\_Free, 2); /\* Last allocated cluster# \*/ |
|  | ST\_WORD(tbl + BS\_55AA, 0xAA55); |
|  | disk\_write(pdrv, tbl, b\_vol + 1, 1); /\* Write original (VBR + 1) \*/ |
|  | disk\_write(pdrv, tbl, b\_vol + 7, 1); /\* Write backup (VBR + 7) \*/ |
|  | } |
|  |  |
|  | return (disk\_ioctl(pdrv, CTRL\_SYNC, 0) == RES\_OK) ? FR\_OK : FR\_DISK\_ERR; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | #if \_MULTI\_PARTITION |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Create partition table on the physical drive \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | FRESULT f\_fdisk ( |
|  | BYTE pdrv, /\* Physical drive number \*/ |
|  | const DWORD szt[], /\* Pointer to the size table for each partitions \*/ |
|  | void\* work /\* Pointer to the working buffer \*/ |
|  | ) |
|  | { |
|  | UINT i, n, sz\_cyl, tot\_cyl, b\_cyl, e\_cyl, p\_cyl; |
|  | BYTE s\_hd, e\_hd, \*p, \*buf = (BYTE\*)work; |
|  | DSTATUS stat; |
|  | DWORD sz\_disk, sz\_part, s\_part; |
|  |  |
|  |  |
|  | stat = disk\_initialize(pdrv); |
|  | if (stat & STA\_NOINIT) return FR\_NOT\_READY; |
|  | if (stat & STA\_PROTECT) return FR\_WRITE\_PROTECTED; |
|  | if (disk\_ioctl(pdrv, GET\_SECTOR\_COUNT, &sz\_disk)) return FR\_DISK\_ERR; |
|  |  |
|  | /\* Determine CHS in the table regardless of the drive geometry \*/ |
|  | for (n = 16; n < 256 && sz\_disk / n / 63 > 1024; n \*= 2) ; |
|  | if (n == 256) n--; |
|  | e\_hd = n - 1; |
|  | sz\_cyl = 63 \* n; |
|  | tot\_cyl = sz\_disk / sz\_cyl; |
|  |  |
|  | /\* Create partition table \*/ |
|  | mem\_set(buf, 0, \_MAX\_SS); |
|  | p = buf + MBR\_Table; b\_cyl = 0; |
|  | for (i = 0; i < 4; i++, p += SZ\_PTE) { |
|  | p\_cyl = (szt[i] <= 100U) ? (DWORD)tot\_cyl \* szt[i] / 100 : szt[i] / sz\_cyl; |
|  | if (!p\_cyl) continue; |
|  | s\_part = (DWORD)sz\_cyl \* b\_cyl; |
|  | sz\_part = (DWORD)sz\_cyl \* p\_cyl; |
|  | if (i == 0) { /\* Exclude first track of cylinder 0 \*/ |
|  | s\_hd = 1; |
|  | s\_part += 63; sz\_part -= 63; |
|  | } else { |
|  | s\_hd = 0; |
|  | } |
|  | e\_cyl = b\_cyl + p\_cyl - 1; |
|  | if (e\_cyl >= tot\_cyl) return FR\_INVALID\_PARAMETER; |
|  |  |
|  | /\* Set partition table \*/ |
|  | p[1] = s\_hd; /\* Start head \*/ |
|  | p[2] = (BYTE)((b\_cyl >> 2) + 1); /\* Start sector \*/ |
|  | p[3] = (BYTE)b\_cyl; /\* Start cylinder \*/ |
|  | p[4] = 0x06; /\* System type (temporary setting) \*/ |
|  | p[5] = e\_hd; /\* End head \*/ |
|  | p[6] = (BYTE)((e\_cyl >> 2) + 63); /\* End sector \*/ |
|  | p[7] = (BYTE)e\_cyl; /\* End cylinder \*/ |
|  | ST\_DWORD(p + 8, s\_part); /\* Start sector in LBA \*/ |
|  | ST\_DWORD(p + 12, sz\_part); /\* Partition size \*/ |
|  |  |
|  | /\* Next partition \*/ |
|  | b\_cyl += p\_cyl; |
|  | } |
|  | ST\_WORD(p, 0xAA55); |
|  |  |
|  | /\* Write it to the MBR \*/ |
|  | return (disk\_write(pdrv, buf, 0, 1) != RES\_OK || disk\_ioctl(pdrv, CTRL\_SYNC, 0) != RES\_OK) ? FR\_DISK\_ERR : FR\_OK; |
|  | } |
|  |  |
|  |  |
|  | #endif /\* \_MULTI\_PARTITION \*/ |
|  | #endif /\* \_USE\_MKFS && !\_FS\_READONLY \*/ |
|  |  |
|  |  |
|  |  |
|  |  |
|  | #if \_USE\_STRFUNC |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Get a string from the file \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | TCHAR\* f\_gets ( |
|  | TCHAR\* buff, /\* Pointer to the string buffer to read \*/ |
|  | int len, /\* Size of string buffer (characters) \*/ |
|  | FIL\* fp /\* Pointer to the file object \*/ |
|  | ) |
|  | { |
|  | int n = 0; |
|  | TCHAR c, \*p = buff; |
|  | BYTE s[2]; |
|  | UINT rc; |
|  |  |
|  |  |
|  | while (n < len - 1) { /\* Read characters until buffer gets filled \*/ |
|  | #if \_USE\_LFN && \_LFN\_UNICODE |
|  | #if \_STRF\_ENCODE == 3 /\* Read a character in UTF-8 \*/ |
|  | f\_read(fp, s, 1, &rc); |
|  | if (rc != 1) break; |
|  | c = s[0]; |
|  | if (c >= 0x80) { |
|  | if (c < 0xC0) continue; /\* Skip stray trailer \*/ |
|  | if (c < 0xE0) { /\* Two-byte sequence \*/ |
|  | f\_read(fp, s, 1, &rc); |
|  | if (rc != 1) break; |
|  | c = (c & 0x1F) << 6 | (s[0] & 0x3F); |
|  | if (c < 0x80) c = '?'; |
|  | } else { |
|  | if (c < 0xF0) { /\* Three-byte sequence \*/ |
|  | f\_read(fp, s, 2, &rc); |
|  | if (rc != 2) break; |
|  | c = c << 12 | (s[0] & 0x3F) << 6 | (s[1] & 0x3F); |
|  | if (c < 0x800) c = '?'; |
|  | } else { /\* Reject four-byte sequence \*/ |
|  | c = '?'; |
|  | } |
|  | } |
|  | } |
|  | #elif \_STRF\_ENCODE == 2 /\* Read a character in UTF-16BE \*/ |
|  | f\_read(fp, s, 2, &rc); |
|  | if (rc != 2) break; |
|  | c = s[1] + (s[0] << 8); |
|  | #elif \_STRF\_ENCODE == 1 /\* Read a character in UTF-16LE \*/ |
|  | f\_read(fp, s, 2, &rc); |
|  | if (rc != 2) break; |
|  | c = s[0] + (s[1] << 8); |
|  | #else /\* Read a character in ANSI/OEM \*/ |
|  | f\_read(fp, s, 1, &rc); |
|  | if (rc != 1) break; |
|  | c = s[0]; |
|  | if (IsDBCS1(c)) { |
|  | f\_read(fp, s, 1, &rc); |
|  | if (rc != 1) break; |
|  | c = (c << 8) + s[0]; |
|  | } |
|  | c = ff\_convert(c, 1); /\* OEM -> Unicode \*/ |
|  | if (!c) c = '?'; |
|  | #endif |
|  | #else /\* Read a character without conversion \*/ |
|  | f\_read(fp, s, 1, &rc); |
|  | if (rc != 1) break; |
|  | c = s[0]; |
|  | #endif |
|  | if (\_USE\_STRFUNC == 2 && c == '\r') continue; /\* Strip '\r' \*/ |
|  | \*p++ = c; |
|  | n++; |
|  | if (c == '\n') break; /\* Break on EOL \*/ |
|  | } |
|  | \*p = 0; |
|  | return n ? buff : 0; /\* When no data read (eof or error), return with error. \*/ |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | #if !\_FS\_READONLY |
|  | #include <stdarg.h> |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Put a character to the file \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | typedef struct { |
|  | FIL\* fp; |
|  | int idx, nchr; |
|  | BYTE buf[64]; |
|  | } putbuff; |
|  |  |
|  |  |
|  | static |
|  | void putc\_bfd ( |
|  | putbuff\* pb, |
|  | TCHAR c |
|  | ) |
|  | { |
|  | UINT bw; |
|  | int i; |
|  |  |
|  |  |
|  | if (\_USE\_STRFUNC == 2 && c == '\n') /\* LF -> CRLF conversion \*/ |
|  | putc\_bfd(pb, '\r'); |
|  |  |
|  | i = pb->idx; /\* Buffer write index (-1:error) \*/ |
|  | if (i < 0) return; |
|  |  |
|  | #if \_USE\_LFN && \_LFN\_UNICODE |
|  | #if \_STRF\_ENCODE == 3 /\* Write a character in UTF-8 \*/ |
|  | if (c < 0x80) { /\* 7-bit \*/ |
|  | pb->buf[i++] = (BYTE)c; |
|  | } else { |
|  | if (c < 0x800) { /\* 11-bit \*/ |
|  | pb->buf[i++] = (BYTE)(0xC0 | c >> 6); |
|  | } else { /\* 16-bit \*/ |
|  | pb->buf[i++] = (BYTE)(0xE0 | c >> 12); |
|  | pb->buf[i++] = (BYTE)(0x80 | (c >> 6 & 0x3F)); |
|  | } |
|  | pb->buf[i++] = (BYTE)(0x80 | (c & 0x3F)); |
|  | } |
|  | #elif \_STRF\_ENCODE == 2 /\* Write a character in UTF-16BE \*/ |
|  | pb->buf[i++] = (BYTE)(c >> 8); |
|  | pb->buf[i++] = (BYTE)c; |
|  | #elif \_STRF\_ENCODE == 1 /\* Write a character in UTF-16LE \*/ |
|  | pb->buf[i++] = (BYTE)c; |
|  | pb->buf[i++] = (BYTE)(c >> 8); |
|  | #else /\* Write a character in ANSI/OEM \*/ |
|  | c = ff\_convert(c, 0); /\* Unicode -> OEM \*/ |
|  | if (!c) c = '?'; |
|  | if (c >= 0x100) |
|  | pb->buf[i++] = (BYTE)(c >> 8); |
|  | pb->buf[i++] = (BYTE)c; |
|  | #endif |
|  | #else /\* Write a character without conversion \*/ |
|  | pb->buf[i++] = (BYTE)c; |
|  | #endif |
|  |  |
|  | if (i >= (int)(sizeof pb->buf) - 3) { /\* Write buffered characters to the file \*/ |
|  | f\_write(pb->fp, pb->buf, (UINT)i, &bw); |
|  | i = (bw == (UINT)i) ? 0 : -1; |
|  | } |
|  | pb->idx = i; |
|  | pb->nchr++; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | int f\_putc ( |
|  | TCHAR c, /\* A character to be output \*/ |
|  | FIL\* fp /\* Pointer to the file object \*/ |
|  | ) |
|  | { |
|  | putbuff pb; |
|  | UINT nw; |
|  |  |
|  |  |
|  | pb.fp = fp; /\* Initialize output buffer \*/ |
|  | pb.nchr = pb.idx = 0; |
|  |  |
|  | putc\_bfd(&pb, c); /\* Put a character \*/ |
|  |  |
|  | if ( pb.idx >= 0 /\* Flush buffered characters to the file \*/ |
|  | && f\_write(pb.fp, pb.buf, (UINT)pb.idx, &nw) == FR\_OK |
|  | && (UINT)pb.idx == nw) return pb.nchr; |
|  | return EOF; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Put a string to the file \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | int f\_puts ( |
|  | const TCHAR\* str, /\* Pointer to the string to be output \*/ |
|  | FIL\* fp /\* Pointer to the file object \*/ |
|  | ) |
|  | { |
|  | putbuff pb; |
|  | UINT nw; |
|  |  |
|  |  |
|  | pb.fp = fp; /\* Initialize output buffer \*/ |
|  | pb.nchr = pb.idx = 0; |
|  |  |
|  | while (\*str) /\* Put the string \*/ |
|  | putc\_bfd(&pb, \*str++); |
|  |  |
|  | if ( pb.idx >= 0 /\* Flush buffered characters to the file \*/ |
|  | && f\_write(pb.fp, pb.buf, (UINT)pb.idx, &nw) == FR\_OK |
|  | && (UINT)pb.idx == nw) return pb.nchr; |
|  | return EOF; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\*-----------------------------------------------------------------------\*/ |
|  | /\* Put a formatted string to the file \*/ |
|  | /\*-----------------------------------------------------------------------\*/ |
|  |  |
|  | int f\_printf ( |
|  | FIL\* fp, /\* Pointer to the file object \*/ |
|  | const TCHAR\* fmt, /\* Pointer to the format string \*/ |
|  | ... /\* Optional arguments... \*/ |
|  | ) |
|  | { |
|  | va\_list arp; |
|  | BYTE f, r; |
|  | UINT nw, i, j, w; |
|  | DWORD v; |
|  | TCHAR c, d, s[16], \*p; |
|  | putbuff pb; |
|  |  |
|  |  |
|  | pb.fp = fp; /\* Initialize output buffer \*/ |
|  | pb.nchr = pb.idx = 0; |
|  |  |
|  | va\_start(arp, fmt); |
|  |  |
|  | for (;;) { |
|  | c = \*fmt++; |
|  | if (c == 0) break; /\* End of string \*/ |
|  | if (c != '%') { /\* Non escape character \*/ |
|  | putc\_bfd(&pb, c); |
|  | continue; |
|  | } |
|  | w = f = 0; |
|  | c = \*fmt++; |
|  | if (c == '0') { /\* Flag: '0' padding \*/ |
|  | f = 1; c = \*fmt++; |
|  | } else { |
|  | if (c == '-') { /\* Flag: left justified \*/ |
|  | f = 2; c = \*fmt++; |
|  | } |
|  | } |
|  | while (IsDigit(c)) { /\* Precision \*/ |
|  | w = w \* 10 + c - '0'; |
|  | c = \*fmt++; |
|  | } |
|  | if (c == 'l' || c == 'L') { /\* Prefix: Size is long int \*/ |
|  | f |= 4; c = \*fmt++; |
|  | } |
|  | if (!c) break; |
|  | d = c; |
|  | if (IsLower(d)) d -= 0x20; |
|  | switch (d) { /\* Type is... \*/ |
|  | case 'S' : /\* String \*/ |
|  | p = va\_arg(arp, TCHAR\*); |
|  | for (j = 0; p[j]; j++) ; |
|  | if (!(f & 2)) { |
|  | while (j++ < w) putc\_bfd(&pb, ' '); |
|  | } |
|  | while (\*p) putc\_bfd(&pb, \*p++); |
|  | while (j++ < w) putc\_bfd(&pb, ' '); |
|  | continue; |
|  | case 'C' : /\* Character \*/ |
|  | putc\_bfd(&pb, (TCHAR)va\_arg(arp, int)); continue; |
|  | case 'B' : /\* Binary \*/ |
|  | r = 2; break; |
|  | case 'O' : /\* Octal \*/ |
|  | r = 8; break; |
|  | case 'D' : /\* Signed decimal \*/ |
|  | case 'U' : /\* Unsigned decimal \*/ |
|  | r = 10; break; |
|  | case 'X' : /\* Hexdecimal \*/ |
|  | r = 16; break; |
|  | default: /\* Unknown type (pass-through) \*/ |
|  | putc\_bfd(&pb, c); continue; |
|  | } |
|  |  |
|  | /\* Get an argument and put it in numeral \*/ |
|  | v = (f & 4) ? (DWORD)va\_arg(arp, long) : ((d == 'D') ? (DWORD)(long)va\_arg(arp, int) : (DWORD)va\_arg(arp, unsigned int)); |
|  | if (d == 'D' && (v & 0x80000000)) { |
|  | v = 0 - v; |
|  | f |= 8; |
|  | } |
|  | i = 0; |
|  | do { |
|  | d = (TCHAR)(v % r); v /= r; |
|  | if (d > 9) d += (c == 'x') ? 0x27 : 0x07; |
|  | s[i++] = d + '0'; |
|  | } while (v && i < sizeof s / sizeof s[0]); |
|  | if (f & 8) s[i++] = '-'; |
|  | j = i; d = (f & 1) ? '0' : ' '; |
|  | while (!(f & 2) && j++ < w) putc\_bfd(&pb, d); |
|  | do putc\_bfd(&pb, s[--i]); while (i); |
|  | while (j++ < w) putc\_bfd(&pb, d); |
|  | } |
|  |  |
|  | va\_end(arp); |
|  |  |
|  | if ( pb.idx >= 0 /\* Flush buffered characters to the file \*/ |
|  | && f\_write(pb.fp, pb.buf, (UINT)pb.idx, &nw) == FR\_OK |
|  | && (UINT)pb.idx == nw) return pb.nchr; |
|  | return EOF; |
|  | } |
|  |  |
|  | #endif /\* !\_FS\_READONLY \*/ |
|  | #endif /\* \_USE\_STRFUNC \*/ |

* [Status](https://status.github.com/)

# Appendix 3

## Firmware algorithm:



**A**

**Source code** used in the prototype demo presentation (excluding SD Card and Bluetooth open source libraries, as well as the essential drivers for the GPIO and System initialization of the arm microcontroller):

Firmware structure:

1. Main.cpp
2. TLC5941 Driver (tlc5941.cpp/.h, tlcrgb.cpp/.h)
3. SD Card library
4. Bluetooth Library (hc06.cpp/.h)
5. Utility files (debug.h, port.h, cpufreq.h, sdcard\_inc.h)
6. ARM Libraries (GPIO, SPI, USART, System config and initialization files, etc…)

|  |
| --- |
| **// main.cpp**  **#include** <stdio.h>  **#include** "sdcard\_inc.h" /\* include sdcard library \*/  **#include** "tlc5941/tlcrgb.h" /\* include TLC5941 library (which includes rgb led handling) \*/  **#include** "cpufreq.h" /\* include cpu overclock \*/  **#include** "port.h" /\* include port / gpio handling header \*/  **#include** "debug.h" /\* include debug.h, which helps indicate when there's problems with the program \*/  **extern** "C" {  **#include** "hc06/hc06.h"  }  **class** Propeller {  **public**:  **#define** LED\_COUNT 3 /\* Will be 32 leds in the end \*/  **#define** HC06\_BAUD 115200  **#define** HALL\_PIN GPIO\_Pin\_0  **#define** HALL\_PORT GPIOD  **#define** HALL\_PORT\_INDEX 3  TLC5941 tlc;  TLCRGB leds[LED\_COUNT];  //TLCRGB::rgb\_t pix[1024]; /\* Small buffer for storing an animation frame. The size of the buffer hasn't been calculated yet \*/  FATFS fs;  FIL file;  **Propeller**() {  /\* Initialize everything: \*/  SystemInit(); /\* Normal init \*/  rcc\_set\_frequency(*SYSCLK\_84\_MHZ*); /\* Overclock CPU (i'll keep it at the original clock for now) \*/  HC06\_Init(HC06\_BAUD); /\* initialize early to allow debugging messages \*/  **#if** DEBUGMODE==1  HC06\_PutStr("Initializing Propeller Display...\n\r");  **#endif**  /\* Initialize Board's LEDs:  \* (RED - Error | Green - Success and program ended | Orange - Initialized successfully) \*/  init\_board\_debug();  init\_sdcard(); /\* Initialize SD Card \*/  hall\_init();  calibrate();  tlc.init(); /\* Initialize TLC5941 \*/  /\* Initialize LEDs: \*/  **for**(**int** i = 0; i < LED\_COUNT; i++) leds[i].init(i);  /\* Initialization success: \*/  SET\_BOARD\_LED(*BLUE*);  **#if** DEBUGMODE==1  HC06\_PutStr("Propeller Display online\n\r");  **#endif**  }  **void** **stop**() {  deinit\_sdcard();  **#if** DEBUGMODE==1  HC06\_PutStr("Propeller Display offline\n\r");  **#endif**  IDLE();  }  **void** **px**(**char** x, **char** y, uint16\_t r, uint16\_t g, uint16\_t b) {  /\* **TODO**: px should apply the color to a space buffer, not directly to the leds \*/  leds[(**int**)x].updateColor(r, g, b);  }  **void** **update**() {  /\* Update everything related to the propeller display here \*/  /\* **TODO**: For every update, a rotation probably happened. Because of this,  \* we'll change the leds according to a certain buffer \*/  tlc.*update*();  }  uint8\_t **hall\_get**(**void**) {  **return** PDBI(HALL\_PIN);  }  **private**:  uint8\_t is\_first\_spin;  **void** **calibrate**() {  **#if** DEBUGMODE==1  HC06\_PutStr("Calibrating...\n\r");  **#endif**  uint8\_t old\_hall = is\_first\_spin;  **while**(old\_hall == hall\_get()); /\* Wait until the hall effect sensor toggles \*/  is\_first\_spin = old\_hall; /\* store the toggled value \*/  }  **void** **deinit\_sdcard**(**void**) {  f\_mount(0, "", 1);  PA(0);  printf("SD Card unmounted\n\n");  }  **void** **init\_sdcard**(**void**) {  **if**(f\_mount(&fs, "", 1) != *FR\_OK*) ERROR("SD Card failed!");  printf("SD Card mounted\n\n");  }  **void** **hall\_init**(**void**) {  init\_gpio(HALL\_PORT, HALL\_PORT\_INDEX, 0, HALL\_PIN);  is\_first\_spin = hall\_get();  }  };  **int** **main**(**void**) {  Propeller pr;  pr.px(0, 0, 0xFF, 0, 0xFF); /\* LED RGB 0 is now magenta \*/  pr.update();  pr.stop();  } |