GCC Code Coverage Report

		Exec	Total	Coverage
Directory:	./			
Date:	2022-03-20 01:57:36	**Lines:**	0	169	0.0 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	0	88	0.0 %

File	Lines			Branches	
include/diskio_layer/protocol/diskio_protocol_spi.hpp		0.0 %	0 / 15	0.0 %	0 / 2
src/diskio_layer/implementation/diskio_mmc_spi.cpp		0.0 %	0 / 47	0.0 %	0 / 10
src/diskio_layer/implementation/diskio_usb.cpp		0.0 %	0 / 12	- %	0 / 0
src/ff_driver_common.cpp		0.0 %	0 / 95	0.0 %	0 / 76

GCC Code Coverage Report

		Exec	Total	Coverage
Directory:	./			
Date:	2022-03-20 01:57:36	**Lines:**	0	169	0.0 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	0	88	0.0 %

File	Lines			Branches	
include/diskio_layer/protocol/diskio_protocol_spi.hpp		0.0 %	0 / 15	0.0 %	0 / 2
src/diskio_layer/implementation/diskio_mmc_spi.cpp		0.0 %	0 / 47	0.0 %	0 / 10
src/diskio_layer/implementation/diskio_usb.cpp		0.0 %	0 / 12	- %	0 / 0
src/ff_driver_common.cpp		0.0 %	0 / 95	0.0 %	0 / 76

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	include/diskio_layer/protocol/diskio_protocol_spi.hpp	Lines:	0	15	0.0 %
Date:	2022-03-20 01:57:36	Branches:	0	2	0.0 %

```
Line Branch Exec  Source
   1                
   2                // MIT License
   3                
   4                // Copyright (c) 2022 Chris Sutton
   5                
   6                // Permission is hereby granted, free of charge, to any person obtaining a copy
   7                // of this software and associated documentation files (the "Software"), to deal
   8                // in the Software without restriction, including without limitation the rights
   9                // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  10                // copies of the Software, and to permit persons to whom the Software is
  11                // furnished to do so, subject to the following conditions:
  12                
  13                // The above copyright notice and this permission notice shall be included in all
  14                // copies or substantial portions of the Software.
  15                
  16                // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  17                // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  18                // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  19                // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  20                // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  21                // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  22                // SOFTWARE.
  23                
  24                #ifndef __DISKIO_PROTOCOL_SPI_HPP__
  25                #define __DISKIO_PROTOCOL_SPI_HPP__
  26                
  27                
  28                #if defined(X86_UNIT_TESTING_ONLY)
  29                 #include <iostream>
  30                    // This file should contain bit definitions
  31                    #include <mock_cmsis.hpp>
  32                #else
  33                 #pragma GCC diagnostic push
  34                 #pragma GCC diagnostic ignored "-Wvolatile"
  35                  #include <stm32g0xx_ll_gpio.h>
  36                        #include <stm32g0xx_ll_spi.h>
  37                        #include <stm32g0xx_ll_bus.h>
  38                 #pragma GCC diagnostic pop
  39                
  40                
  41                #endif
  42                
  43                #include <spi_utils.hpp>
  44                #include <stdint.h>
  45                #include <utility>
  46                
  47                namespace fatfs
  48                {
  49                
  50                #if defined(ENABLE_MMC_SPI)
  51                
  52                // @brief Class containing pointers to the STM32 peripheral,
  53                // GPIO ports and pins numbers for use with SPI protocol.
  54                class DiskioProtocolSPI
  55                {
  56                public:
  57                 /// @brief Construct a new Diskio object for the SPI Protocol
  58                 /// @param spi
  59                 /// @param cs_gpio
  60                 /// @param mosi_gpio
  61                 /// @param miso_gpio
  62                 /// @param sck_gpio
  63                 /// @param rcc_spi_clk
  64                 DiskioProtocolSPI(
  65                        SPI_TypeDef *spi,
  66                        std::pair<GPIO_TypeDef*, uint32_t>  sck_gpio,
  67                        std::pair<GPIO_TypeDef*, uint32_t>  mosi_gpio,
  68                        std::pair<GPIO_TypeDef*, uint32_t>  miso_gpio,
  69                        std::pair<GPIO_TypeDef*, uint32_t>  cs_gpio,
  70                        uint32_t rcc_spi_clk
  71                    )
  72                 :   m_spi(spi), m_sck_gpio(sck_gpio), m_mosi_gpio(mosi_gpio),
  73                  m_miso_gpio(miso_gpio), m_cs_gpio(cs_gpio), m_rcc_spi_clk(rcc_spi_clk)
  74                 {}
  75               bool setup_spi()
  76                 {
  77                
  78                
  79                        stm32::spi::enable_spi(m_spi, false);
  80                #ifndef X86_UNIT_TESTING_ONLY
  81                
  82                  #pragma GCC diagnostic push
  83                  #pragma GCC diagnostic ignored "-Wvolatile"
```

```
84
85              LL_IOP_GRP1_EnableClock(LL_IOP_GRP1_PERIPH_GPIOA | LL_IOP_GRP1_PERIPH_GPIOB | LL_IOP_GRP1_PERIPH_GPIOD);
86
87              // Enable GPIO (SPI_SCK)
88              LL_GPIO_SetPinSpeed(m_sck_gpio.first, m_sck_gpio.second, LL_GPIO_SPEED_FREQ_VERY_HIGH);
89              LL_GPIO_SetPinOutputType(m_sck_gpio.first, m_sck_gpio.second, LL_GPIO_OUTPUT_PUSHPULL);
90              LL_GPIO_SetPinPull(m_sck_gpio.first, m_sck_gpio.second, LL_GPIO_PULL_UP);
91              LL_GPIO_SetAFPin_8_15(m_sck_gpio.first, m_sck_gpio.second, LL_GPIO_AF_1);
92              LL_GPIO_SetPinMode(m_sck_gpio.first, m_sck_gpio.second, LL_GPIO_MODE_ALTERNATE);
93
94              // Enable GPIO (SPI_MOSI)
95              LL_GPIO_SetPinSpeed(m_mosi_gpio.first, m_mosi_gpio.second, GPIO_OSPEEDR_OSPEED0); // medium output speed
96              LL_GPIO_SetPinOutputType(m_mosi_gpio.first, m_mosi_gpio.second, LL_GPIO_OUTPUT_PUSHPULL);
97              LL_GPIO_SetPinPull(m_mosi_gpio.first, m_mosi_gpio.second, LL_GPIO_PULL_UP);
98              LL_GPIO_SetAFPin_0_7(m_mosi_gpio.first, m_mosi_gpio.second, LL_GPIO_AF_1);
99              LL_GPIO_SetPinMode(m_mosi_gpio.first, m_mosi_gpio.second, LL_GPIO_MODE_ALTERNATE);
100
101             // Enable GPIO (SPI_MISO)
102             LL_GPIO_SetPinSpeed(m_miso_gpio.first, m_miso_gpio.second, GPIO_OSPEEDR_OSPEED0); // medium output speed
103             LL_GPIO_SetPinOutputType(m_miso_gpio.first, m_miso_gpio.second, LL_GPIO_OUTPUT_PUSHPULL);
104             LL_GPIO_SetPinPull(m_miso_gpio.first, m_miso_gpio.second, LL_GPIO_PULL_UP); // must be PUP for init
105             LL_GPIO_SetAFPin_0_7(m_miso_gpio.first, m_miso_gpio.second, LL_GPIO_AF_1);
106             LL_GPIO_SetPinMode(m_miso_gpio.first, m_miso_gpio.second, LL_GPIO_MODE_ALTERNATE);
107
108             // Enable GPIO (CS)
109             LL_GPIO_SetPinSpeed(m_cs_gpio.first, m_cs_gpio.second, LL_GPIO_SPEED_FREQ_VERY_HIGH); // medium output speed
110             LL_GPIO_SetPinOutputType(m_cs_gpio.first, m_cs_gpio.second, LL_GPIO_OUTPUT_PUSHPULL);
111             LL_GPIO_SetPinPull(m_cs_gpio.first, m_cs_gpio.second, LL_GPIO_PULL_UP);
112             LL_GPIO_SetPinMode(m_cs_gpio.first, m_cs_gpio.second, LL_GPIO_MODE_OUTPUT);
113             #pragma GCC diagnostic pop  // ignored "-Wvolatile"
114         #endif // X86_UNIT_TESTING_ONLY
115
116
117
118             RCC->APBENR1 = RCC->APBENR1 | m_rcc_spi_clk;
119             //m_spi->CR1 |= ((SPI_CR1_BIDIMODE | SPI_CR1_BIDIOE) | (SPI_CR1_MSTR | SPI_CR1_SSI) | SPI_CR1_SSM | SPI_CR1_BR_1);
120
121             // Enable software NSS management
122             m_spi->CR1 = m_spi->CR1 | SPI_CR1_SSI;
123             m_spi->CR1 = m_spi->CR1 | SPI_CR1_SSM;
124             m_spi->CR1 = m_spi->CR1 & ~ SPI_CR2_NSSP;
125
126             // Set the default prescaler to 8. e.g. set the baudrate
127             m_spi->CR1 = m_spi->CR1 | (SPI_CR1_BR_1);
128
129             // Enable SPI Master mode
130             m_spi->CR1 = m_spi->CR1 | SPI_CR1_MSTR;
131
132             stm32::spi::enable_spi(m_spi);
133
134             // check for Mode Faults in the configuration
135             if (m_spi->SR & SPI_SR_MODF)
136             {
137                 return false;
138             }
139             return true;
140          }
141
142          SPI_TypeDef * spi_handle() { return m_spi; }
143          std::pair<GPIO_TypeDef*, uint32_t> cs_gpio() { return m_cs_gpio; }
144          std::pair<GPIO_TypeDef*, uint32_t> mosi_gpio() { return m_mosi_gpio; }
145          std::pair<GPIO_TypeDef*, uint32_t> miso_gpio() { return m_miso_gpio; }

146          std::pair<GPIO_TypeDef*, uint32_t> sck_gpio() { return m_sck_gpio; }
147             uint32_t rcc_spi_clk() { return m_rcc_spi_clk; }
148
149             void set_cs_low() { m_cs_gpio.first->BRR = m_cs_gpio.second; }
150             void set_cs_high() { m_cs_gpio.first->BSRR = m_cs_gpio.second; }
151             void toggle_cs()
152             {
153                 // read the ODR state of this GPIO port
154                 uint32_t odr_reg = m_cs_gpio.first->ODR;
155                 // reset/set the BSRR using the ODR and the pin number (second)
156                 m_cs_gpio.first->BSRR = m_cs_gpio.first->BSRR
157                     | (((odr_reg & m_cs_gpio.second) << 16U) | (~odr_reg & m_cs_gpio.second));
158             }
159
160         private:
161          // @brief The SPI peripheral
162          SPI_TypeDef *m_spi;
163          std::pair<GPIO_TypeDef*, uint16_t>  m_sck_gpio;
164          std::pair<GPIO_TypeDef*, uint16_t>  m_mosi_gpio;
165          std::pair<GPIO_TypeDef*, uint16_t>  m_miso_gpio;
166          std::pair<GPIO_TypeDef*, uint16_t>  m_cs_gpio;
167             // @brief Used to enable the SPI clock for CS, MOSI, MISO and SCK pins
168             uint32_t m_rcc_spi_clk;
169         };
170
171         #endif // #if defined(ENABLE_MMC_SPI)
172
173         } // namespace fatfs
174
175         #endif // __DISKIO_PROTOCOL_SPI_HPP__
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	src/diskio_layer/implementation/diskio_mmc_spi.cpp	Lines:	0	47	0.0 %
Date:	2022-03-20 01:57:36	Branches:	0	10	0.0 %

```
Line Branch Exec  Source
   1                // C++ port of the original source code is subject to MIT License
   2
   3                // Copyright (c) 2022 Chris Sutton
   4
   5                // Permission is hereby granted, free of charge, to any person obtaining a copy
   6                // of this software and associated documentation files (the "Software"), to deal
   7                // in the Software without restriction, including without limitation the rights
   8                // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
   9                // copies of the Software, and to permit persons to whom the Software is
  10                // furnished to do so, subject to the following conditions:
  11
  12                // The above copyright notice and this permission notice shall be included in all
  13                // copies or substantial portions of the Software.
  14
  15                // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  16                // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  17                // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  18                // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  19                // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  20                // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  21                // SOFTWARE.
  22
  23                #include <diskio_hardware_mmc.hpp>
  24
  25                #include <timer_manager.hpp>
  26
  27                #ifdef X86_UNIT_TESTING_ONLY
  28                #else
  29                    #pragma GCC diagnostic push
  30                    #pragma GCC diagnostic ignored "-Wvolatile"
  31                    #include <stm32g0xx_ll_gpio.h>
  32                    #pragma GCC diagnostic pop  // ignored "-Wvolatile"
  33                #endif
  34                #if defined(USE_RTT)
  35                #include <SEGGER_RTT.h>
  36                #endif
  37
  38
  39
  40                namespace fatfs {
  41
  42                #if defined(ENABLE_MMC_SPI)
  43
  44
  45                template<>
  46    ######      DiskioHardwareBase::DSTATUS DiskioHardwareMMC<DiskioProtocolSPI>::initialize(BYTE pdrv [[maybe_unused]])
  47                {
  48                    // get STM32 SPI peripheral ready
  49    ######          if (!m_protocol.setup_spi())
  50                    {
  51                        // SPI config error!
  52    ######              while(true)
  53                        {}
  54                    }
  55
  56    ######          DSTATUS res = 0;
  57
  58                 // BYTE n, cmd, ty, ocr[4];
  59
  60                    // drive #0 only
  61    ######       if (pdrv != 0) return STA_NOINIT;
  62
  63                    // Is card inserted?
  64    ######       if (Stat & STA_NODISK) return Stat;
  65
  66                    // 1. Power On
  67                    // send some bytes at slow speed to trigger 74+ clock pulses via SCLK [MCLK 64MHz / SPI-PSC 256 = SPI-CLK 250KHz]
  68    ######       stm32::spi::set_prescaler(m_protocol.spi_handle(), (SPI_CR1_BR_2 | SPI_CR1_BR_1 | SPI_CR1_BR_0));
  69
  70    ######          m_protocol.set_cs_high();
  71    ######          for (uint8_t n = 20; n; n--)
  72                    {
  73                        // send a byte, value is not important
  74    ######              stm32::spi::send_byte(m_protocol.spi_handle(), 0xFF);
  75                    }
  76
  77                    // 2. Software Reset the MMC into SPI Mode
  78    ######          m_protocol.set_cs_low();
  79    ######          stm32::spi::send_byte(m_protocol.spi_handle(), (0x40 | CMD0));
  80    ######          stm32::spi::send_byte(m_protocol.spi_handle(), (0x00));
```

```
81          stm32::spi::send_byte(m_protocol.spi_handle(), (0x00));
82          stm32::spi::send_byte(m_protocol.spi_handle(), (0x00));
83          stm32::spi::send_byte(m_protocol.spi_handle(), (0x00));
84          stm32::spi::send_byte(m_protocol.spi_handle(), (0x95)); // CRC for CMD0
85          stm32::delay_millisecond(1);
86
87          [[maybe_unused]] volatile uint8_t rxbyte {0};
88          while (rxbyte != R1)
89          {
90              stm32::spi::send_byte(m_protocol.spi_handle(), 0xFF);
91              rxbyte = m_protocol.spi_handle()->DR;
92              stm32::delay_millisecond(1);
93          }
94
95          // 2. Initialization
96          // send some bytes at fast speed [MCLK 64MHz / SPI-PSC 8 = SPI-CLK 8MHz]
97       // stm32::spi::set_prescaler(m_protocol.spi_handle(), SPI_CR1_BR_1);
98          stm32::spi::send_byte(m_protocol.spi_handle(), ACMD41);
99          stm32::spi::send_byte(m_protocol.spi_handle(), (0x40));
100         stm32::spi::send_byte(m_protocol.spi_handle(), (0x00));
101         stm32::spi::send_byte(m_protocol.spi_handle(), (0x00));
102         stm32::spi::send_byte(m_protocol.spi_handle(), (0x00));
103         stm32::spi::send_byte(m_protocol.spi_handle(), (0x77));  // crc for CMD8
104
105         stm32::delay_millisecond(1);
106         // while (rxbyte != R0)
107         {
108             stm32::spi::send_byte(m_protocol.spi_handle(), 0xFF);
109             // cppcheck-suppress unreadVariable
110             rxbyte = m_protocol.spi_handle()->DR;
111             #if defined(USE_RTT)
112             if (rxbyte != 0xFF)
113                 SEGGER_RTT_printf(0, "\nrx: %u", +rxbyte);
114             #endif
115             stm32::delay_millisecond(1);
116         }
117
118         return res;
119     }
120
121     template<>
122     DiskioHardwareBase::DSTATUS DiskioHardwareMMC<DiskioProtocolSPI>::status(BYTE pdrv [[maybe_unused]])
123     {
124         // get STM32 SPI peripheral ready
125         m_protocol.setup_spi();
126         DSTATUS res = 0;
127         return res;
128     }
129
130     template<>
131     DiskioHardwareBase::DRESULT DiskioHardwareMMC<DiskioProtocolSPI>::read(
132         BYTE pdrv [[maybe_unused]],
133         BYTE* buff [[maybe_unused]],
134         LBA_t sector [[maybe_unused]],
135         UINT count [[maybe_unused]])
136     {
137         // get STM32 SPI peripheral ready
138         m_protocol.setup_spi();
139         return DRESULT::RES_OK;
140     }
141
142     template<>
143     DiskioHardwareBase::DRESULT DiskioHardwareMMC<DiskioProtocolSPI>::write(
144         BYTE pdrv [[maybe_unused]],
145         const BYTE* buff [[maybe_unused]],
146         LBA_t sector [[maybe_unused]],
147         UINT count [[maybe_unused]])
148     {
149         // get STM32 SPI peripheral ready
150         m_protocol.setup_spi();
151         return DRESULT::RES_OK;
152     }
153
154     template<>
155     DiskioHardwareBase::DRESULT DiskioHardwareMMC<DiskioProtocolSPI>::ioctl (
156         BYTE pdrv [[maybe_unused]],
157         BYTE cmd [[maybe_unused]],
158         void *buff [[maybe_unused]])
159     {
160         // get STM32 SPI peripheral ready
161         m_protocol.setup_spi();
162         return DRESULT::RES_OK;
163     }
164
165     #endif // #if defined(ENABLE_MMC_SPI)
166
167     } // namespace fatfs
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	src/diskio_layer/implementation/diskio_usb.cpp	**Lines:**	0	12	0.0 %
Date:	2022-03-20 01:57:36	**Branches:**	0	0	- %

Line	Branch	Exec	Source
1			
2			/*--/
3			/ FatFs - Generic FAT Filesystem module R0.14b /
4			/--/
5			/
6			/ Copyright (C) 2021, ChaN, all right reserved.
7			/
8			/ FatFs module is an open source software. Redistribution and use of FatFs in
9			/ source and binary forms, with or without modification, are permitted provided
10			/ that the following condition is met:
11			
12			/ 1. Redistributions of source code must retain the above copyright notice,
13			/ this condition and the following disclaimer.
14			/
15			/ This software is provided by the copyright holder and contributors "AS IS"
16			/ and any warranties related to this software are DISCLAIMED.
17			/ The copyright owner or contributors be NOT LIABLE for any damages caused
18			/ by use of this software.
19			/
20			/--*/
21			
22			// C++ port of the original source code is subject to MIT License
23			
24			// Copyright (c) 2022 Chris Sutton
25			
26			// Permission is hereby granted, free of charge, to any person obtaining a copy
27			// of this software and associated documentation files (the "Software"), to deal
28			// in the Software without restriction, including without limitation the rights
29			// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
30			// copies of the Software, and to permit persons to whom the Software is
31			// furnished to do so, subject to the following conditions:
32			
33			// The above copyright notice and this permission notice shall be included in all
34			// copies or substantial portions of the Software.
35			
36			// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
37			// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
38			// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
39			// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
40			// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
41			// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
42			// SOFTWARE.
43			
44			#include <diskio_hardware_usb.hpp>
45			
46			
47			namespace fatfs {
48			
49			template<>
50			DiskioHardwareBase::DSTATUS DiskioHardwareUSB<DiskioProtocolUSB>::initialize(BYTE pdrv [[maybe_unused]])
51			{
52			DSTATUS res = 0;
53			return res;
54			}
55			
56			template<>
57			DiskioHardwareBase::DSTATUS DiskioHardwareUSB<DiskioProtocolUSB>::status(BYTE pdrv [[maybe_unused]])
58			{
59			DSTATUS res = 0;
60			return res;
61			}
62			
63			template<>
64			DiskioHardwareBase::DRESULT DiskioHardwareUSB<DiskioProtocolUSB>::read(
65			BYTE pdrv [[maybe_unused]],
66			BYTE* buff [[maybe_unused]],
67			LBA_t sector [[maybe_unused]],
68			UINT count [[maybe_unused]])
69			{
70			return DRESULT::RES_OK;
71			}

```
72
73          template<>
74          DiskioHardwareBase::DRESULT DiskioHardwareUSB<DiskioProtocolUSB>::write(
75              BYTE pdrv [[maybe_unused]],
76              const BYTE* buff [[maybe_unused]],
77              LBA_t sector [[maybe_unused]],
78              UINT count [[maybe_unused]])
79          {
80              return DRESULT::RES_OK;
81          }
82
83          template<>
84          DiskioHardwareBase::DRESULT DiskioHardwareUSB<DiskioProtocolUSB>::ioctl (
85              BYTE pdrv [[maybe_unused]],
86              BYTE cmd [[maybe_unused]],
87              void *buff [[maybe_unused]])
88          {
89              return DRESULT::RES_OK;
90          }
91
92
93          } // namespace fatfs
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	src/ff_driver_common.cpp	Lines:	0	95	0.0 %
Date:	2022-03-20 01:57:36	Branches:	0	76	0.0 %

```
Line Branch Exec  Source
   1                /*----------------------------------------------------------------------------/
   2                /  FatFs - Generic FAT Filesystem Module  R0.14b                             /
   3                /-----------------------------------------------------------------------------/
   4                /
   5                / Copyright (C) 2021, ChaN, all right reserved.
   6                /
   7                / FatFs module is an open source software. Redistribution and use of FatFs in
   8                / source and binary forms, with or without modification, are permitted provided
   9                / that the following condition is met:
  10                /
  11                / 1. Redistributions of source code must retain the above copyright notice,
  12                /    this condition and the following disclaimer.
  13                /
  14                / This software is provided by the copyright holder and contributors "AS IS"
  15                / and any warranties related to this software are DISCLAIMED.
  16                / The copyright owner or contributors be NOT LIABLE for any damages caused
  17                / by use of this software.
  18                /
  19                /----------------------------------------------------------------------------*/
  20
  21
  22                // C++ port of the original source code is subject to MIT License
  23
  24                // Copyright (c) 2022 Chris Sutton
  25
  26                // Permission is hereby granted, free of charge, to any person obtaining a copy
  27                // of this software and associated documentation files (the "Software"), to deal
  28                // in the Software without restriction, including without limitation the rights
  29                // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  30                // copies of the Software, and to permit persons to whom the Software is
  31                // furnished to do so, subject to the following conditions:
  32
  33                // The above copyright notice and this permission notice shall be included in all
  34                // copies or substantial portions of the Software.
  35
  36                // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  37                // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  38                // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  39                // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  40                // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  41                // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  42                // SOFTWARE.
  43
  44                #include <ff_driver_common.hpp>
  45                #include <cstring>
  46
  47                namespace fatfs
  48                {
  49
  50                #if FF_FS_LOCK != 0
  51                [[maybe_unused]] static FILESEM Files[FF_FS_LOCK]; /* Open object lock semaphores */
  52                #endif
  53
  54                /* Character code support macros */
  55                #define IsSeparator(c) ((c) == '/' || (c) == '\\')
  56                #define IsTerminator(c) ((UINT)(c) < (FF_USE_LFN ? ' ' : '!'))
  57                #define IsSurrogate(c) ((c) >= 0xD800 && (c) <= 0xDFFF)
  58                #define IsSurrogateH(c) ((c) >= 0xD800 && (c) <= 0xDBFF)
  59                #define IsSurrogateL(c) ((c) >= 0xDC00 && (c) <= 0xDFFF)
  60
  61
  62
  63
  64
  65                /*----------------------------------------------------------------------*/
  66                /* Load/Store multi-byte word in the FAT structure                      */
  67                /*----------------------------------------------------------------------*/
  68
  69    #           WORD DriverCommon::ld_word (const BYTE* ptr) /*  Load a 2-byte little-endian word */
  70    #           {
  71    #            WORD rv;
  72
  73    #            rv = ptr[1];
  74    #            rv = rv << 8 | ptr[0];
  75    #            return rv;
  76    #           }
  77
  78    #           DWORD DriverCommon::ld_dword (const BYTE* ptr) /* Load a 4-byte little-endian word */
  79    #           {
  80    #            DWORD rv;
  81
  82    #            rv = ptr[3];
  83    #            rv = rv << 8 | ptr[2];
  84    #            rv = rv << 8 | ptr[1];
  85    #            rv = rv << 8 | ptr[0];
  86    #            return rv;
  87    #           }
  88
  89                #if FF_FS_EXFAT
  90                QWORD DriverCommon::ld_qword (const BYTE* ptr) /* Load an 8-byte little-endian word */
  91                {
  92                 QWORD rv;
```

```
 rv = ptr[7];
 rv = rv << 8 | ptr[6];
 rv = rv << 8 | ptr[5];
 rv = rv << 8 | ptr[4];
 rv = rv << 8 | ptr[3];
 rv = rv << 8 | ptr[2];
 rv = rv << 8 | ptr[1];
 rv = rv << 8 | ptr[0];
 return rv;
}
#endif

#if !FF_FS_READONLY
void DriverCommon::st_word (BYTE* ptr, WORD val) /* Store a 2-byte word in little-endian */
{
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val;
}

void DriverCommon::st_dword (BYTE* ptr, DWORD val) /* Store a 4-byte word in little-endian */
{
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val;
}

#if FF_FS_EXFAT
void DriverCommon::st_qword (BYTE* ptr, QWORD val) /* Store an 8-byte word in little-endian */
{
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val; val >>= 8;
 *ptr++ = (BYTE)val;
}
#endif
#endif /* !FF_FS_READONLY */

/*----------------------------------------------------------------------*/
/* String functions                                                     */
/*----------------------------------------------------------------------*/

/* Test if the byte is DBC 1st byte */
int DriverCommon::dbc_1st (BYTE c)
{
#if FF_CODE_PAGE == 0  /* Variable code page */
 if (DbcTbl && c >= DbcTbl[0]) {
  if (c <= DbcTbl[1]) return 1;     /* 1st byte range 1 */
  if (c >= DbcTbl[2] && c <= DbcTbl[3]) return 1; /* 1st byte range 2 */
 }
#elif FF_CODE_PAGE >= 900 /* DBCS fixed code page */
 if (c >= DbcTbl[0]) {
  if (c <= DbcTbl[1]) return 1;
  if (c >= DbcTbl[2] && c <= DbcTbl[3]) return 1;
 }
#else      /* SBCS fixed code page */
 if (c != 0) return 0; /* Always false */
#endif
 return 0;
}


/* Test if the byte is DBC 2nd byte */
int DriverCommon::dbc_2nd (BYTE c)
{
#if FF_CODE_PAGE == 0  /* Variable code page */
 if (DbcTbl && c >= DbcTbl[4]) {
  if (c <= DbcTbl[5]) return 1;     /* 2nd byte range 1 */
  if (c >= DbcTbl[6] && c <= DbcTbl[7]) return 1; /* 2nd byte range 2 */
  if (c >= DbcTbl[8] && c <= DbcTbl[9]) return 1; /* 2nd byte range 3 */
 }
#elif FF_CODE_PAGE >= 900 /* DBCS fixed code page */
 if (c >= DbcTbl[4]) {
  if (c <= DbcTbl[5]) return 1;
  if (c >= DbcTbl[6] && c <= DbcTbl[7]) return 1;
  if (c >= DbcTbl[8] && c <= DbcTbl[9]) return 1;
 }
#else      /* SBCS fixed code page */
 if (c != 0) return 0; /* Always false */
#endif
 return 0;
}


#if FF_USE_LFN

/* Get a Unicode code point from the TCHAR string in defined API encodeing */
DWORD DriverCommon::tchar2uni ( /* Returns a character in UTF-16 encoding (>=0x10000 on surrogate pair, 0xFFFFFFFF on decode error) */
 const TCHAR** str  /* Pointer to pointer to TCHAR string in configured encoding */
)
{
 DWORD uc;
 const TCHAR *p = *str;

#if FF_LFN_UNICODE == 1  /* UTF-16 input */
 WCHAR wc;
```

```
194
195         uc = *p++; /* Get a unit */
196         if (IsSurrogate(uc)) { /* Surrogate? */
197          wc = *p++;  /* Get low surrogate */
198          if (!IsSurrogateH(uc) || !IsSurrogateL(wc)) return 0xFFFFFFFF; /* Wrong surrogate? */
199          uc = uc << 16 | wc;
200         }
201
202        #elif FF_LFN_UNICODE == 2 /* UTF-8 input */
203         BYTE b;
204         int nf;
205
206         uc = (BYTE)*p++; /* Get an encoding unit */
207         if (uc & 0x80) { /* Multiple byte code? */
208          if        ((uc & 0xE0) == 0xC0) { /* 2-byte sequence? */
209           uc &= 0x1F; nf = 1;
210          } else if ((uc & 0xF0) == 0xE0) { /* 3-byte sequence? */
211           uc &= 0x0F; nf = 2;
212          } else if ((uc & 0xF8) == 0xF0) { /* 4-byte sequence? */
213           uc &= 0x07; nf = 3;
214          } else {       /* Wrong sequence */
215           return 0xFFFFFFFF;
216          }
217          do { /* Get trailing bytes */
218           b = (BYTE)*p++;
219           if ((b & 0xC0) != 0x80) return 0xFFFFFFFF; /* Wrong sequence? */
220           uc = uc << 6 | (b & 0x3F);
221          } while (--nf != 0);
222          if (uc < 0x80 || IsSurrogate(uc) || uc >= 0x110000) return 0xFFFFFFFF; /* Wrong code? */
223          if (uc >= 0x010000) uc = 0xD800DC00 | ((uc - 0x10000) << 6 & 0x3FF0000) | (uc & 0x3FF); /* Make a surrogate pair if needed */
224         }
225
226        #elif FF_LFN_UNICODE == 3 /* UTF-32 input */
227         uc = (TCHAR)*p++; /* Get a unit */
228         if (uc >= 0x110000 || IsSurrogate(uc)) return 0xFFFFFFFF; /* Wrong code? */
229         if (uc >= 0x010000) uc = 0xD800DC00 | ((uc - 0x10000) << 6 & 0x3FF0000) | (uc & 0x3FF); /* Make a surrogate pair if needed */
230
231        #else  /* ANSI/OEM input */
232         BYTE b;
233         WCHAR wc;
234
235         wc = (BYTE)*p++;   /* Get a byte */
236         if (dbc_1st((BYTE)wc)) { /* Is it a DBC 1st byte? */
237          b = (BYTE)*p++;   /* Get 2nd byte */
238          if (!dbc_2nd(b)) return 0xFFFFFFFF; /* Invalid code? */
239          wc = (wc << 8) + b;  /* Make a DBC */
240         }
241         if (wc != 0) {
242          wc = ff_oem2uni(wc, CODEPAGE); /* ANSI/OEM ==> Unicode */
243          if (wc == 0) return 0xFFFFFFFF; /* Invalid code? */
244         }
245         uc = wc;
246
247        #endif
248         *str = p; /* Next read pointer */
249         return uc;
250        }
251
252
253        /* Store a Unicode char in defined API encoding */
254        UINT DriverCommon::put_utf ( /* Returns number of encoding units written (0:buffer overflow or wrong encoding) */
255         DWORD chr, /* UTF-16 encoded character (Surrogate pair if >=0x10000) */
256         TCHAR* buf, /* Output buffer */
257         UINT szb /* Size of the buffer */
258        )
259        {
260        #if FF_LFN_UNICODE == 1 /* UTF-16 output */
261         WCHAR hs, wc;
262
263         hs = (WCHAR)(chr >> 16);
264         wc = (WCHAR)chr;
265         if (hs == 0) { /* Single encoding unit? */
266          if (szb < 1 || IsSurrogate(wc)) return 0; /* Buffer overflow or wrong code? */
267          *buf = wc;
268          return 1;
269         }
270         if (szb < 2 || !IsSurrogateH(hs) || !IsSurrogateL(wc)) return 0; /* Buffer overflow or wrong surrogate? */
271         *buf++ = hs;
272         *buf++ = wc;
273         return 2;
274
275        #elif FF_LFN_UNICODE == 2 /* UTF-8 output */
276         DWORD hc;
277
278         if (chr < 0x80) { /* Single byte code? */
279          if (szb < 1) return 0; /* Buffer overflow? */
280          *buf = (TCHAR)chr;
281          return 1;
282         }
283         if (chr < 0x800) { /* 2-byte sequence? */
284          if (szb < 2) return 0; /* Buffer overflow? */
285          *buf++ = (TCHAR)(0xC0 | (chr >> 6 & 0x1F));
286          *buf++ = (TCHAR)(0x80 | (chr >> 0 & 0x3F));
287          return 2;
288         }
289         if (chr < 0x10000) { /* 3-byte sequence? */
290          if (szb < 3 || IsSurrogate(chr)) return 0; /* Buffer overflow or wrong code? */
291          *buf++ = (TCHAR)(0xE0 | (chr >> 12 & 0x0F));
292          *buf++ = (TCHAR)(0x80 | (chr >> 6 & 0x3F));
293          *buf++ = (TCHAR)(0x80 | (chr >> 0 & 0x3F));
294          return 3;
```

```
 }
 /* 4-byte sequence */
 if (szb < 4) return 0; /* Buffer overflow? */
 hc = ((chr & 0xFFFF0000) - 0xD8000000) >> 6; /* Get high 10 bits */
 chr = (chr & 0xFFFF) - 0xDC00;     /* Get low 10 bits */
 if (hc >= 0x100000 || chr >= 0x400) return 0; /* Wrong surrogate? */
 chr = (hc | chr) + 0x10000;
 *buf++ = (TCHAR)(0xF0 | (chr >> 18 & 0x07));
 *buf++ = (TCHAR)(0x80 | (chr >> 12 & 0x3F));
 *buf++ = (TCHAR)(0x80 | (chr >> 6 & 0x3F));
 *buf++ = (TCHAR)(0x80 | (chr >> 0 & 0x3F));
 return 4;

#elif FF_LFN_UNICODE == 3 /* UTF-32 output */
 DWORD hc;

 if (szb < 1) return 0; /* Buffer overflow? */
 if (chr >= 0x10000) { /* Out of BMP? */
  hc = ((chr & 0xFFFF0000) - 0xD8000000) >> 6; /* Get high 10 bits */
  chr = (chr & 0xFFFF) - 0xDC00;     /* Get low 10 bits */
  if (hc >= 0x100000 || chr >= 0x400) return 0; /* Wrong surrogate? */
  chr = (hc | chr) + 0x10000;
 }
 *buf++ = (TCHAR)chr;
 return 1;

#else      /* ANSI/OEM output */
 WCHAR wc;

 wc = ff_uni2oem(chr, CODEPAGE);
 if (wc >= 0x100) { /* Is this a DBC? */
  if (szb < 2) return 0;
  *buf++ = (char)(wc >> 8); /* Store DBC 1st byte */
  *buf++ = (TCHAR)wc;   /* Store DBC 2nd byte */
  return 2;
 }
 if (wc == 0 || szb < 1) return 0; /* Invalid char or buffer overflow? */
 *buf++ = (TCHAR)wc;     /* Store the character */
 return 1;
#endif
}
#endif /* FF_USE_LFN */


#if FF_FS_REENTRANT
/*------------------------------------------------------------------------*/
/* Request/Release grant to access the volume                             */
/*------------------------------------------------------------------------*/
int DriverCommon::lock_fs (  /* 1:Ok, 0:timeout */
 FATFS* fs  /* Filesystem object */
)
{
 return ff_req_grant(fs->sobj);
}


void DriverCommon::unlock_fs (
 FATFS* fs,  /* Filesystem object */
 FRESULT res  /* Result code to be returned */
)
{
 if (fs && res != FR_NOT_ENABLED && res != FR_INVALID_DRIVE && res != FR_TIMEOUT) {
  ff_rel_grant(fs->sobj);
 }
}

#endif // FF_FS_REENTRANT



#if FF_FS_LOCK != 0
/*------------------------------------------------------------------------*/
/* File lock control functions                                            */
/*------------------------------------------------------------------------*/

FRESULT DriverCommon::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 mode open, 1:Write mode open, 2:Delete or rename) */
)
{
 UINT i, be;

 /* Search open object table for the object */
 be = 0;
 for (i = 0; i < FF_FS_LOCK; i++) {
  if (Files[i].fs) { /* Existing entry */
   if (Files[i].fs == dp->obj.fs &&   /* Check if the object matches with an open object */
    Files[i].clu == dp->obj.sclust &&
    Files[i].ofs == dp->dptr) break;
  } else {   /* Blank entry */
   be = 1;
  }
 }
 if (i == FF_FS_LOCK) { /* The object has not been opened */
  return (!be && acc != 2) ? FR_TOO_MANY_OPEN_FILES : FR_OK; /* Is there a blank entry for new object? */
 }

 /* The object was opened. Reject any open against writing file and all write mode open */
 return (acc != 0 || Files[i].ctr == 0x100) ? FR_LOCKED : FR_OK;
}
```

```
int DriverCommon::enq_lock (void) /* Check if an entry is available for a new object */
{
 UINT i;

 for (i = 0; i < FF_FS_LOCK && Files[i].fs; i++) ;
 return (i == FF_FS_LOCK) ? 0 : 1;
}


UINT DriverCommon::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 < FF_FS_LOCK; i++) { /* Find the object */
  if (Files[i].fs == dp->obj.fs
   && Files[i].clu == dp->obj.sclust
   && Files[i].ofs == dp->dptr) break;
 }

 if (i == FF_FS_LOCK) {   /* Not opened. Register it as new. */
  for (i = 0; i < FF_FS_LOCK && Files[i].fs; i++) ;
  if (i == FF_FS_LOCK) return 0; /* No free entry to register (int err) */
  Files[i].fs = dp->obj.fs;
  Files[i].clu = dp->obj.sclust;
  Files[i].ofs = dp->dptr;
  Files[i].ctr = 0;
 }

 if (acc >= 1 && Files[i].ctr) return 0; /* Access violation (int err) */

 Files[i].ctr = acc ? 0x100 : Files[i].ctr + 1; /* Set semaphore value */

 return i + 1; /* Index number origin from 1 */
}


FRESULT DriverCommon::dec_lock ( /* Decrement object open counter */
 UINT i   /* Semaphore index (1..) */
)
{
 WORD n;
 FRESULT res;


 if (--i < FF_FS_LOCK) { /* Index number origin from 0 */
  n = Files[i].ctr;
  if (n == 0x100) n = 0; /* If write mode open, delete the entry */
  if (n > 0) n--;   /* Decrement read mode open count */
  Files[i].ctr = n;
  if (n == 0) 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;
}


void DriverCommon::clear_lock ( /* Clear lock entries of the volume */
 FATFS *fs
)
{
 UINT i;

 for (i = 0; i < FF_FS_LOCK; i++) {
  if (Files[i].fs == fs) Files[i].fs = 0;
 }
}
#endif /* FF_FS_LOCK != 0 */

/*-----------------------------------------------------------------------*/
/* Get physical sector number from cluster number                        */
/*-----------------------------------------------------------------------*/

LBA_t DriverCommon::clst2sect ( /* !=0:Sector number, 0:Failed (invalid cluster#) */
 FATFS* fs,  /* Filesystem object */
 DWORD clst  /* Cluster# to be converted */
)
{
 clst -= 2;  /* Cluster number is origin from 2 */
 if (clst >= fs->n_fatent - 2) return 0;  /* Is it invalid cluster number? */
 return fs->database + (LBA_t)fs->csize * clst; /* Start sector number of the cluster */
}



#if FF_USE_FASTSEEK
/*-----------------------------------------------------------------------*/
/* FAT handling - Convert offset into cluster with link map table        */
/*-----------------------------------------------------------------------*/

DWORD DriverCommon::clmt_clust ( /* <2:Error, >=2:Cluster number */
 FIL* fp,  /* Pointer to the file object */
 FSIZE_t ofs  /* File offset to be converted to cluster# */
)
```

```
{
 DWORD cl, ncl, *tbl;
 FATFS *fs = fp->obj.fs;


 tbl = fp->cltbl + 1; /* Top of CLMT */
 cl = (DWORD)(ofs / SS(fs) / fs->csize); /* Cluster order from top of the file */
 for (;;) {
  ncl = *tbl++;   /* Number of cluters in the fragment */
  if (ncl == 0) 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 /* FF_USE_FASTSEEK */



/*-----------------------------------------------------------------------*/
/* FAT: Directory handling - Load/Store start cluster number             */
/*-----------------------------------------------------------------------*/

DWORD DriverCommon::ld_clust ( /* Returns the top cluster value of the SFN entry */
 FATFS* fs,   /* Pointer to the fs object */
 const BYTE* dir  /* Pointer to the key 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 !FF_FS_READONLY
void DriverCommon::st_clust (
 FATFS* fs, /* Pointer to the fs object */
 BYTE* dir, /* Pointer to the key entry */
 DWORD cl /* Value to be set */
)
{
 st_word(dir + DIR_FstClusLO, (WORD)cl);
 if (fs->fs_type == FS_FAT32) {
  st_word(dir + DIR_FstClusHI, (WORD)(cl >> 16));
 }
}
#endif



#if FF_USE_LFN
/*-----------------------------------------------------------*/
/* FAT-LFN: Compare a part of file name with an LFN entry */
/*-----------------------------------------------------------*/

int DriverCommon::cmp_lfn (  /* 1:matched, 0:not matched */
 const WCHAR* lfnbuf, /* Pointer to the LFN working buffer to be compared */
 BYTE* dir    /* Pointer to the directory entry containing the part of LFN */
)
{
 UINT i, s;
 WCHAR wc, uc;


 if (ld_word(dir + LDIR_FstClusLO) != 0) return 0; /* Check LDIR_FstClusLO */

 i = ((dir[LDIR_Ord] & 0x3F) - 1) * 13; /* Offset in the LFN buffer */

 for (wc = 1, s = 0; s < 13; s++) {  /* Process all characters in the entry */
  uc = ld_word(dir + LfnOfs[s]);  /* Pick an LFN character */
  if (wc != 0) {
   if (i >= FF_MAX_LFN + 1 || f_wtoupper(uc) != f_wtoupper(lfnbuf[i++])) { /* Compare it */
    return 0;     /* Not matched */
   }
   wc = uc;
  } else {
   if (uc != 0xFFFF) return 0;  /* Check filler */
  }
 }

 if ((dir[LDIR_Ord] & LLEF) && wc && lfnbuf[i]) return 0; /* Last segment matched but different length */

 return 1;  /* The part of LFN matched */
}


#if FF_FS_MINIMIZE <= 1 || FF_FS_RPATH >= 2 || FF_USE_LABEL || FF_FS_EXFAT
/*-----------------------------------------------------*/
/* FAT-LFN: Pick a part of file name from an LFN entry */
/*-----------------------------------------------------*/

int DriverCommon::pick_lfn ( /* 1:succeeded, 0:buffer overflow or invalid LFN entry */
 WCHAR* lfnbuf,  /* Pointer to the LFN working buffer */
 BYTE* dir   /* Pointer to the LFN entry */
)
{
```

```
 UINT i, s;
 WCHAR wc, uc;


 if (ld_word(dir + LDIR_FstClusLO) != 0) return 0; /* Check LDIR_FstClusLO is 0 */

 i = ((dir[LDIR_Ord] & ~LLEF) - 1) * 13; /* Offset in the LFN buffer */

 for (wc = 1, s = 0; s < 13; s++) {  /* Process all characters in the entry */
  uc = ld_word(dir + LfnOfs[s]);  /* Pick an LFN character */
  if (wc != 0) {
   if (i >= FF_MAX_LFN + 1) return 0; /* Buffer overflow? */
   lfnbuf[i++] = wc = uc;   /* Store it */
  } else {
   if (uc != 0xFFFF) return 0;  /* Check filler */
  }
 }

 if (dir[LDIR_Ord] & LLEF && wc != 0) { /* Put terminator if it is the last LFN part and not terminated */
  if (i >= FF_MAX_LFN + 1) return 0; /* Buffer overflow? */
  lfnbuf[i] = 0;
 }

 return 1;  /* The part of LFN is valid */
}
#endif


#if !FF_FS_READONLY
/*-----------------------------------------*/
/* FAT-LFN: Create an entry of LFN entries */
/*-----------------------------------------*/

void DriverCommon::put_lfn (
 const WCHAR* lfn, /* Pointer to the LFN */
 BYTE* dir,   /* Pointer to the LFN entry to be created */
 BYTE ord,   /* LFN order (1-20) */
 BYTE sum   /* Checksum of the corresponding SFN */
)
{
 UINT i, s;
 WCHAR wc;


 dir[LDIR_Chksum] = sum;   /* Set checksum */
 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 working buffer */
 s = wc = 0;
 do {
  if (wc != 0xFFFF) wc = lfn[i++]; /* Get an effective character */
  st_word(dir + LfnOfs[s], wc);  /* Put it */
  if (wc == 0) wc = 0xFFFF;   /* Padding characters for following items */
 } while (++s < 13);
 if (wc == 0xFFFF || !lfn[i]) ord |= LLEF; /* Last LFN part is the start of LFN sequence */
 dir[LDIR_Ord] = ord;   /* Set the LFN order */
}

#endif /* !FF_FS_READONLY */
#endif /* FF_USE_LFN */



#if FF_USE_LFN && !FF_FS_READONLY
/*-----------------------------------------------------------------------*/
/* FAT-LFN: Create a Numbered SFN                                        */
/*-----------------------------------------------------------------------*/

void DriverCommon::gen_numname (
 BYTE* dst,   /* Pointer to the buffer to store numbered SFN */
 const BYTE* src, /* Pointer to SFN in directory form */
 const WCHAR* lfn, /* Pointer to LFN */
 UINT seq   /* Sequence number */
)
{
 BYTE ns[8], c;
 UINT i, j;
 WCHAR wc;
 DWORD sreg;


 std::memcpy(dst, src, 11); /* Prepare the SFN to be modified */

 if (seq > 5) { /* In case of many collisions, generate a hash number instead of sequential number */
  sreg = seq;
  while (*lfn) { /* Create a CRC as hash value */
   wc = *lfn++;
   for (i = 0; i < 16; i++) {
    sreg = (sreg << 1) + (wc & 1);
    wc >>= 1;
    if (sreg & 0x10000) sreg ^= 0x11021;
   }
  }
  seq = (UINT)sreg;
 }

 /* Make suffix (~ + hexdecimal) */
 i = 7;
 do {
```

```
  c = (BYTE)((seq % 16) + '0'); seq /= 16;
  if (c > '9') c += 7;
  ns[i--] = c;
 } while (i && seq);
 ns[i] = '~';

 /* Append the suffix to the SFN body */
 for (j = 0; j < i && dst[j] != ' '; j++) { /* Find the offset to append */
  if (dbc_1st(dst[j])) { /* To avoid DBC break up */
   if (j == i - 1) break;
   j++;
  }
 }
 do { /* Append the suffix */
  dst[j++] = (i < 8) ? ns[i++] : ' ';
 } while (j < 8);
}
#endif /* FF_USE_LFN && !FF_FS_READONLY */



#if FF_USE_LFN
/*-----------------------------------------------------------------------*/
/* FAT-LFN: Calculate checksum of an SFN entry                           */
/*-----------------------------------------------------------------------*/

BYTE DriverCommon::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 /* FF_USE_LFN */



#if FF_FS_EXFAT
/*-----------------------------------------------------------------------*/
/* exFAT: Checksum                                                       */
/*-----------------------------------------------------------------------*/

WORD DriverCommon::xdir_sum ( /* Get checksum of the directoly entry block */
 const BYTE* dir  /* Directory entry block to be calculated */
)
{
 UINT i, szblk;
 WORD sum;


 szblk = (dir[XDIR_NumSec] + 1) * SZDIRE; /* Number of bytes of the entry block */
 for (i = sum = 0; i < szblk; i++) {
  if (i == XDIR_SetSum) { /* Skip 2-byte sum field */
   i++;
  } else {
   sum = ((sum & 1) ? 0x8000 : 0) + (sum >> 1) + dir[i];
  }
 }
 return sum;
}



WORD DriverCommon::xname_sum ( /* Get check sum (to be used as hash) of the file name */
 const WCHAR* name /* File name to be calculated */
)
{
 WCHAR chr;
 WORD sum = 0;


 while ((chr = *name++) != 0) {
  chr = (WCHAR)f_wtoupper(chr);  /* File name needs to be up-case converted */
  sum = ((sum & 1) ? 0x8000 : 0) + (sum >> 1) + (chr & 0xFF);
  sum = ((sum & 1) ? 0x8000 : 0) + (sum >> 1) + (chr >> 8);
 }
 return sum;
}


#if !FF_FS_READONLY && FF_USE_MKFS
DWORD DriverCommon::xsum32 ( /* Returns 32-bit checksum */
 BYTE  dat,   /* Byte to be calculated (byte-by-byte processing) */
 DWORD sum   /* Previous sum value */
)
{
 sum = ((sum & 1) ? 0x80000000 : 0) + (sum >> 1) + dat;
 return sum;
}
#endif


/*-----------------------------------------------------------------------*/
/* exFAT: Initialize object allocation info with loaded entry block */
/*-----------------------------------------------------------------------*/
```

```
799
800         void DriverCommon::init_alloc_info (
801          FATFS* fs,  /* Filesystem object */
802          FFOBJID* obj /* Object allocation information to be initialized */
803         )
804         {
805          obj->sclust = ld_dword(fs->dirbuf + XDIR_FstClus);  /* Start cluster */
806          obj->objsize = ld_qword(fs->dirbuf + XDIR_FileSize); /* Size */
807          obj->stat = fs->dirbuf[XDIR_GenFlags] & 2;    /* Allocation status */
808          obj->n_frag = 0;          /* No last fragment info */
809         }
810
811         #endif // #if FF_FS_EXFAT
812
813         #if FF_FS_EXFAT
814         #if !FF_FS_READONLY
815
816         /*----------------------------------------*/
817         /* exFAT: Create a new directory enrty block */
818         /*----------------------------------------*/
819
820         void DriverCommon::create_xdir (
821          BYTE* dirb,   /* Pointer to the direcotry entry block buffer */
822          const WCHAR* lfn /* Pointer to the object name */
823         )
824         {
825          UINT i;
826          BYTE nc1, nlen;
827          WCHAR wc;
828
829
830          /* Create file-directory and stream-extension entry */
831          std::memset(dirb, 0, 2 * SZDIRE);
832          dirb[0 * SZDIRE + XDIR_Type] = ET_FILEDIR;
833          dirb[1 * SZDIRE + XDIR_Type] = ET_STREAM;
834
835          /* Create file-name entries */
836          i = SZDIRE * 2; /* Top of file_name entries */
837          nlen = nc1 = 0; wc = 1;
838          do {
839           dirb[i++] = ET_FILENAME; dirb[i++] = 0;
840           do { /* Fill name field */
841            if (wc != 0 && (wc = lfn[nlen]) != 0) nlen++; /* Get a character if exist */
842            st_word(dirb + i, wc);  /* Store it */
843            i += 2;
844           } while (i % SZDIRE != 0);
845           nc1++;
846          } while (lfn[nlen]); /* Fill next entry if any char follows */
847
848          dirb[XDIR_NumName] = nlen;  /* Set name length */
849          dirb[XDIR_NumSec] = 1 + nc1; /* Set secondary count (C0 + C1s) */
850          st_word(dirb + XDIR_NameHash, xname_sum(lfn)); /* Set name hash */
851         }
852
853         #endif /* !FF_FS_READONLY */
854         #endif /* FF_FS_EXFAT */
855
856
857         #if FF_FS_MINIMIZE <= 1 || FF_FS_RPATH >= 2
858         /*-----------------------------------------------------------------------*/
859         /* Get file information from directory entry                             */
860         /*-----------------------------------------------------------------------*/
861
862         void DriverCommon::get_fileinfo (
863          DIR* dp,   /* Pointer to the directory object */
864          FILINFO* fno  /* Pointer to the file information to be filled */
865         )
866         {
867          UINT si, di;
868         #if FF_USE_LFN
869          BYTE lcf;
870          WCHAR wc, hs;
871          FATFS *fs = dp->obj.fs;
872          UINT nw;
873         #else
874          TCHAR c;
875         #endif
876
877
878          fno->fname[0] = 0;   /* Invaidate file info */
879          if (dp->sect == 0) return; /* Exit if read pointer has reached end of directory */
880
881         #if FF_USE_LFN  /* LFN configuration */
882         #if FF_FS_EXFAT
883          if (fs->fs_type == FS_EXFAT) { /* exFAT volume */
884           UINT nc = 0;
885
886           si = SZDIRE * 2; di = 0; /* 1st C1 entry in the entry block */
887           hs = 0;
888           while (nc < fs->dirbuf[XDIR_NumName]) {
889            if (si >= MAXDIRB(FF_MAX_LFN)) { di = 0; break; } /* Truncated directory block? */
890            if ((si % SZDIRE) == 0) si += 2;  /* Skip entry type field */
891            wc = ld_word(fs->dirbuf + si); si += 2; nc++; /* Get a character */
892            if (hs == 0 && IsSurrogate(wc)) {  /* Is it a surrogate? */
893             hs = wc; continue;     /* Get low surrogate */
894            }
895            nw = put_utf((DWORD)hs << 16 | wc, &fno->fname[di], FF_LFN_BUF - di); /* Store it in API encoding */
896            if (nw == 0) { di = 0; break; }   /* Buffer overflow or wrong char? */
897            di += nw;
898            hs = 0;
899           }
```

```
  if (hs != 0) di = 0;     /* Broken surrogate pair? */
  if (di == 0) fno->fname[di++] = '?'; /* Inaccessible object name? */
  fno->fname[di] = 0;      /* Terminate the name */
  fno->altname[0] = 0;     /* exFAT does not support SFN */

  fno->fattrib = fs->dirbuf[XDIR_Attr] & AM_MASKX;  /* Attribute */
  fno->fsize = (fno->fattrib & AM_DIR) ? 0 : ld_qword(fs->dirbuf + XDIR_FileSize); /* Size */
  fno->ftime = ld_word(fs->dirbuf + XDIR_ModTime + 0); /* Time */
  fno->fdate = ld_word(fs->dirbuf + XDIR_ModTime + 2); /* Date */
  return;
 } else
#endif
 { /* FAT/FAT32 volume */
  if (dp->blk_ofs != 0xFFFFFFFF) { /* Get LFN if available */
   si = di = 0;
   hs = 0;
   while (fs->lfnbuf[si] != 0) {
    wc = fs->lfnbuf[si++];  /* Get an LFN character (UTF-16) */
    if (hs == 0 && IsSurrogate(wc)) { /* Is it a surrogate? */
     hs = wc; continue;  /* Get low surrogate */
    }
    nw = put_utf((DWORD)hs << 16 | wc, &fno->fname[di], FF_LFN_BUF - di); /* Store it in API encoding */
    if (nw == 0) { di = 0; break; } /* Buffer overflow or wrong char? */
    di += nw;
    hs = 0;
   }
   if (hs != 0) di = 0; /* Broken surrogate pair? */
   fno->fname[di] = 0;  /* Terminate the LFN (null string means LFN is invalid) */
  }
 }

 si = di = 0;
 while (si < 11) {  /* Get SFN from SFN entry */
  wc = dp->dir[si++];   /* Get a char */
  if (wc == ' ') continue; /* Skip padding spaces */
  if (wc == RDDEM) wc = DDEM; /* Restore replaced DDEM character */
  if (si == 9 && di < FF_SFN_BUF) fno->altname[di++] = '.'; /* Insert a . if extension is exist */
#if FF_LFN_UNICODE >= 1 /* Unicode output */
  if (dbc_1st((BYTE)wc) && si != 8 && si != 11 && dbc_2nd(dp->dir[si])) { /* Make a DBC if needed */
   wc = wc << 8 | dp->dir[si++];
  }
  wc = f_oem2uni(wc, CODEPAGE);  /* ANSI/OEM -> Unicode */
  if (wc == 0) { di = 0; break; }  /* Wrong char in the current code page? */
  nw = put_utf(wc, &fno->altname[di], FF_SFN_BUF - di); /* Store it in API encoding */
  if (nw == 0) { di = 0; break; }  /* Buffer overflow? */
  di += nw;
#else     /* ANSI/OEM output */
  fno->altname[di++] = (TCHAR)wc; /* Store it without any conversion */
#endif
 }
 fno->altname[di] = 0; /* Terminate the SFN  (null string means SFN is invalid) */

 if (fno->fname[0] == 0) { /* If LFN is invalid, altname[] needs to be copied to fname[] */
  if (di == 0) { /* If LFN and SFN both are invalid, this object is inaccesible */
   fno->fname[di++] = '?';
  } else {
   for (si = di = 0, lcf = NS_BODY; fno->altname[si]; si++, di++) { /* Copy altname[] to fname[] with case information */
    wc = (WCHAR)fno->altname[si];
    if (wc == '.') lcf = NS_EXT;
    if (std::isupper(wc) && (dp->dir[DIR_NTres] & lcf)) wc += 0x20;
    fno->fname[di] = (TCHAR)wc;
   }
  }
  fno->fname[di] = 0; /* Terminate the LFN */
  if (!dp->dir[DIR_NTres]) fno->altname[0] = 0; /* Altname is not needed if neither LFN nor case info is exist. */
 }

#else /* Non-LFN configuration */
 si = di = 0;
 while (si < 11) {  /* Copy name body and extension */
  c = (TCHAR)dp->dir[si++];
  if (c == ' ') continue;  /* Skip padding spaces */
  if (c == RDDEM) c = DDEM; /* Restore replaced DDEM character */
  if (si == 9) fno->fname[di++] = '.';/* Insert a . if extension is exist */
  fno->fname[di++] = c;
 }
 fno->fname[di] = 0;  /* Terminate the SFN */
#endif

 fno->fattrib = dp->dir[DIR_Attr] & AM_MASK;   /* Attribute */
 fno->fsize = ld_dword(dp->dir + DIR_FileSize);  /* Size */
 fno->ftime = ld_word(dp->dir + DIR_ModTime + 0); /* Time */
 fno->fdate = ld_word(dp->dir + DIR_ModTime + 2); /* Date */
}

#endif /* FF_FS_MINIMIZE <= 1 || FF_FS_RPATH >= 2 */




#if FF_USE_FIND && FF_FS_MINIMIZE <= 1
/*-----------------------------------------------------------------------*/
/* Pattern matching                                                      */
/*-----------------------------------------------------------------------*/

#define FIND_RECURS 4 /* Maximum number of wildcard terms in the pattern to limit recursion */


DWORD DriverCommon::get_achar ( /* Get a character and advance ptr */
 const TCHAR** ptr  /* Pointer to pointer to the ANSI/OEM or Unicode string */
)
{
```

```
 DWORD chr;


#if FF_USE_LFN && FF_LFN_UNICODE >= 1 /* Unicode input */
 chr = tchar2uni(ptr);
 if (chr == 0xFFFFFFFF) chr = 0;  /* Wrong UTF encoding is recognized as end of the string */
 chr = f_wtoupper(chr);

#else         /* ANSI/OEM input */
 chr = (BYTE)*(*ptr)++;    /* Get a byte */
 if (std::islower(chr)) chr -= 0x20;  /* To upper ASCII char */
#if FF_CODE_PAGE == 0
 if (ExCvt && chr >= 0x80) chr = ExCvt[chr - 0x80]; /* To upper SBCS extended char */
#elif FF_CODE_PAGE < 900
 if (chr >= 0x80) chr = ExCvt[chr - 0x80]; /* To upper SBCS extended char */
#endif
#if FF_CODE_PAGE == 0 || FF_CODE_PAGE >= 900
 if (dbc_1st((BYTE)chr)) { /* Get DBC 2nd byte if needed */
  chr = dbc_2nd((BYTE)**ptr) ? chr << 8 | (BYTE)*(*ptr)++ : 0;
 }
#endif

#endif
 return chr;
}


int DriverCommon::pattern_match ( /* 0:mismatched, 1:matched */
 const TCHAR* pat, /* Matching pattern */
 const TCHAR* nam, /* String to be tested */
 UINT skip,   /* Number of pre-skip chars (number of ?s, b8:infinite (* specified)) */
 UINT recur   /* Recursion count */
)
{
 const TCHAR *pptr, *nptr;
 DWORD pchr, nchr;
 UINT sk;


 while ((skip & 0xFF) != 0) {  /* Pre-skip name chars */
  if (!get_achar(&nam)) return 0; /* Branch mismatched if less name chars */
  skip--;
 }
 if (*pat == 0 && skip) return 1; /* Matched? (short circuit) */

 do {
  pptr = pat; nptr = nam;   /* Top of pattern and name to match */
  for (;;) {
   if (*pptr == '?' || *pptr == '*') { /* Wildcard term? */
    if (recur == 0) return 0; /* Too many wildcard terms? */
    sk = 0;
    do { /* Analyze the wildcard term */
     if (*pptr++ == '?') sk++; else sk |= 0x100;
    } while (*pptr == '?' || *pptr == '*');
    if (pattern_match(pptr, nptr, sk, recur - 1)) return 1; /* Test new branch (recursive call) */
    nchr = *nptr; break; /* Branch mismatched */
   }
   pchr = get_achar(&pptr); /* Get a pattern char */
   nchr = get_achar(&nptr); /* Get a name char */
   if (pchr != nchr) break; /* Branch mismatched? */
   if (pchr == 0) return 1; /* Branch matched? (matched at end of both strings) */
  }
  get_achar(&nam);   /* nam++ */
 } while (skip && nchr);  /* Retry until end of name if infinite search is specified */

 return 0;
}

#endif /* FF_USE_FIND && FF_FS_MINIMIZE <= 1 */



/*-----------------------------------------------------------------------*/
/* Pick a top segment and create the object name in directory form       */
/*-----------------------------------------------------------------------*/

FRESULT DriverCommon::create_name ( /* FR_OK: successful, FR_INVALID_NAME: could not create */
 DIR* dp,     /* Pointer to the directory object */
 const TCHAR** path   /* Pointer to pointer to the segment in the path string */
)
{
#if FF_USE_LFN  /* LFN configuration */
 BYTE b, cf;
 WCHAR wc, *lfn;
 DWORD uc;
 UINT i, ni, si, di;
 const TCHAR *p;


 /* Create LFN into LFN working buffer */
 p = *path; lfn = dp->obj.fs->lfnbuf; di = 0;
 for (;;) {
  uc = tchar2uni(&p);   /* Get a character */
  if (uc == 0xFFFFFFFF) return FR_INVALID_NAME;  /* Invalid code or UTF decode error */
  if (uc >= 0x10000) lfn[di++] = (WCHAR)(uc >> 16); /* Store high surrogate if needed */
  wc = (WCHAR)uc;
  if (wc < ' ' || IsSeparator(wc)) break; /* Break if end of the path or a separator is found */
  if (wc < 0x80 && strchr("*:<>|\"\?\x7F", (int)wc)) return FR_INVALID_NAME; /* Reject illegal characters for LFN */
  if (di >= FF_MAX_LFN) return FR_INVALID_NAME; /* Reject too long name */
  lfn[di++] = wc;    /* Store the Unicode character */
 }
 if (wc < ' ') {    /* Stopped at end of the path? */
```

```
            cf = NS_LAST;   /* Last segment */
           } else {     /* Stopped at a separator */
            while (IsSeparator(*p)) p++; /* Skip duplicated separators if exist */
            cf = 0;     /* Next segment may follow */
            if (IsTerminator(*p)) cf = NS_LAST; /* Ignore terminating separator */
           }
           *path = p;     /* Return pointer to the next segment */

          #if FF_FS_RPATH != 0
           if ((di == 1 && lfn[di - 1] == '.') ||
            (di == 2 && lfn[di - 1] == '.' && lfn[di - 2] == '.')) { /* Is this segment a dot name? */
            lfn[di] = 0;
            for (i = 0; i < 11; i++) { /* Create dot name for SFN entry */
             dp->fn[i] = (i < di) ? '.' : ' ';
            }
            dp->fn[i] = cf | NS_DOT; /* This is a dot entry */
            return FR_OK;
           }
          #endif
           while (di) {     /* Snip off trailing spaces and dots if exist */
            wc = lfn[di - 1];
            if (wc != ' ' && wc != '.') break;
            di--;
           }
           lfn[di] = 0;       /* LFN is created into the working buffer */
           if (di == 0) return FR_INVALID_NAME; /* Reject null name */

           /* Create SFN in directory form */
           for (si = 0; lfn[si] == ' '; si++) ; /* Remove leading spaces */
           if (si > 0 || lfn[si] == '.') cf |= NS_LOSS | NS_LFN; /* Is there any leading space or dot? */
           while (di > 0 && lfn[di - 1] != '.') di--; /* Find last dot (di<=si: no extension) */

           std::memset(dp->fn, ' ', 11);
           i = b = 0; ni = 8;
           for (;;) {
            wc = lfn[si++];     /* Get an LFN character */
            if (wc == 0) break;    /* Break on end of the LFN */
            if (wc == ' ' || (wc == '.' && si != di)) { /* Remove embedded spaces and dots */
             cf |= NS_LOSS | NS_LFN;
             continue;
            }

            if (i >= ni || si == di) {  /* End of field? */
             if (ni == 11) {    /* Name extension overflow? */
              cf |= NS_LOSS | NS_LFN;
              break;
             }
             if (si != di) cf |= NS_LOSS | NS_LFN; /* Name body overflow? */
             if (si > di) break;      /* No name extension? */
             si = di; i = 8; ni = 11; b <<= 2;  /* Enter name extension */
             continue;
            }

            if (wc >= 0x80) { /* Is this an extended character? */
             cf |= NS_LFN; /* LFN entry needs to be created */
          #if FF_CODE_PAGE == 0
             if (ExCvt) { /* In SBCS cfg */
              wc = ff_uni2oem(wc, CODEPAGE);   /* Unicode ==> ANSI/OEM code */
              if (wc & 0x80) wc = ExCvt[wc & 0x7F]; /* Convert extended character to upper (SBCS) */
             } else {  /* In DBCS cfg */
              wc = ff_uni2oem(ff_wtoupper(wc), CODEPAGE); /* Unicode ==> Up-convert ==> ANSI/OEM code */
             }
          #elif FF_CODE_PAGE < 900 /* In SBCS cfg */
             wc = f_uni2oem(wc, CODEPAGE);   /* Unicode ==> ANSI/OEM code */
             if (wc & 0x80) wc = ExCvt[wc & 0x7F]; /* Convert extended character to upper (SBCS) */
          #else      /* In DBCS cfg */
             wc = ff_uni2oem(ff_wtoupper(wc), CODEPAGE); /* Unicode ==> Up-convert ==> ANSI/OEM code */
          #endif
            }

            if (wc >= 0x100) {    /* Is this a DBC? */
             if (i >= ni - 1) {   /* Field overflow? */
              cf |= NS_LOSS | NS_LFN;
              i = ni; continue;  /* Next field */
             }
             dp->fn[i++] = (BYTE)(wc >> 8); /* Put 1st byte */
            } else {      /* SBC */
             if (wc == 0 || strchr("+,;=[]", (int)wc)) { /* Replace illegal characters for SFN */
              wc = '_'; cf |= NS_LOSS | NS_LFN;/* Lossy conversion */
             } else {
              if (std::isupper(wc)) {  /* ASCII upper case? */
               b |= 2;
              }
              if (std::islower(wc)) {  /* ASCII lower case? */
               b |= 1; wc -= 0x20;
              }
             }
            }
            dp->fn[i++] = (BYTE)wc;
           }

           if (dp->fn[0] == DDEM) dp->fn[0] = RDDEM; /* If the first character collides with DDEM, replace it with RDDEM */

           if (ni == 8) b <<= 2;    /* Shift capital flags if no extension */
           if ((b & 0x0C) == 0x0C || (b & 0x03) == 0x03) cf |= NS_LFN; /* LFN entry needs to be created if composite capitals */
           if (!(cf & NS_LFN)) {    /* When LFN is in 8.3 format without extended character, NT flags are created */
            if (b & 0x01) cf |= NS_EXT;  /* NT flag (Extension has small capital letters only) */
            if (b & 0x04) cf |= NS_BODY; /* NT flag (Body has small capital letters only) */
           }

           dp->fn[NSFLAG] = cf; /* SFN is created into dp->fn[] */
```

```
 return FR_OK;


#else /* FF_USE_LFN : Non-LFN configuration */
 BYTE c, d, *sfn;
 UINT ni, si, i;
 const char *p;

 /* Create file name in directory form */
 p = *path; sfn = dp->fn;
 std::memset(sfn, ' ', 11);
 si = i = 0; ni = 8;
#if FF_FS_RPATH != 0
 if (p[si] == '.') { /* Is this a dot entry? */
  for (;;) {
   c = (BYTE)p[si++];
   if (c != '.' || si >= 3) break;
   sfn[i++] = c;

  }
  if (!IsSeparator(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 the path */
  return FR_OK;
 }
#endif
 for (;;) {
  c = (BYTE)p[si++];    /* Get a byte */
  if (c <= ' ') break;    /* Break if end of the path name */
  if (IsSeparator(c)) {   /* Break if a separator is found */
   while (IsSeparator(p[si])) si++; /* Skip duplicated separator if exist */
   break;
  }
  if (c == '.' || i >= ni) {  /* End of body or field overflow? */
   if (ni == 11 || c != '.') return FR_INVALID_NAME; /* Field overflow or invalid dot? */
   i = 8; ni = 11;    /* Enter file extension field */
   continue;
  }
#if FF_CODE_PAGE == 0
  if (ExCvt && c >= 0x80) {  /* Is SBC extended character? */
   c = ExCvt[c & 0x7F];  /* To upper SBC extended character */
  }
#elif FF_CODE_PAGE < 900
  if (c >= 0x80) {    /* Is SBC extended character? */
   c = ExCvt[c & 0x7F];  /* To upper SBC extended character */
  }
#endif
  if (dbc_1st(c)) {    /* Check if it is a DBC 1st byte */
   d = (BYTE)p[si++];   /* Get 2nd byte */
   if (!dbc_2nd(d) || i >= ni - 1) return FR_INVALID_NAME; /* Reject invalid DBC */
   sfn[i++] = c;
   sfn[i++] = d;
  } else {      /* SBC */
   if (strchr("*+,:;<=>[]|\"\?\x7F", (int)c)) return FR_INVALID_NAME; /* Reject illegal chrs for SFN */
   if (std::islower(c)) c -= 0x20; /* To upper */
   sfn[i++] = c;
  }
 }
 *path = &p[si];      /* Return pointer to the next segment */
 if (i == 0) return FR_INVALID_NAME; /* Reject nul string */

 if (sfn[0] == DDEM) sfn[0] = RDDEM; /* If the first character collides with DDEM, replace it with RDDEM */
 sfn[NSFLAG] = (c <= ' ' || p[si] <= ' ') ? NS_LAST : 0; /* Set last segment flag if end of the path */

 return FR_OK;
#endif /* FF_USE_LFN */
}


/*-----------------------------------------------------------------------*/
/* Get logical drive number from path name                               */
/*-----------------------------------------------------------------------*/

int DriverCommon::get_ldnumber ( /* Returns logical drive number (-1:invalid drive number or null pointer) */
 const TCHAR** path  /* Pointer to pointer to the path name */
)
{
 const TCHAR *tp, *tt;
 TCHAR tc;
 int i;
 int vol = -1;
#if FF_STR_VOLUME_ID  /* Find string volume ID */
 const char *sp;
 char c;
#endif

 tt = tp = *path;
 if (!tp) return vol; /* Invalid path name? */
 do tc = *tt++; while (!IsTerminator(tc) && tc != ':'); /* Find a colon in the path */

 if (tc == ':') { /* DOS/Windows style volume ID? */
  i = FF_VOLUMES;
  if (std::isdigit(*tp) && tp + 2 == tt) { /* Is there a numeric volume ID + colon? */
   i = (int)*tp - '0'; /* Get the LD number */
  }
#if FF_STR_VOLUME_ID == 1 /* Arbitrary string is enabled */

  else {
   i = 0;
   do {
```

```
    sp = VolumeStr[i]; tp = *path; /* This string volume ID and path name */
    do { /* Compare the volume ID with path name */
     c = *sp++; tc = *tp++;
     if (std::islower(c)) c -= 0x20;
     if (std::islower(tc)) tc -= 0x20;
    } while (c && (TCHAR)c == tc);
   } while ((c || tp != tt) && ++i < FF_VOLUMES); /* Repeat for each id until pattern match */
  }
#endif
  if (i < FF_VOLUMES) { /* If a volume ID is found, get the drive number and strip it */
   vol = i;  /* Drive number */
   *path = tt;  /* Snip the drive prefix off */
  }
  return vol;
 }
#if FF_STR_VOLUME_ID == 2  /* Unix style volume ID is enabled */
 if (*tp == '/') {   /* Is there a volume ID? */
  while (*(tp + 1) == '/') tp++; /* Skip duplicated separator */
  i = 0;
  do {
   tt = tp; sp = VolumeStr[i]; /* Path name and this string volume ID */
   do { /* Compare the volume ID with path name */
    c = *sp++; tc = *(++tt);
    if (std::islower(c)) c -= 0x20;
    if (std::islower(tc)) tc -= 0x20;
   } while (c && (TCHAR)c == tc);
  } while ((c || (tc != '/' && !IsTerminator(tc))) && ++i < FF_VOLUMES); /* Repeat for each ID until pattern match */
  if (i < FF_VOLUMES) { /* If a volume ID is found, get the drive number and strip it */
   vol = i;  /* Drive number */
   *path = tt;  /* Snip the drive prefix off */
  }
  return vol;
 }
#endif
 /* No drive prefix is found */
#if FF_FS_RPATH != 0
 vol = CurrVol; /* Default drive is current drive */
#else
 vol = 0;  /* Default drive is 0 */
#endif
 return vol;  /* Return the default drive */
}


/*-----------------------------------------------------------------------*/
/* GPT support functions                                                 */
/*-----------------------------------------------------------------------*/

#if FF_LBA64

/* Calculate CRC32 in byte-by-byte */

DWORD DriverCommon::crc32 ( /* Returns next CRC value */
 DWORD crc,   /* Current CRC value */
 BYTE d    /* A byte to be processed */
)
{
 BYTE b;


 for (b = 1; b; b <<= 1) {
  crc ^= (d & b) ? 1 : 0;
  crc = (crc & 1) ? crc >> 1 ^ 0xEDB88320 : crc >> 1;
 }
 return crc;
}


/* Check validity of GPT header */

int DriverCommon::test_gpt_header ( /* 0:Invalid, 1:Valid */
 const BYTE* gpth   /* Pointer to the GPT header */
)
{

 UINT i;
 DWORD bcc;


 if (memcmp(gpth + GPTH_Sign, "EFI PART" "\0\0\1\0" "\x5C\0\0", 16)) return 0; /* Check sign, version (1.0) and length (92) */
 for (i = 0, bcc = 0xFFFFFFFF; i < 92; i++) {  /* Check header BCC */
  bcc = crc32(bcc, i - GPTH_Bcc < 4 ? 0 : gpth[i]);
 }
 if (~bcc != ld_dword(gpth + GPTH_Bcc)) return 0;
 if (ld_dword(gpth + GPTH_PteSize) != SZ_GPTE) return 0; /* Table entry size (must be SZ_GPTE bytes) */
 if (ld_dword(gpth + GPTH_PtNum) > 128) return 0; /* Table size (must be 128 entries or less) */

 return 1;
}

#if !FF_FS_READONLY && FF_USE_MKFS

/* Generate random value */
DWORD DriverCommon::make_rand (
 DWORD seed,  /* Seed value */
 BYTE* buff,  /* Output buffer */
 UINT n   /* Data length */
)
{
 UINT r;


```

```
1404            if (seed == 0) seed = 1;
1405            do {
1406              for (r = 0; r < 8; r++) seed = seed & 1 ? seed >> 1 ^ 0xA3000000 : seed >> 1; /* Shift 8 bits the 32-bit LFSR */
1407              *buff++ = (BYTE)seed;
1408            } while (--n);
1409            return seed;
1410          }
1411
1412          #endif
1413          #endif
1414
1415          } // namespace fatfs
```