```c
/* ************************************************************************ */
/** Descriptive File Name

  @Company
    ETML-ES

  @File Name
    sd_fat_gest.c

  @Summary
    SD card fat system management

  @Description
    SD card fat system management
 */
/* ************************************************************************ */

/* ************************************************************************ */
/* ************************************************************************ */
/* Section: Included Files                                                  */
/* ************************************************************************ */
/* ************************************************************************ */

/* This section lists the other files that are included in this file.
 */

#include "Mc32_sdFatGest.h"
#include <stdio.h>
#include "app.h"
#include "bno055_support.h"
#include "GNSS/u_gnss_pos.h"
#include <stdio.h>
#include "usart_FIFO.h"
#include "MC32_serComm.h"

/* ************************************************************************ */
/* ************************************************************************ */
/* Section: File Scope or Global Data                                       */
/* ************************************************************************ */
/* ************************************************************************ */

APP_FAT_DATA COHERENT_ALIGNED appFatData;
/* ************************************************************************ */
/** Descriptive Data Item Name

  @Summary
    Brief one-line summary of the data item.

  @Description
    Full description, explaining the purpose and usage of data item.
    <p>
    Additional description in consecutive paragraphs separated by HTML
    paragraph breaks, as necessary.
    <p>
    Type "JavaDoc" in the "How Do I?" IDE toolbar for more information on tags.

  @Remarks
    Any additional remarks
 */


/* ************************************************************************ */
/* ************************************************************************ */
// Section: Local Functions                                                 */
/* ************************************************************************ */
/* ************************************************************************ */
// Function prototype
static uint8_t parseConfig(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
unsigned long *tInactive);
/* ************************************************************************ */

/* ************************************************************************ */
/* ************************************************************************ */
```

```
73  // Section: Interface Functions                                              */
74  /* ************************************************************************** */
75  /* ************************************************************************** */
76
77  void sd_fat_config_task ( bool init )
78  {
79      /* The application task cfg_state machine */
80      switch(appFatData.cfg_state)
81      {
82          case APP_CFG_MOUNT_DISK:
83              if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != 0)
84              {
85                  /* The disk could not be mounted. Try
86                   * mounting again untill success. */
87                  LED_ROn();
88                  appFatData.cfg_state = APP_CFG_MOUNT_DISK;
89              }
90              else
91              {
92                  /* Mount was successful. Unmount the disk, for testing. */
93                  LED_ROff();
94                  appFatData.cfg_state = APP_CFG_SET_CURRENT_DRIVE;
95              }
96              break;
97
98          case APP_CFG_SET_CURRENT_DRIVE:
99              if(SYS_FS_CurrentDriveSet("/mnt/myDrive") == SYS_FS_RES_FAILURE)
100             {
101                 /* Error while setting current drive */
102                 appFatData.cfg_state = APP_CFG_ERROR;
103             }
104             else
105             {
106                 if(init == true)
107                     /* Open config file for reading. */
108                     appFatData.cfg_state = APP_CFG_OPEN_READ_CONFIG_FILE;
109                 else
110                     /* Wait for further commands. */
111                     appFatData.cfg_state = APP_CFG_IDLE;
112             }
113             break;
114
115         case APP_CFG_OPEN_READ_CONFIG_FILE:
116             appFatData.fileCfgHandle = SYS_FS_FileOpen("CONFIG.txt",
117                     (SYS_FS_FILE_OPEN_READ));
118             if(appFatData.fileCfgHandle == SYS_FS_HANDLE_INVALID)
119             {
120                 /* No config file, write default config file */
121                 sd_CFG_Write(T_INTERVAL_GNSS_DEFAULT, T_INTERVAL_IMU_DEFAULT,
                    LED_STATE_DEFAULT, T_INACTIVE_PERIOD_DEFAULT, true);
122
123                 /* Re-try to open file as read */
124                 //appFatData.cfg_state = APP_CFG_OPEN_READ_CONFIG_FILE;
125
126             }
127             else
128             {
129                 /* Create a directory. */
130                 appFatData.cfg_state = APP_CFG_READ_CONFIG_FILE;
131             }
132             break;
133
134         case APP_CFG_READ_CONFIG_FILE:
135             /* If read was success, try writing to the new file */
136             if(SYS_FS_FileRead(appFatData.fileCfgHandle, appFatData.cfg_data,
137                     SYS_FS_FileSize(appFatData.fileCfgHandle)) == -1)
138             {
139                 /* Write was not successful. Close the file
140                  * and error out.*/
141                 SYS_FS_FileClose(appFatData.fileCfgHandle);
142                 appFatData.cfg_state = APP_CFG_ERROR;
143             }
144             else
```

```
145             {
146                 appFatData.cfg_state = APP_CFG_CLOSE_FILE;
147             }
148             break;
149         case APP_CFG_OPEN_WRITE_CONFIG_FILE:
150             appFatData.fileCfgHandle = SYS_FS_FileOpen("CONFIG.txt",
151                     (SYS_FS_FILE_OPEN_WRITE));
152             if(appFatData.fileCfgHandle == SYS_FS_HANDLE_INVALID)
153             {
154                 /* Could not open the file. Error out*/
155                 appFatData.cfg_state = APP_CFG_ERROR;
156             }
157             else
158             {
159                 /* Create a directory. */
160                 appFatData.cfg_state = APP_CFG_WRITE_CONFIG_FILE;
161             }
162             break;
163 
164         case APP_CFG_WRITE_CONFIG_FILE:
165             /* If read was success, try writing to the new file */
166             if(SYS_FS_FileStringPut(appFatData.fileCfgHandle, appFatData.cfg_data)== -
                1)
167             {
168                 /* Write was not successful. Close the file
169                  * and error out.*/
170                 SYS_FS_FileClose(appFatData.fileCfgHandle);
171                 appFatData.cfg_state = APP_CFG_ERROR;
172             }
173             else
174             {
175                 appFatData.cfg_state = APP_CFG_CLOSE_FILE;
176             }
177             break;
178         case APP_CFG_CLOSE_FILE:
179             /* Close the file */
180             SYS_FS_FileClose(appFatData.fileCfgHandle);
181              /* The test was successful. Lets idle. */
182             if(init == true)
183                 appFatData.cfg_state = APP_CFG_UNMOUNT_DISK;
184             else
185                 appFatData.cfg_state = APP_CFG_IDLE;
186             break;
187 
188         case APP_CFG_IDLE:
189             /* The appliction comes here when the demo
190              * has completed successfully. Switch on
191              * green LED. */
192             //BSP_LEDOn(APP_SUCCESS_LED);
193             LED_ROff();
194             break;
195         case APP_CFG_ERROR:
196             /* The appliction comes here when the demo
197              * has failed. Switch on the red LED.*/
198             //BSP_LEDOn(APP_FAILURE_LED);
199             LED_ROn();
200             break;
201         default:
202             break;
203 
204         case APP_CFG_UNMOUNT_DISK:
205             if(SYS_FS_Unmount("/mnt/myDrive") != 0)
206             {
207                 /* The disk could not be un mounted. Try
208                  * un mounting again untill success. */
209 
210                 appFatData.cfg_state = APP_CFG_UNMOUNT_DISK;
211             }
212             else
213             {
214                 /* UnMount was successful. Mount the disk again */
215                 appFatData.cfg_state = APP_CFG_IDLE;
216             }
```

```
217                 break;
218         }
219     }
220
221     // Loggin task
222     void sd_fat_logging_task ( void )
223     {
224         /* The application task log_state machine */
225         switch(appFatData.log_state)
226         {
227             case APP_MOUNT_DISK:
228                 if(SYS_FS_Mount("/dev/mmcblka1", "/mnt/myDrive", FAT, 0, NULL) != 0)
229                 {
230                     /* The disk could not be mounted. Try
231                      * mounting again untill success. */
232
233                     appFatData.log_state = APP_MOUNT_DISK;
234                 }
235                 else
236                 {
237                     /* Mount was successful. Unmount the disk, for testing. */
238
239                     appFatData.log_state = APP_SET_CURRENT_DRIVE;
240                 }
241                 break;
242
243             case APP_SET_CURRENT_DRIVE:
244                 if(SYS_FS_CurrentDriveSet("/mnt/myDrive") == SYS_FS_RES_FAILURE)
245                 {
246                     /* Error while setting current drive */
247                     appFatData.log_state = APP_ERROR;
248                 }
249                 else
250                 {
251                     /* Open a file for reading. */
252                     appFatData.log_state = APP_IDLE;
253                 }
254                 break;
255
256             case APP_WRITE_MEASURE_FILE:
257                 appFatData.fileMeasureHandle = SYS_FS_FileOpen(appFatData.fileName,
258                         (SYS_FS_FILE_OPEN_APPEND_PLUS));
259                 if(appFatData.fileMeasureHandle == SYS_FS_HANDLE_INVALID)
260                 {
261                     /* Could not open the file. Error out*/
262                     appFatData.log_state = APP_ERROR;
263                 }
264                 else
265                 {
266                     /* Create a directory. */
267                     appFatData.log_state = APP_WRITE_TO_MEASURE_FILE;
268                 }
269                 break;
270
271             case APP_WRITE_TO_MEASURE_FILE:
272                 /* If read was success, try writing to the new file */
273                 if(SYS_FS_FileStringPut(appFatData.fileMeasureHandle, appFatData.data) ==
                    -1)
274                 {
275                     /* Write was not successful. Close the file
276                      * and error out.*/
277                     SYS_FS_FileClose(appFatData.fileMeasureHandle);
278                     appFatData.log_state = APP_ERROR;
279                 }
280                 else
281                 {
282                     appFatData.log_state = APP_CLOSE_FILE;
283                 }
284                 break;
285
286             case APP_CLOSE_FILE:
287                 /* Close both files */
288                 SYS_FS_FileClose(appFatData.fileMeasureHandle);
```

```
289                 /* The test was successful. Lets idle. */
290                 appFatData.log_state = APP_IDLE;
291                 break;
292 
293             case APP_IDLE:
294                 /* The appliction comes here when the demo
295                  * has completed successfully. Switch on
296                  * green LED. */
297                 //BSP_LEDOn(APP_SUCCESS_LED);
298                 LED_ROff();
299                 break;
300             case APP_ERROR:
301                 /* The appliction comes here when the demo
302                  * has failed. Switch on the red LED.*/
303                 //BSP_LEDOn(APP_FAILURE_LED);
304                 LED_ROn();
305                 break;
306             default:
307                 break;
308 
309             case APP_UNMOUNT_DISK:
310                 if(SYS_FS_Unmount("/mnt/myDrive") != 0)
311                 {
312                     /* The disk could not be un mounted. Try
313                      * un mounting again untill success. */
314 
315                     appFatData.log_state = APP_UNMOUNT_DISK;
316                 }
317                 else
318                 {
319                     /* UnMount was successful. Mount the disk again */
320                     appFatData.log_state = APP_IDLE;
321                 }
322                 break;
323 
324         }
325 
326     //    SYS_FS_Tasks();
327     } //End of APP_Tasks
328 
329     void sd_IMU_scheduleWrite (s_bno055_data * data)
330     {
331         /* If sd Card available */
332         if(appFatData.log_state == APP_IDLE)
333         {
334             // Prepare file name
335             sprintf(appFatData.fileName, "LOG_IMU.csv");
336             /* Next log_state : write to file */
337             appFatData.log_state = APP_WRITE_MEASURE_FILE;
338             /* Write the buffer */
339             sprintf(appFatData.data,
                    "%d;%d0;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;%.4f;
                    %.4f;%d;%d;%d;%d;\r\n"
340                                     ,data->flagImportantMeas, (data->d_time), data->
                                        gravity.x, data->gravity.y, data->gravity.z, data->
                                        gyro.x, data->gyro.y, data->gyro.z
341                                     ,data->mag.x, data->mag.y, data->mag.z, data->
                                        linear_accel.x, data->linear_accel.y, data->
                                        linear_accel.z
342                                     ,data->euler.h, data->euler.p, data->euler.r, data->
                                        quaternion.w, data->quaternion.x, data->quaternion.y
                                        , data->quaternion.z);
343             /* Compute the number of bytes to send */
344             appFatData.nBytesToWrite = strlen(appFatData.data);
345         }
346     }
347 
348     void sd_GNSS_scheduleWrite (minmea_messages * pGnssData)
349     {
350         char fifoBuffer[FIFO_RX_SIZE];
351         /* If sd Card available */
352         if(appFatData.log_state == APP_IDLE)
353         {
```

```
354             // Prepare file name
355             sprintf(appFatData.fileName, "LOG_GNSS.txt");
356             /* Next log_state : write to file */
357             appFatData.log_state = APP_WRITE_MEASURE_FILE;
358             /* Write the buffer */
359             getFifoToLastReturn(&usartFifoRx, fifoBuffer);
360 
361             sprintf(appFatData.data, "%s", fifoBuffer);
362 
363             //sprintf(appFatData.data, "%s",  );
364             /* Compute the number of bytes to send */
365             appFatData.nBytesToWrite = strlen(appFatData.data);
366         }
367     }
368 
369     void sd_CFG_Write (uint32_t tLogGNSS_ms, uint32_t tLogIMU_ms, uint8_t ledState,
    uint32_t tInactiveP, bool skipMount)
370     {
371         /* If sd Card available */
372         if((appFatData.cfg_state == APP_CFG_IDLE)||(appFatData.cfg_state ==
        APP_CFG_OPEN_READ_CONFIG_FILE))
373         {
374             /* Close the file */
375             SYS_FS_FileClose(appFatData.fileCfgHandle);
376 
377             if(skipMount == false)
378                 /* Next config : mount disk */
379                 appFatData.cfg_state = APP_CFG_MOUNT_DISK;
380             else if(skipMount == true)
381                 /* Next config : write to file */
382                 appFatData.cfg_state = APP_CFG_OPEN_WRITE_CONFIG_FILE;
383 
384             /* Write the buffer */
385             sprintf(appFatData.cfg_data, "$LOG INTERVAL GNSS [ms] : %u\r\n$LOG INTERVAL
            IMU [ms] : %u\r\n$LED ENABLE [1/0] : %u\r\n$INACTIVE PERIOD [s] : %u\r\n",
386                     tLogGNSS_ms, tLogIMU_ms, ledState, tInactiveP);
387             /* Compute the number of bytes to send */
388             appFatData.nBytesToWrite = strlen(appFatData.cfg_data);
389 
390         }
391     }
392 
393 
394     APP_FAT_LOG_STATES sd_logGetState( void )
395     {
396         return appFatData.log_state;
397     }
398 
399     void sd_logSetState( APP_FAT_LOG_STATES newState )
400     {
401         appFatData.log_state = newState;
402     }
403 
404     // CONFIG FUNCTIONS
405 
406     APP_FAT_CONFIG_STATES sd_cfgGetState( void )
407     {
408         return appFatData.cfg_state;
409     }
410     void sd_cfgSetState( APP_FAT_CONFIG_STATES newState )
411     {
412          appFatData.cfg_state = newState;
413     }
414 
415     char* sd_cfgGetCfgBuffer( void )
416     {
417         return appFatData.cfg_data;
418     }
419 
420     void sd_fat_cfg_init(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
    uint32_t *tInactivePeriod)
421     {
422         // Config parser error
```

```
423         uint8_t parseError = 0;
424         unsigned long tGnssLocal = 0;
425         unsigned long tImuLocal = 0;
426         unsigned long tInactive = 0;
427         bool ledStateLocal = 0;
428 
429         //appFatData.nBytesRead = 0;
430         //appFatData.nBytesToWrite = 0;
431 
432         //appFatData.log_state = APP_MOUNT_DISK;
433         //appFatData.cfg_state = APP_CFG_MOUNT_DISK;
434 
435         // Read config routine, until error or success
436         sd_fat_config_task(true);
437 
438         // If read config routine was a success
439         if(sd_cfgGetState() == APP_CFG_IDLE)
440             // Parse config buffer to get parameters
441             parseError = parseConfig(&tGnssLocal, &tImuLocal, &ledStateLocal, &tInactive);
442         // If the parsing failed or the read config routine failed
443         if((parseError > 0)||(sd_cfgGetState() == APP_CFG_ERROR))
444         {
445             // Set default system parameters
446             *tGnss = T_INTERVAL_GNSS_DEFAULT;
447             *tImu = T_INTERVAL_IMU_DEFAULT;
448             *ledState = LED_STATE_DEFAULT;
449             *tInactivePeriod = T_INACTIVE_PERIOD_DEFAULT;
450             appStateSet(APP_STATE_LOGGING);
451             // Start measure timer
452             DRV_TMR1_Start();
453         }
454         else if ((sd_cfgGetState() == APP_CFG_IDLE))
455         {
456             *tGnss = tGnssLocal;
457             *tImu = tImuLocal;
458             *ledState = ledStateLocal;
459             *tInactivePeriod = tInactive;
460             appStateSet(APP_STATE_LOGGING);
461             // Start measure timer
462             DRV_TMR1_Start();
463         }
464     }
465 
466     static uint8_t parseConfig(unsigned long *tGnss, unsigned long *tImu, bool *ledState,
    unsigned long *tInactive)
467     {
468         char *ptBufferHead;
469         char *ptBufferTail;
470         char ptTrame[10];
471         uint8_t error = 0;
472 
473         // Locate the head and tail of the first data
474         ptBufferHead = strstr(appFatData.cfg_data, " :");
475         ptBufferTail = strstr(appFatData.cfg_data, "\r\n");
476         // Check if the pointers are corrects
477         if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
478             // Copy the data between the head and the tail in a sub-pointer
479             strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
480             // Convert the character to value
481             *tGnss = (uint32_t) atoi(ptTrame);
482         }
483         else
484             error++;
485 
486         // Locate the head and tail of the first data
487         ptBufferHead = strstr(ptBufferTail, " :");
488         ptBufferTail = strstr(ptBufferHead, "\r\n");
489         // Check if the pointers are corrects
490         if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
491             // Copy the data between the head and the tail in a sub-pointer
492             strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
493             // Convert the character to value
494             *tImu = (uint32_t) atoi(ptTrame);
```

```
    }
    else
        error++;

    // Locate the head and tail of the first data
    ptBufferHead = strstr(ptBufferTail, " :");
    ptBufferTail = strstr(ptBufferHead, "\r\n");
    // Check if the pointers are corrects
    if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
        // Copy the data between the head and the tail in a sub-pointer
        strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
        // Convert the character to value
        *ledState = (bool) atoi(ptTrame);
    }
    else
        error++;

    // Locate the head and tail of the first data
    ptBufferHead = strstr(ptBufferTail, " :");
    ptBufferTail = strstr(ptBufferHead, "\r\n");
    // Check if the pointers are corrects
    if((ptBufferHead != NULL)&&(ptBufferTail != NULL)&&(ptBufferHead < ptBufferTail)){
        // Copy the data between the head and the tail in a sub-pointer
        strncpy(ptTrame, (ptBufferHead+2), (ptBufferTail-ptBufferHead));
        // Convert the character to value
        *tInactive = (uint32_t) atoi(ptTrame);
    }
    else
        error++;

    return error;

}

void sd_fat_readDisplayFile(const char * fileName)
{
    const uint16_t READSIZE = 256;
    uint32_t i = 0;
    char stringRead[READSIZE];
    unsigned long cntTimeaout = 0;

    /* Close both files */
    SYS_FS_FileClose(appFatData.fileMeasureHandle);
    // Read config file
    appFatData.fileCfgHandle = SYS_FS_FileOpen(fileName,(SYS_FS_FILE_OPEN_READ));

    do{

        SYS_FS_FileStringGet(appFatData.fileCfgHandle, stringRead, READSIZE);

        do{
            if(!PLIB_USART_TransmitterBufferIsFull(USART_ID_1))
            {
                PLIB_USART_TransmitterByteSend(USART_ID_1, stringRead[i]);
                i++;
            }
            cntTimeaout++;
        }while((i < strlen(stringRead))&&(cntTimeaout<TIME_OUT));

        i = 0;
        cntTimeaout = 0;

        if(pollSerialCmds(USART_ID_1, "exit", "EXIT", "x" ,"X"))
            break;

    }while(!SYS_FS_FileEOF(appFatData.fileCfgHandle));

    /* Close both files */
    SYS_FS_FileClose(appFatData.fileMeasureHandle);
}

//bool sd_fat_readFile(const char * fileName, char readBuffer[])
//{
```

```
568   //    uint32_t fileSize = 0;
569   //    static bool fullyRead = false;
570   //    /* Close both files */
571   //    SYS_FS_FileClose(appFatData.fileMeasureHandle);
572   //    // Read config file
573   //    appFatData.fileCfgHandle = SYS_FS_FileOpen(fileName,(SYS_FS_FILE_OPEN_READ));
574   //
575   //    fileSize = SYS_FS_FileSize(appFatData.fileCfgHandle);
576   //
577   //    if (fileSize <= sizeof(readBuffer))
578   //        SYS_FS_FileRead(appFatData.fileCfgHandle, readBuffer, fileSize);
579   //    else{
580   //
581   //    }
582   //    /* Close both files */
583   //    SYS_FS_FileClose(appFatData.fileMeasureHandle);
584   //}
585
586   /* *****************************************************************************
587    End of File
588    */
589
```