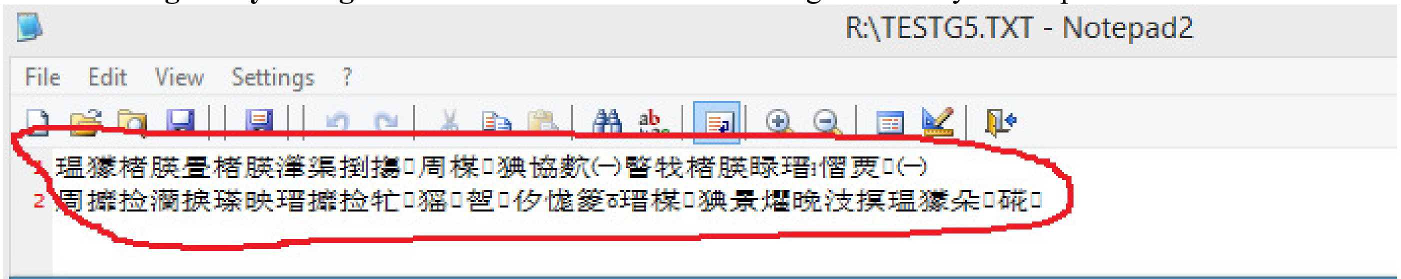
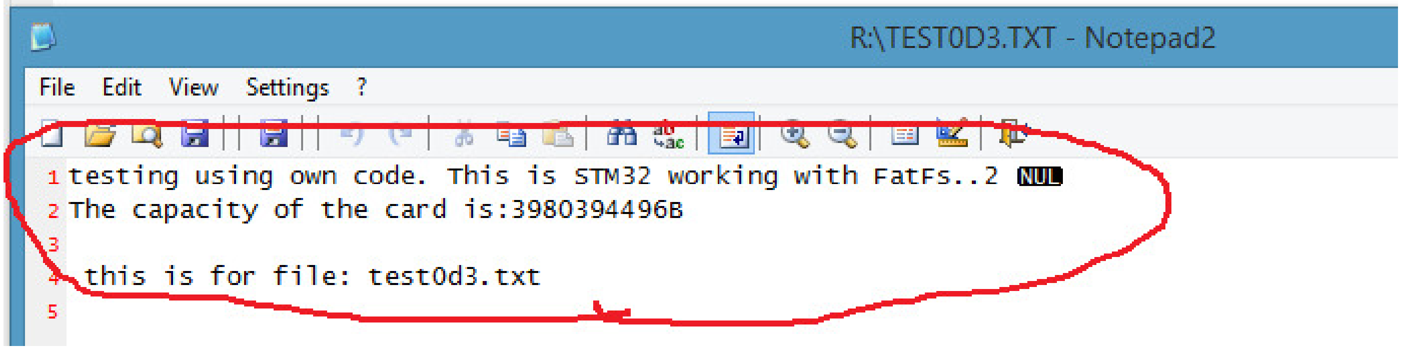
I developed a board using STM32F411 chip with uSD card.The code is generated by cubemx. only add a little bit code in main(). The hardware is OK. I can correctly write a file to sd card. But the weird thing is that the filename accepts 7 chars, not 6 chars. If I use 6 chars in the filename, the character encoding is totally different.

In the testing I only change the filename. I attached two files generated by the chip.



```
瑥獴楍畬畧楍畬瀁渠搠搗□周楳□獭協歁(一)瞥牧楍畬睬瑨楒瞴漢□(一)
周擶捡灁掾瑹映瑨擶捡牤□猫□㐲□伤慥簽ɞ瑨楳□獭景爤晚汥摸瑥獴朵□硫□
```

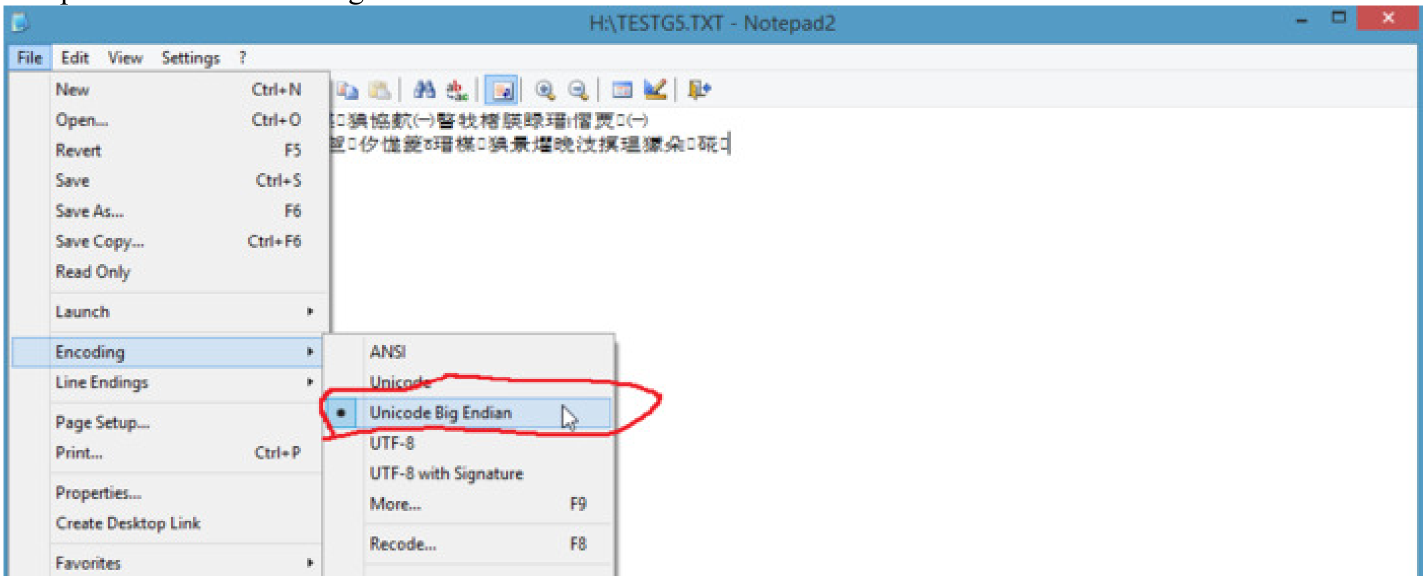


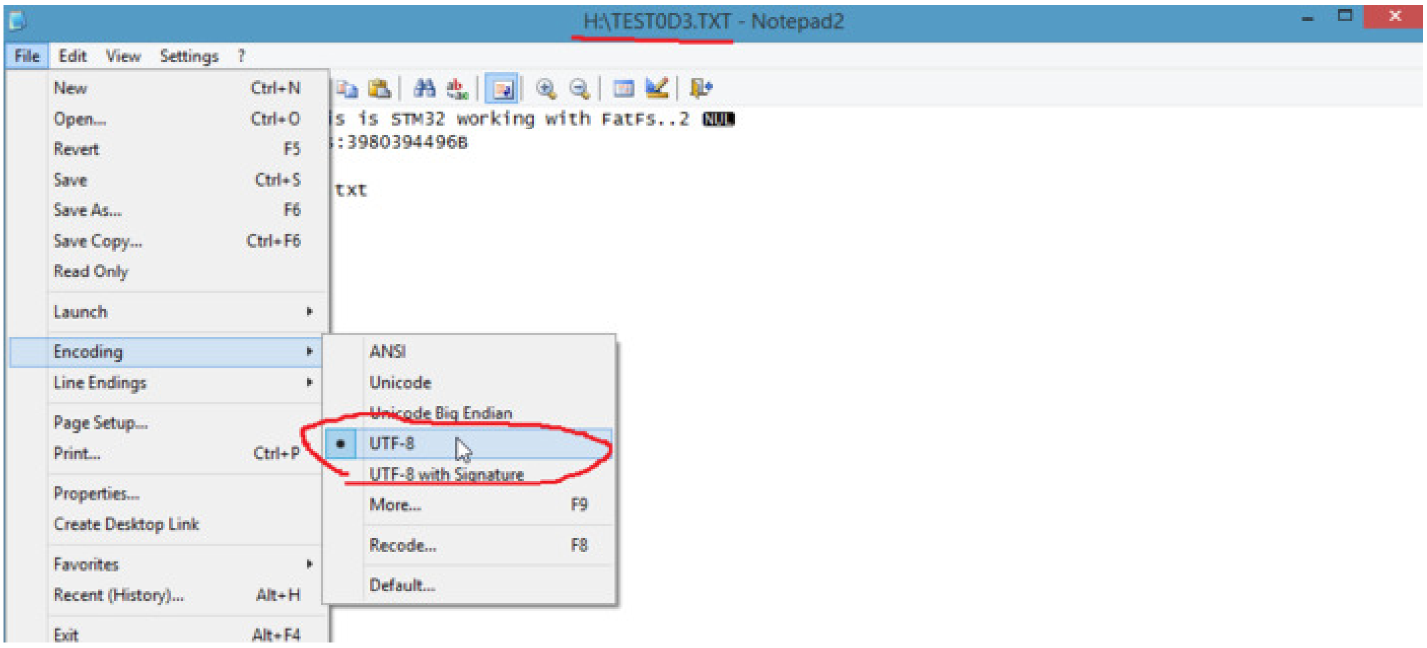
```
testing using own code. This is STM32 working with FatFs..2 NUL
The capacity of the card is:3980394496B

 this is for file: test0d3.txt
```

I can't read the first the characters in the first file.

The problem is the encoding of the files are different.





```
is is STM32 working with FatFs..2 NUL
:3980394496B

txt
```

the thing is why the encoding are different. I didn't change anything. Actually I use ANSI in the setting.

My question is
1. **why the program uses two different encodings** for the files just because I use different filenames? The file with 7 chars in filename always uses utf-8 encoding.
2. how to correctly set the encoding. Actually in the file I specify it to use ANSI encoding, as in ffconf.h. Why the program uses others?
Please help.
The two txt files and the ffconf.h are enclosed as attachment.

```
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_SDIO_SD_Init();
  MX_TIM9_Init();
//  MX_FATFS_Init();
  MX_USB_DEVICE_Init();



  /* USER CODE BEGIN 2 */
  /* USER CODE BEGIN WHILE */
      FRESULT res;                                          /* FatFs function common result code */
  uint32_t byteswritten, bytesread,byteswrittentotal;                     /* File write/read counts
*/
```

```
  uint8_t wtext[] = "testing using own code. This is STM32 working with FatFs..2 "; /* File write
buffer . note it will produce a null symbol at the line end*/
  uint8_t rtext[100];
    char textbuf[200]; //string buffer
    char filename[]="testa12.txt"; //filename can be >8, depends on
setting                                  /* File read buffer */


  /*##-1- Link the micro SD disk I/O driver ##################################*/
  if(FATFS_LinkDriver(&SD_Driver, SD_Path) == 0) //SD_Path was defined in fatfs.c  by cubemx
  {
    /*##-2- Register the file system object to the FatFs module ##############*/
    if(f_mount(&SDFatFs, (TCHAR const*)SD_Path, 0) != FR_OK)
    {
      /* FatFs Initialization Error */
      Error_Handler();
    }
    else
    {

        /* ##-4- Create and Open a new text file object with write access #####*/
        if(f_open(&MyFile, filename, FA_CREATE_ALWAYS | FA_WRITE) != FR_OK)
        {
          /* 'STM32.TXT' file Open for write Error */
          Error_Handler();
        }
        else
        {
          /*##-5- Write data to the text file ################################*/
          res = f_write(&MyFile, wtext, sizeof(wtext), (void *)&byteswritten);
                    byteswrittentotal=byteswritten;
         //write capacity info as testing
                    BSP_SD_GetCardInfo(&uSdCardInfotmp);
                    sprintf(textbuf,"\nThe capacity of the card
is:%"PRIu64"B\n",uSdCardInfotmp.CardCapacity);//export it to char buffer first. In this way, no
null symbol
                    f_write(&MyFile, textbuf, strlen(textbuf), (void *)&byteswritten);
                    byteswrittentotal+=byteswritten;
                    sprintf(textbuf,"\n this is for file: %s \n",filename);//export it to char
buffer first. In this way, no null symbol
                    f_write(&MyFile, textbuf, strlen(textbuf), (void *)&byteswritten);
                    byteswrittentotal+=byteswritten;

          if((byteswritten == 0) || (res != FR_OK))
          {
            /* 'STM32.TXT' file Write or EOF Error */
            Error_Handler();
          }
          else
          {
            /*##-6- Close the open text file #################################*/
            f_close(&MyFile);

            /*##-7- Open the text file object with read access ###############*/
            if(f_open(&MyFile, filename, FA_READ) != FR_OK)
            {
              /* 'STM32.TXT' file Open for read Error */
              Error_Handler();
            }
            else
            {
              /*##-8- Read data from the text file ###########################*/
              res = f_read(&MyFile, rtext, sizeof(rtext), (UINT*)&bytesread);

              if((bytesread == 0) || (res != FR_OK))
              {
                /* 'STM32.TXT' file Read or EOF Error */
                Error_Handler();
              }
              else
              {
                /*##-9- Close the open text file ##############################*/
                f_close(&MyFile);

                /*##-10- Compare read data with the expected data ############*/
                if((bytesread != byteswritten))
                {
                  /* Read data is different from the expected data */
                  Error_Handler();
                }
                else
                {
                  /* Success of the demo: no error occurrence */
             //     BSP_LED_On(LED1);
                }
              }
            }
          }
        }

    }
  }

  /*##-11- Unlink the RAM disk I/O driver ####################################*/
  FATFS_UnLinkDriver(SD_Path);
```