

Song list code module

Collect MP3 song list from the SD card

Reference Articles

- [Design a code module](#)
- [Code Modularity](#)

Get a list of MP3 files (naive way)

The objective of this code is to get a list of *.mp3 files at the root directory of your SD card.

```
#include "ff.h"

void print_list_of_mp3_songs(void) {
    FRESULT result;
    FILINFO file_info;
    const char *root_path = "/";
    DIR dir;
    result = f_opendir(&dir, root_path);
    if (result == FR_OK) {
        while (1) {
            result = f_readdir(&dir, &file_info);
            const bool item_name_is_empty = (file_info.fname[0] == 0);
            if ((result != FR_OK) || item_name_is_empty) {
                break; /* Break on error or end of dir */
            }
            const bool is_directory = (file_info.fattrib & AM_DIR);
            if (is_directory) {
                /* Skip nested directories, only focus on MP3 songs at the root */
            } else { /* It is a file. */
                printf("Filename: %s\n", file_info.fname);
            }
        }
    }
}
```

```

    f_closedir(&dir);
}
}

```

Get list of MP3 songs

Here is a better way to design code such that a dedicated code module will obtain song-list for you:

```

// @file: song_list.h
#pragma once
#include <stddef.h> // size_t
typedef char song_memory_t[128];
/* Do not declare variables in a header file */
#ifdef 0
static song_memory_t list_of_songs[32];
static size_t number_of_songs;
#endif
void song_list__populate(void);
size_t song_list__get_item_count(void);
const char *song_list__get_name_for_item(size_t item_number);

```

```

#include <string.h>
#include "song_list.h"
#include "ff.h"

static song_memory_t list_of_songs[32];
static size_t number_of_songs;
static void song_list__handle_filename(const char *filename) {
    // This will not work for cases like "file.mp3.zip"
    if (NULL != strstr(filename, ".mp3")) {
        // printf("Filename: %s\n", filename);
        // Dangerous function: If filename is > 128 chars, then it will copy extra bytes leading to memory
        // strcpy(list_of_songs[number_of_songs], filename);
        // Better: But strncpy() does not guarantee to copy null char if max length encountered
        // So we can manually subtract 1 to reserve as NULL char
        strncpy(list_of_songs[number_of_songs], filename, sizeof(song_memory_t) - 1);
        // Best: Compensates for the null, so if 128 char filename, then it copies 127 chars, AND the NULL
    }
}

```

```

    // snprintf(list_of_songs[number_of_songs], sizeof(song_memory_t), "%.149s", filename);
    ++number_of_songs;
    // or
    // number_of_songs++;
}
}

void song_list__populate(void) {
    FRESULT res;
    static FILINFO file_info;
    const char *root_path = "/";
    DIR dir;
    res = f_opendir(&dir, root_path);
    if (res == FR_OK) {
        for (;;) {
            res = f_readdir(&dir, &file_info); /* Read a directory item */
            if (res != FR_OK || file_info.fname[0] == 0) {
                break; /* Break on error or end of dir */
            }
            if (file_info.fattrib & AM_DIR) {
                /* Skip nested directories, only focus on MP3 songs at the root */
            } else { /* It is a file. */
                song_list__handle_filename(file_info.fname);
            }
        }
        f_closedir(&dir);
    }
}

size_t song_list__get_item_count(void) { return number_of_songs; }

const char *song_list__get_name_for_item(size_t item_number) {
    const char *return_pointer = "";
    if (item_number >= number_of_songs) {
        return_pointer = "";
    } else {
        return_pointer = list_of_songs[item_number];
    }
    return return_pointer;
}

```

```
}
```

```
int main(void) {  
    song_list__populate();  
    for (size_t song_number = 0; song_number < song_list__get_item_count(); song_number++) {  
        printf("Song %2d: %s\n", (1 + song_number), song_list__get_name_for_item(song_number));  
    }  
}
```

Revision #6

Created 4 years ago by [Preet Kang](#)

Updated 2 years ago by [Preet Kang](#)