

```
1 #include "sdcardIO.h"
2 #include "../vars/constants.h"
3 #include "FS.h"
4 #include "SD.h"
5 #include "SPI.h"
6 #include <vector>
7 #include <tuple>
8
9 // @brief Mount the SDCard
10 // @return 0: If mounting was successful;
11 // @return 1: If communication with SDCard module failed;
12 // @return 2: If no card is attached
13 uint8_t sdcardIO::MountCard()
14 {
15     if(!SD.begin(33))
16     {
17         //SD-Card mount failed
18         return 1;
19     }
20     if(SD.cardType() == CARD_NONE)
21     {
22         //No card attached
23         return 2;
24     }
25     return 0;
26 }
27
28 // @brief Get Size of SDCard
29 uint64_t sdcardIO::GetCardSize()
30 {
31     return SD.cardSize();
32 }
33
34 // @brief Get Total Bytes of SDCard
35 uint64_t sdcardIO::GetTotalBytes()
36 {
37     return SD.totalBytes();
38 }
39
40 // @brief Get Used Bytes of SDCard
41 uint64_t sdcardIO::GetUsedBytes()
42 {
43     return SD.usedBytes();
44 }
45
46 // @brief Get Free Bytes of SDCard
47 uint64_t sdcardIO::GetFreeBytes()
48 {
49     return GetTotalBytes() - GetUsedBytes();
50 }
51
52 // @brief Get list of all Files in a Directory
53 // @param &fs The Filesystem
```

```

54 // @param *dirname The path to the directory
55 // @param levels How deep to search in subdirectories
56 // @param deep Internal use only, leave out default or set to zero
57 // @return A Vector of a Tuple of uint8_t (EntryType), uint8_t (How deep
58 //         the subdirectory is), string (path of entry), size_t (size of entry)
59 std::vector<std::tuple<uint8_t, uint8_t, std::string, size_t>>
60     sdcardIO::Directory::List(fs::FS &fs, const char * dirname, uint8_t
61     levels, uint8_t deep)
62 {
63     //EntryType, Level, Path, Size
64     std::vector<std::tuple<uint8_t, uint8_t, std::string, size_t>>
65         entryList;
66     fs::File root = fs.open(dirname);
67     if(!root){
68         entryList.push_back(std::make_tuple(DirectoryEntryType::UNKNOWN,
69         deep, ("Failed to open directory: " + std::string(dirname)).c_str
70         (), 0));
71     return entryList;
72     }
73     if(!root.isDirectory()){
74         entryList.push_back(std::make_tuple(DirectoryEntryType::UNKNOWN,
75         deep, "Not a directory", 0));
76     return entryList;
77     }
78     fs::File file = root.openNextFile();
79     while(file){
80         if(file.isDirectory()){
81             entryList.push_back(std::make_tuple
82             (DirectoryEntryType::Directory, deep, (std::string(file.path())
83             + "/").c_str(), file.size()));
84             if(levels){
85                 std::vector<std::tuple<uint8_t, uint8_t, std::string, size_t>>
86                     tmpList = sdcardIO::Directory::List(fs, file.path(), levels -
87                     1, deep + 1);
88                 entryList.insert(entryList.end(), tmpList.begin(), tmpList.end
89                 ());
90             }
91         }
92         else
93         {
94             entryList.push_back(std::make_tuple(DirectoryEntryType::File,
95             deep, file.path(), file.size()));
96         }
97         file = root.openNextFile();
98     }
99     return entryList;
100 }
101 // @brief Create Directory
102 // @param &fs The Filesystem

```

```
93 // @param path The path of the directory to create
94 // @return True if creation was successful, otherwise false
95 bool sdcardIO::Directory::Create(fs::FS &fs, const char * path)
96 {
97     if(fs.mkdir(path))
98     {
99         return true;
100     }
101     return false;
102 }
103
104 // @brief Delete Directory
105 // @param &fs The Filesystem
106 // @param path The path of the directory to remove
107 // @return True if deletion was successful, otherwise false
108 bool sdcardIO::Directory::Remove(fs::FS &fs, const char * path)
109 {
110     if(fs.rmdir(path))
111     {
112         return true;
113     }
114     return false;
115 }
116
117 // @brief Read file from SDCard and print to Serial Console
118 // @param &fs The Filesystem
119 // @param path The path of the file to read
120 void sdcardIO::File::Read(fs::FS &fs, const char * path)
121 {
122     fs::File file = fs.open(path);
123     if(!file)
124     {
125         Serial.println("Failed to open file for reading");
126         return;
127     }
128
129     Serial.print("Reading from file: ");
130     Serial.println(path);
131     while(file.available())
132     {
133         Serial.write(file.read());
134     }
135     file.close();
136     Serial.println();
137 }
138
139 // @brief Write something to a file
140 // @param &fs The Filesystem
141 // @param path The path of the file
142 // @param content The content to write to the file
143 // @return 0: If writing was successful;
144 // @return 1: If file failed to open for writing
145 // @return 2: If writing failed
```

```
146 uint8_t sdcardIO::File::Write(fs::FS &fs, const char * path, const char *  
    * content)  
147 {  
148     fs::File file = fs.open(path, FILE_WRITE);  
149     if(!file)  
150     {  
151         //Failed to open file for writing  
152         return 1;  
153     }  
154  
155     if(file.print(content))  
156     {  
157         file.close();  
158         return 0;  
159     }  
160     file.close();  
161     //Write Failed  
162     return 2;  
163 }  
164  
165 //@brief Append something to a file  
166 //@param &fs The Filesystem  
167 //@param path The path of the file  
168 //@param content The content to append to the file  
169 //@return 0: If appending was successful;  
170 //@return 1: If file failed to open for appending  
171 //@return 2: If appending failed  
172 uint8_t sdcardIO::File::Append(fs::FS &fs, const char * path, const  
    char * content)  
173 {  
174     fs::File file = fs.open(path, FILE_APPEND);  
175     if(!file)  
176     {  
177         //Failed to open for appending  
178         return 1;  
179     }  
180  
181     if(file.print(content))  
182     {  
183         file.close();  
184         return 0;  
185     }  
186  
187     file.close();  
188     //Append failed  
189     return 2;  
190 }  
191  
192 //@brief Rename a file  
193 //@param &fs The Filesystem  
194 //@param path1 The original path of the file  
195 //@param path2 The new path of the file  
196 //@return true if renaming was successful, otherwise return false
```

```
197 bool sdcardIO::File::Rename(fs::FS &fs, const char * path1, const char *  
    * path2)  
198 {  
199     if(fs.rename(path1, path2))  
200     {  
201         return true;  
202     }  
203     return false;  
204 }  
205  
206 //@brief Delete a file  
207 //@param &fs The Filesystem  
208 //@param path1 The path of the file  
209 //@return true if removing was successful, otherwise return false  
210 bool sdcardIO::File::Delete(fs::FS &fs, const char * path)  
211 {  
212     if(fs.remove(path))  
213     {  
214         return true;  
215     }  
216     return false;  
217 }  
218  
219 //@brief Read the constants from a file on the SD-Card  
220 //@param &fs The Filesystem  
221 //@return 0: If reading was successful  
222 //@return 1: If file does not exist  
223 //@return 2: If file failed to open  
224 uint8_t sdcardIO::ReadConstants(fs::FS &fs)  
225 {  
226     if(!fs.exists(constants::sdcardIO::ConfigFilePath))  
227     {  
228         //Config file does not exist  
229         return 1;  
230     }  
231  
232     fs::File file = fs.open(constants::sdcardIO::ConfigFilePath,  
        FILE_READ);  
233     if(!file)  
234     {  
235         //File failed to open  
236         return 2;  
237     }  
238     while(file.available())  
239     {  
240         String line = file.readStringUntil('\n');  
241  
242         std::string key = line.substring(0, line.indexOf('=')).c_str();  
243         std::string value = line.substring(line.indexOf('=') + 1).c_str()  
            ();  
244  
245         if(!constants::setValue(key.c_str(), value.c_str()))  
246         {
```

```
247         Serial.print("Error setting value: ");
248         Serial.println(key.c_str());
249     }
250 }
251 file.close();
252 return 0;
253 }
```