

DFR0534

1.0.4

Generated by Doxygen 1.12.0



<b>1 DFR0534</b>	<b>1</b>
1.1 SoftwareSerial for Arduino Uno/Nano/ATmega328p	1
1.2 HardwareSerial for ESP32	1
1.3 License and copyright	2
1.4 Appendix	3
1.4.1 DFR0534 pinout	3
<b>2 Class Index</b>	<b>5</b>
2.1 Class List	5
<b>3 File Index</b>	<b>7</b>
3.1 File List	7
<b>4 Class Documentation</b>	<b>9</b>
4.1 DFR0534 Class Reference	9
4.1.1 Detailed Description	11
4.1.2 Member Enumeration Documentation	11
4.1.2.1 DFR0534CHANNELS	11
4.1.2.2 DFR0534DRIVE	11
4.1.2.3 DFR0534EQ	12
4.1.2.4 DFR0534LOOPMODE	12
4.1.2.5 DFR0534STATUS	13
4.1.3 Constructor & Destructor Documentation	13
4.1.3.1 DFR0534()	13
4.1.4 Member Function Documentation	14
4.1.4.1 decreaseVolume()	14
4.1.4.2 fastBackwardDuration()	14
4.1.4.3 fastForwardDuration()	14
4.1.4.4 getDrive()	14
4.1.4.5 getDrivesStates()	15
4.1.4.6 getDuration()	17
4.1.4.7 getFileName()	18
4.1.4.8 getFileNumber()	19
4.1.4.9 getFirstFileNumberInCurrentDirectory()	20
4.1.4.10 getRuntime()	21
4.1.4.11 getStatus()	22
4.1.4.12 getTotalFiles()	23
4.1.4.13 getTotalFilesInCurrentDirectory()	24
4.1.4.14 increaseVolume()	25
4.1.4.15 insertFileByNumber()	26
4.1.4.16 pause()	26
4.1.4.17 play()	26
4.1.4.18 playCombined()	26

4.1.4.19 playFileByName()	27
4.1.4.20 playFileByNumber()	28
4.1.4.21 playLastInDirectory()	28
4.1.4.22 playNext()	29
4.1.4.23 playNextDirectory()	29
4.1.4.24 playPrevious()	29
4.1.4.25 prepareFileByNumber()	29
4.1.4.26 repeatPart()	30
4.1.4.27 setChannel()	30
4.1.4.28 setDirectory()	30
4.1.4.29 setDrive()	31
4.1.4.30 setEqualizer()	31
4.1.4.31 setLoopMode()	32
4.1.4.32 setRepeatLoops()	32
4.1.4.33 setVolume()	32
4.1.4.34 startSendingRuntime()	33
4.1.4.35 stop()	33
4.1.4.36 stopCombined()	33
4.1.4.37 stopInsertedFile()	34
4.1.4.38 stopRepeatPart()	34
4.1.4.39 stopSendingRuntime()	34
<b>5 File Documentation</b>	<b>35</b>
5.1 playCombined.ino	35
5.2 playFileByName.ino	36
5.3 playFileByNumber.ino	37
5.4 DFR0534.cpp File Reference	38
5.4.1 Detailed Description	38
5.5 DFR0534.cpp	40
5.6 DFR0534.h File Reference	51
5.6.1 Detailed Description	51
5.6.2 Macro Definition Documentation	52
5.6.2.1 DFR0534_VERSION	52
5.7 DFR0534.h	52
<b>Index</b>	<b>55</b>

# Chapter 1

## DFR0534

An Arduino Uno/Nano/ATmega328p, ESP32 library for a [DFR0534](#) audio module. The library works with SoftwareSerial/ HardwareSerial and is very similar to [https://github.com/sleemanj/JQ8400\\_↵](https://github.com/sleemanj/JQ8400_↵) [Serial](#), but is no fork.

Examples, how to use the library

- [examples/playFileName/playFileName.ino](#)
- [examples/playFileByNumber/playFileByNumber.ino](#)
- [examples/playCombined/playCombined.ino](#)

### 1.1 SoftwareSerial for Arduino Uno/Nano/ATmega328p

To create a [DFR0534](#) object pass the SoftwareSerial object as a parameter to the [DFR0534](#) constructor, for example

```
#include <SoftwareSerial.h>
#include <DFR0534.h>

#define TX_PIN A0
#define RX_PIN A1
SoftwareSerial g_serial(RX_PIN, TX_PIN);
DFR0534 g_audio(g_serial);
...
```

### 1.2 HardwareSerial for ESP32

To create a [DFR0534](#) object pass the HardwareSerial object as a parameter to the [DFR0534](#) constructor, for example

```
#include <DFR0534.h>

#define TX_PIN 19
#define RX_PIN 23
HardwareSerial g_serial(1);

DFR0534 g_audio(g_serial);

void setup() {
  g_serial.begin(9600, SERIAL_8N1, RX_PIN, TX_PIN);
  ...
}
```

## 1.3 License and copyright

This library is licensed under the terms of

BSD 2-Clause License

Copyright (c) 2024, codingABI All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 1.4 Appendix

### 1.4.1 DFR0534 pinout

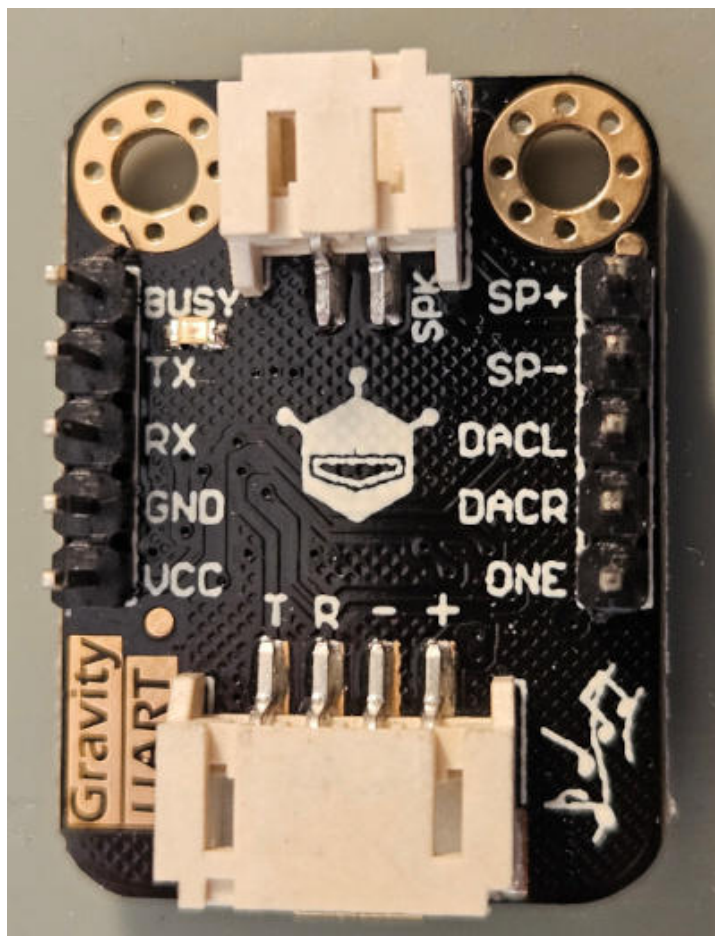


Figure 1.1 DFR0534 frontside

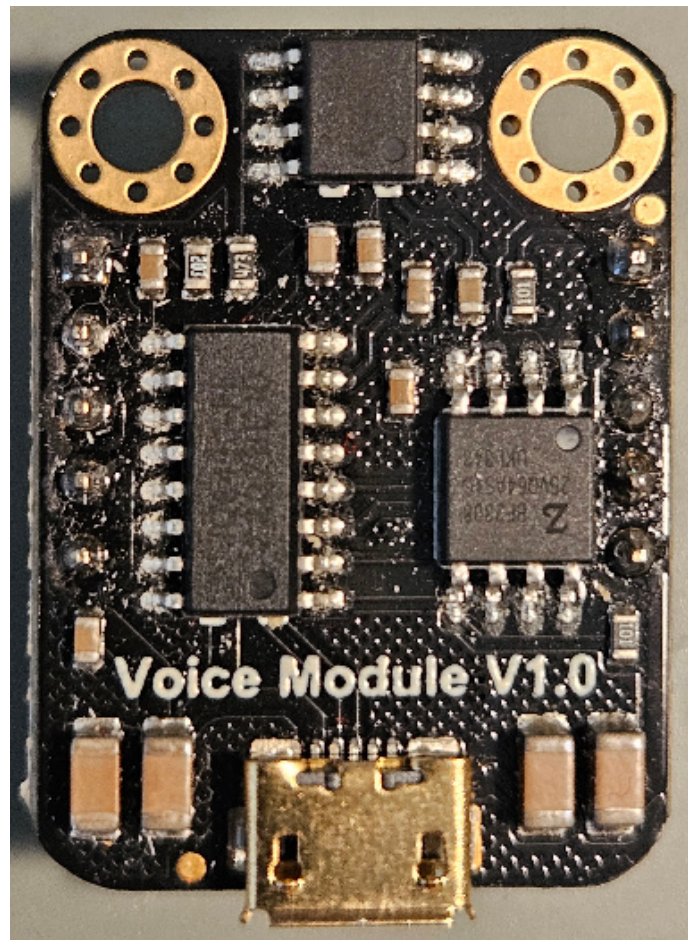


Figure 1.2 DFR0534 backside

Minimal schematic to use this library

Pin	Connected to
TX	SoftwareSerial/HardwareSerial RX
RX	SoftwareSerial/HardwareSerial TX*
GND	Ground
VCC	3.3-5V
SP+	Speaker + connector
SP-	Speaker - connector

\*If your microcontroller runs at 5V, use a 1k resistor between RX and SoftwareSerial/HardwareSerial TX.



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">DFR0534</a>	
Class for a <a href="#">DFR0534</a> audio module	9



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">playCombined.ino</a>	35
<a href="#">playFileByName.ino</a>	36
<a href="#">playFileByNumber.ino</a>	37
<a href="#">DFR0534.cpp</a>	38
<a href="#">DFR0534.h</a>	51



# Chapter 4

## Class Documentation

### 4.1 DFR0534 Class Reference

Class for a [DFR0534](#) audio module.

```
#include <DFR0534.h>
```

#### Public Types

- enum [DFR0534CHANNELS](#) { [CHANNELMP3](#) , [CHANNELAUX](#) , [CHANNELMP3AUX](#) , [CHANNELUNKNOWN](#) }
- enum [DFR0534DRIVE](#) { [DRIVEUSB](#) , [DRIVESD](#) , [DRIVEFLASH](#) , [DRIVEUNKNOWN](#) , [DRIVENO](#) = 0xff }
- enum [DFR0534LOOPMODE](#) { [LOOPBACKALL](#) , [SINGLEAUDIOLOOP](#) , [SINGLEAUDIOSTOP](#) , [PLAYRANDOM](#) , [DIRECTORYLOOP](#) , [RANDOMINDIRECTORY](#) , [SEQUENTIALINDIRECTORY](#) , [SEQUENTIAL](#) , [PLAYMODEUNKNOWN](#) }
- enum [DFR0534EQ](#) { [NORMAL](#) , [POP](#) , [ROCK](#) , [JAZZ](#) , [CLASSIC](#) , [EQUNKNOWN](#) }
- enum [DFR0534STATUS](#) { [STOPPED](#) , [PLAYING](#) , [PAUSED](#) , [STATUSUNKNOWN](#) }

#### Public Member Functions

- [DFR0534](#) (Stream &stream)  
*Constructor of a the [DFR0534](#) audio module.*
- void [decreaseVolume](#) ()  
*Decrease volume by one step.*
- void [fastBackwardDuration](#) (word seconds)  
*Fast backward.*
- void [fastForwardDuration](#) (word seconds)  
*Fast forward in seconds.*
- byte [getDrive](#) ()  
*Get current drive.*
- byte [getDrivesStates](#) ()

- Checks which drives are ready/online.*

  - bool `getDuration` (byte &hour, byte &minute, byte &second)

*Get duration/length of current file.*
- bool `getFileName` (char \*name)

*Get name for current file.*
- word `getFileNumber` ()

*Get file number of current file.*
- int `getFirstFileNumberInCurrentDirectory` ()

*Get number of first file in current directory.*
- bool `getRuntime` (byte &hour, byte &minute, byte &second)

*Get elapsed runtime/duration of the current file.*
- byte `getStatus` ()

*Get module status.*
- int `getTotalFiles` ()

*Get total number of supported audio files on current drive.*
- int `getTotalFilesInCurrentDirectory` ()

*Count all audio files for the current directory.*
- void `increaseVolume` ()

*Increase volume by one step.*
- void `insertFileByNumber` (word track, byte drive=`DRIVEFLASH`)

*Pause current file and play another file by number.*
- void `pause` ()

*Pause the current file.*
- void `play` ()

*Play the current selected file.*
- void `playCombined` (char \*list)

*Combined/concatenated play of files.*
- void `playFileByName` (char \*path, byte drive=`DRIVEFLASH`)

*Play audio file by file name/path.*
- void `playFileByNumber` (word track)

*Play audio file by number.*
- void `playLastInDirectory` ()

*Play last file in directory (in "file copy order")*
- void `playNext` ()

*Play next file (in "file copy order")*
- void `playNextDirectory` ()

*Play first file in next directory (in "file copy order")*
- void `playPrevious` ()

*Play previous file (in "file copy order")*
- void `prepareFileByNumber` (word track)

*Select file by number, but not start playing.*
- void `repeatPart` (byte startMinute, byte startSecond, byte stopMinute, byte stopSecond)

*Repeat part of the current file.*
- void `setChannel` (byte channel)

*Set output for DAC to channel MP3, AUX or both.*
- void `setDirectory` (char \*path, byte drive=`DRIVEFLASH`)

*Should set directory, but does not work for me.*
- void `setDrive` (byte drive)

*Switch to drive.*
- void `setEqualizer` (byte mode)

*Set equalizer to NORMAL, POP, ROCK, JAZZ or CLASSIC.*

- void [setLoopMode](#) (byte mode)  
*Set loop mode.*
- void [setRepeatLoops](#) (word loops)  
*Set repeat loops.*
- void [setVolume](#) (byte volume)  
*Set volume.*
- void [stop](#) ()  
*Stop the current file.*
- void [stopInsertedFile](#) ()  
*Stop inserted file.*
- void [startSendingRuntime](#) ()  
*Start sending elapsed runtime every 1 second.*
- void [stopCombined](#) ()  
*Stop combined play (playlist)*
- void [stopRepeatPart](#) ()  
*Stop repeating part of the current file.*
- void [stopSendingRuntime](#) ()  
*Stop sending runtime.*

### 4.1.1 Detailed Description

Class for a [DFR0534](#) audio module.

Definition at line 32 of file [DFR0534.h](#).

### 4.1.2 Member Enumeration Documentation

#### 4.1.2.1 DFR0534CHANNELS

```
enum DFR0534::DFR0534CHANNELS
```

Supported input channels

Enumerator

<a href="#">CHANNELMP3</a>	Use MP3 input channel for DAC output (=default after device startup)
<a href="#">CHANNELAUX</a>	Use AUX input (P26 and P27) for DAC output
<a href="#">CHANNELMP3AUX</a>	Combines MP3 and AUX audio from P26 and P27 for DAC output
<a href="#">CHANNELUNKNOWN</a>	Unknown

Definition at line 35 of file [DFR0534.h](#).

```
00036     {
00037         CHANNELMP3,
00038         CHANNELAUX,
00039         CHANNELMP3AUX,
00040         CHANNELUNKNOWN
00041     };
```

#### 4.1.2.2 DFR0534DRIVE

```
enum DFR0534::DFR0534DRIVE
```

Supported drives

## Enumerator

DRIVEUSB	USB drive
DRIVESD	SD card
DRIVEFLASH	Flash memory chip
DRIVEUNKNOWN	Unknown
DRIVENO	No drive

Definition at line 43 of file [DFR0534.h](#).

```
00044 {
00045     DRIVEUSB,
00046     DRIVESD,
00047     DRIVEFLASH,
00048     DRIVEUNKNOWN,
00049     DRIVENO = 0xff
00050 };
```

## 4.1.2.3 DFR0534EQ

```
enum DFR0534::DFR0534EQ
```

## EQ modes

## Enumerator

NORMAL	(=default after device startup)
--------	---------------------------------

Definition at line 65 of file [DFR0534.h](#).

```
00066 {
00067     NORMAL,
00068     POP,
00069     ROCK,
00070     JAZZ ,
00071     CLASSIC,
00072     EQUNKNOWN
00073 };
```

## 4.1.2.4 DFR0534LOOPMODE

```
enum DFR0534::DFR0534LOOPMODE
```

## Loop modes

## Enumerator

LOOPBACKALL	Every file on drive in "file copy order" and loop afterwards
SINGLEAUDIOLOOP	Repeat current file
SINGLEAUDIOSTOP	Stops after single file (=default after device startup)
PLAYRANDOM	Random play order
DIRECTORYLOOP	Every file in current director in "file copy order" and loop afterwards
RANDOMINDIRECTORY	Random play order in current directory
SEQUENTIALINDIRECTORY	Every file in current directory in "file copy order" without loop
SEQUENTIAL	Every file on drive in "file copy order" without loop
PLAYMODEUNKNOWN	Unknown



Definition at line 52 of file [DFR0534.h](#).

```
00053     {
00054         LOOPBACKALL,
00055         SINGLEAUDIOLOOP,
00056         SINGLEAUDIOSTOP,
00057         PLAYRANDOM,
00058         DIRECTORYLOOP,
00059         RANDOMINDIRECTORY,
00060         SEQUENTIALINDIRECTORY,
00061         SEQUENTIAL,
00062         PLAYMODEUNKNOWN
00063     };
```

#### 4.1.2.5 DFR0534STATUS

enum [DFR0534::DFR0534STATUS](#)

Modul states

Enumerator

STOPPED	Audio module is idle
PLAYING	Audio module is playing a file
PAUSED	Audio module is paused
STATUSUNKNOWN	Unkown

Definition at line 75 of file [DFR0534.h](#).

```
00076     {
00077         STOPPED,
00078         PLAYING,
00079         PAUSED,
00080         STATUSUNKNOWN
00081     };
```

### 4.1.3 Constructor & Destructor Documentation

#### 4.1.3.1 DFR0534()

```
DFR0534::DFR0534 (
    Stream & stream) [inline]
```

Constructor of a the [DFR0534](#) audio module.

Parameters

in	<i>stream</i>	Serial connection object, like SoftwareSerial or HardwareSerial
----	---------------	---

Definition at line 87 of file [DFR0534.h](#).

```
00088     {
00089         m_ptrStream = &stream;
00090     }
```

## 4.1.4 Member Function Documentation

### 4.1.4.1 decreaseVolume()

```
void DFR0534::decreaseVolume ()
```

Decrease volume by one step.

Definition at line 748 of file [DFR0534.cpp](#).

```
00749 {
00750     if (m_ptrStream == NULL) return; // Should not happen
00751     sendStartingCode();
00752     sendDataByte(0x15);
00753     sendDataByte(0x00);
00754     sendChecksum();
00755 }
```

### 4.1.4.2 fastBackwardDuration()

```
void DFR0534::fastBackwardDuration (
    word seconds)
```

Fast backward.

Fast backward in seconds

Parameters

in	<i>seconds</i>	Seconds to go backward
----	----------------	------------------------

Definition at line 1025 of file [DFR0534.cpp](#).

```
01026 {
01027     if (m_ptrStream == NULL) return; // Should not happen
01028     sendStartingCode();
01029     sendDataByte(0x22);
01030     sendDataByte(0x02);
01031     sendDataByte((seconds » 8) & 0xff);
01032     sendDataByte(seconds & 0xff);
01033     sendChecksum();
01034 }
```

### 4.1.4.3 fastForwardDuration()

```
void DFR0534::fastForwardDuration (
    word seconds)
```

Fast forward in seconds.

Parameters

in	<i>seconds</i>	Seconds to go forward
----	----------------	-----------------------

Definition at line 1042 of file [DFR0534.cpp](#).

```
01043 {
01044     if (m_ptrStream == NULL) return; // Should not happen
01045     sendStartingCode();
01046     sendDataByte(0x23);
01047     sendDataByte(0x02);
01048     sendDataByte((seconds » 8) & 0xff);
01049     sendDataByte(seconds & 0xff);
01050     sendChecksum();
01051 }
```

### 4.1.4.4 getDrive()

```
byte DFR0534::getDrive ()
```

Get current drive.

## Return values

<a href="#"><i>DFR0534::DRIVEUSB</i></a>	USB drive
<a href="#"><i>DFR0534::DRIVESD</i></a>	SD card
<a href="#"><i>DFR0534::DRIVEFLASH</i></a>	Flash memory chip
<a href="#"><i>DFR0534::DRIVENO</i></a>	No drive found
<a href="#"><i>DFR0534::DRIVEUNKNOWN</i></a>	Error (for example request timeout)

Definition at line 345 of file [DFR0534.cpp](#).

```

00346 {
00347     #define COMMAND 0x0A
00348     #define RECEIVEBYTETIMEOUTMS 100
00349     #define RECEIVEGLOBALTIMEOUTMS 500
00350     #define RECEIVEFAILED DRIVEUNKNOWN
00351     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00352
00353     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00354     sendStartingCode();
00355     sendDataByte(COMMAND);
00356     sendDataByte(0x00);
00357     sendChecksum();
00358
00359     // Receive
00360     int i=0;
00361     byte data, firstByte = 0, sum, length=0xff, result = 0;
00362     unsigned long receiveStartMS = millis();
00363     do {
00364         byte dataReady = 0;
00365         unsigned long lastMS = millis();
00366         // Wait for response or timeout
00367         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00368
00369         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00370         data = m_ptrStream->read();
00371
00372         if (i==0) { // Begin of transmission
00373             firstByte=data;
00374             sum = 0;
00375         }
00376         if ((i == 1) && (data != COMMAND)) {
00377             // Invalid signal => reset receive
00378             i=0;
00379             firstByte = 0;
00380         }
00381         if (i == RECEIVEHEADERLENGTH) {
00382             length = data; // Length of receiving data
00383             if (length != 1) {
00384                 // Invalid length => reset receive
00385                 i=0;
00386                 firstByte = 0;
00387             }
00388         }
00389         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00390             result = data;
00391         }
00392         if (firstByte == STARTINGCODE) {
00393             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00394             i++;
00395         }
00396         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00397     } while (i<length+RECEIVEHEADERLENGTH+2);
00398
00399     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00400     return result;
00401 }
```

#### 4.1.4.5 getDrivesStates()

```
byte DFR0534::getDrivesStates ()
```

Checks which drives are ready/online.

Returned value is a bit pattern that shows which drives are ready/online (1=online,0=offline):

- Bit 0 = [DFR0534::DRIVEUSB](#)
- Bit 1 = [DFR0534::DRIVESD](#)
- Bit 2 = [DFR0534::DRIVEFLASH](#)

## Returns

Bit pattern for drives

## Return values

<a href="#">DFR0534::DRIVEUNKNOWN</a>	Error (for example request timeout)
---------------------------------------	-------------------------------------

Definition at line 278 of file [DFR0534.cpp](#).

```

00279 {
00280     #define COMMAND 0x09
00281     #define RECEIVEBYTETIMEOUTMS 100
00282     #define RECEIVEGLOBALTIMEOUTMS 500
00283     #define RECEIVEFAILED DRIVEUNKNOWN
00284     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00285
00286     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00287     sendStartingCode();
00288     sendDataByte(COMMAND);
00289     sendDataByte(0x00);
00290     sendChecksum();
00291
00292     // Receive
00293     int i=0;
00294     byte data, firstByte = 0, sum, length=0xff, result = 0;
00295     unsigned long receiveStartMS = millis();
00296     do {
00297         byte dataReady = 0;
00298         unsigned long lastMS = millis();
00299         // Wait for response or timeout
00300         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00301
00302         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00303         data = m_ptrStream->read();
00304
00305         if (i==0) { // Begin of transmission
00306             firstByte=data;
00307             sum = 0;
00308         }
00309         if ((i == 1) && (data != COMMAND)) {
00310             // Invalid signal => reset receive
00311             i=0;
00312             firstByte = 0;
00313         }
00314         if (i == RECEIVEHEADERLENGTH) {
00315             length = data; // Length of receiving data
00316             if (length != 1) {
00317                 // Invalid length => reset receive
00318                 i=0;
00319                 firstByte = 0;
00320             }
00321         }
00322         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00323             result = data;
00324         }
00325         if (firstByte == STARTINGCODE) {
00326             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00327             i++;
00328         }
00329         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00330     } while (i<length+RECEIVEHEADERLENGTH+2);
00331
00332     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00333     return result;
00334 }
```

## 4.1.4.6 getDuration()

```
bool DFR0534::getDuration (
    byte & hour,
    byte & minute,
    byte & second)
```

Get duration/length of current file.

Get duration/length of current file in hours:minutes:seconds

## Parameters

out	<i>hour</i>	Hours
out	<i>minute</i>	Minutes
out	<i>second</i>	Seconds

## Return values

<i>true</i>	Request was successful
<i>false</i>	Request failed

Definition at line 1065 of file DFR0534.cpp.

```
01066 {
01067     #define COMMAND 0x24
01068     #define RECEIVEFAILED false
01069     #define RECEIVEBYTETIMEOUTMS 100
01070     #define RECEIVEGLOBALTIMEOUTMS 500
01071     #define RECEIVEHEADERLENGTH 2 // startingcode+command
01072
01073     if (m_ptrStream == NULL) return false; // Should not happen
01074     sendStartingCode();
01075     sendDataByte(COMMAND);
01076     sendDataByte(0x00);
01077     sendChecksum();
01078
01079     // Receive
01080     int i=0;
01081     byte data, firstByte = 0, sum, length=0xff;
01082     word result = 0;
01083     unsigned long receiveStartMS = millis();
01084     do {
01085         byte dataReady = 0;
01086         unsigned long lastMS = millis();
01087         // Wait for response or timeout
01088         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01089
01090         if (dataReady == 0) return RECEIVEFAILED; // Timeout
01091         data = m_ptrStream->read();
01092
01093         if (i==0) { // Begin of transmission
01094             firstByte=data;
01095             sum = 0;
01096         }
01097         if ((i == 1) && (data != COMMAND)) {
01098             // Invalid signal => reset receive
01099             i=0;
01100             firstByte = 0;
01101         }
01102         if (i == RECEIVEHEADERLENGTH) {
01103             length = data; // Length of receiving data
01104             if (length != 3) {
01105                 // Invalid length => reset receive
01106                 i=0;
01107                 firstByte = 0;
01108             }
01109         }
01110         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01111             switch (i-RECEIVEHEADERLENGTH-1) {
01112                 case 0:
```

```

01113         hour=data;
01114         break;
01115     case 1:
01116         minute=data;
01117         break;
01118     case 2:
01119         second=data;
01120         break;
01121     }
01122 }
01123 if (firstByte == STARTINGCODE) {
01124     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01125     i++;
01126 }
01127 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01128 } while (i<length+RECEIVEHEADERLENGTH+2);
01129
01130 return (data == sum); // Does checksum matches?
01131 }

```

#### 4.1.4.7 getFileName()

```

bool DFR0534::getFileName (
    char * name)

```

Get name for current file.

File name is in 8+3 format in upper case, with spaces without the dot "." between name and extension, e.g. "TEST WAV" for the file test.wav

##### Parameters

out	<i>name</i>	Filename. You have to allocate at least 12 chars memory for this variable.
-----	-------------	--

Definition at line 913 of file [DFR0534.cpp](#).

```

00914 {
00915     #define COMMAND 0x1E
00916     #define RECEIVEBYTETIMEOUTMS 100
00917     #define RECEIVEGLOBALTIMEOUTMS 500
00918     #define RECEIVEFAILED false
00919     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00920
00921     if (m_ptrStream == NULL) return false; // Should not happen
00922     if (name == NULL) return false;
00923     name[0] = '\0';
00924
00925     sendStartingCode();
00926     sendDataByte(COMMAND);
00927     sendDataByte(0x00);
00928     sendChecksum();
00929
00930     // Receive
00931     int i=0;
00932     byte data, firstByte = 0, sum, length=0xff;
00933     unsigned long receiveStartMS = millis();
00934     do {
00935         byte dataReady = 0;
00936         unsigned long lastMS = millis();
00937         // Wait for response or timeout
00938         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00939
00940         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00941         data = m_ptrStream->read();
00942         if (i==0) { // Begin of transmission
00943             firstByte=data;
00944             sum = 0;
00945         }
00946         if ((i == 1) && (data != COMMAND)) {
00947             // Invalid signal => reset receive
00948             i=0;
00949             firstByte = 0;
00950         }
00951         if (i == RECEIVEHEADERLENGTH) length = data; // Length of receiving string
00952         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {

```

```

00953         if ((i-RECEIVEHEADERLENGTH) < 12) { // I expect no longer file names than 8+3 chars plus '\0'
00954             name[i-RECEIVEHEADERLENGTH-1] = data;
00955             name[i-RECEIVEHEADERLENGTH] = '\0';
00956         }
00957     }
00958     if (firstByte == STARTINGCODE) {
00959         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00960         i++;
00961     }
00962     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00963 } while (i<length+RECEIVEHEADERLENGTH+2);
00964 return (data == sum); // Does checksum matches?
00965 }

```

#### 4.1.4.8 getFileNumber()

word DFR0534::getFileNumber ()

Get file number of current file.

File number is in "file copy order". First audio file copied to the drive get number 1...

##### Returns

File number

##### Return values

0	Error (for example request timeout)
---	-------------------------------------

Definition at line 427 of file DFR0534.cpp.

```

00428 {
00429     #define COMMAND 0x0D
00430     #define RECEIVEFAILED 0
00431     #define RECEIVEBYTETIMEOUTMS 100
00432     #define RECEIVEGLOBALTIMEOUTMS 500
00433     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00434
00435     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00436     sendStartingCode();
00437     sendDataByte(COMMAND);
00438     sendDataByte(0x00);
00439     sendChecksum();
00440
00441     // Receive
00442     int i=0;
00443     byte data, firstByte = 0, sum, length=0xff;
00444     word result = 0;
00445     unsigned long receiveStartMS = millis();
00446     do {
00447         byte dataReady = 0;
00448         unsigned long lastMS = millis();
00449         // Wait for response or timeout
00450         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00451
00452         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00453         data = m_ptrStream->read();
00454
00455         if (i==0) { // Begin of transmission
00456             firstByte=data;
00457             sum = 0;
00458         }
00459         if ((i == 1) && (data != COMMAND)) {
00460             // Invalid signal => reset receive
00461             i=0;
00462             firstByte = 0;
00463         }
00464         if (i == RECEIVEHEADERLENGTH) {
00465             length = data; // Length of receiving data
00466             if (length != 2) {
00467                 // Invalid length => reset receive
00468                 i=0;

```

```

00469         firstByte = 0;
00470     }
00471 }
00472 if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00473     switch (i-RECEIVEHEADERLENGTH-1) {
00474         case 0:
00475             result=data<<8;
00476             break;
00477         case 1:
00478             result+=data;
00479             break;
00480     }
00481 }
00482 if (firstByte == STARTINGCODE) {
00483     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00484     i++;
00485 }
00486 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00487 } while (i<length+RECEIVEHEADERLENGTH+2);
00488
00489 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00490 return result;
00491 }

```

#### 4.1.4.9 getFirstFileNumberInCurrentDirectory()

```
int DFR0534::getFirstFileNumberInCurrentDirectory ()
```

Get number of first file in current directory.

#### Returns

File number

#### Return values

-1	Error (for example request timeout)
----	-------------------------------------

Definition at line 595 of file [DFR0534.cpp](#).

```

00596 {
00597     #define COMMAND 0x11
00598     #define RECEIVEFAILED -1
00599     #define RECEIVEBYTETIMEOUTMS 100
00600     #define RECEIVEGLOBALTIMEOUTMS 500
00601     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00602
00603     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00604     sendStartingCode();
00605     sendDataByte(COMMAND);
00606     sendDataByte(0x00);
00607     sendCheckSum();
00608
00609     // Receive
00610     int i=0;
00611     byte data, firstByte = 0, sum, length=0xff;
00612     word result = 0;
00613     unsigned long receiveStartMS = millis();
00614     do {
00615         byte dataReady = 0;
00616         unsigned long lastMS = millis();
00617         // Wait for response or timeout
00618         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00619
00620         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00621         data = m_ptrStream->read();
00622
00623         if (i==0) { // Begin of transmission
00624             firstByte=data;
00625             sum = 0;
00626         }
00627         if ((i == 1) && (data != COMMAND)) {
00628             // Invalid signal => reset receive
00629             i=0;

```



```

00630     firstByte = 0;
00631 }
00632 if (i == RECEIVEHEADERLENGTH) {
00633     length = data; // Length of receiving data
00634     if (length != 2) {
00635         // Invalid length => reset receive
00636         i=0;
00637         firstByte = 0;
00638     }
00639 }
00640 if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00641     switch (i-RECEIVEHEADERLENGTH-1) {
00642         case 0:
00643             result=data<<8;
00644             break;
00645         case 1:
00646             result+=data;
00647             break;
00648     }
00649 }
00650 if (firstByte == STARTINGCODE) {
00651     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00652     i++;
00653 }
00654 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00655 } while (i<length+RECEIVEHEADERLENGTH+2);
00656
00657 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00658 return result;
00659 }

```

#### 4.1.4.10 getRuntime()

```

bool DFR0534::getRuntime (
    byte & hour,
    byte & minute,
    byte & second)

```

Get elapsed runtime/duration of the current file.

Runtime is in hours:minutes:seconds. You have to call [startSendingRuntime\(\)](#) before runtimes can be received.

##### Parameters

out	<i>hour</i>	Hours
out	<i>minute</i>	Minutes
out	<i>second</i>	Seconds

##### Return values

<i>true</i>	Request was successful
<i>false</i>	Request failed

Definition at line 1158 of file [DFR0534.cpp](#).

```

01159 {
01160     #define COMMAND 0x25
01161     #define RECEIVEFAILED false
01162     #define RECEIVEBYTETIMEOUTMS 100
01163     #define RECEIVEGLOBALTIMEOUTMS 500
01164     #define RECEIVEHEADERLENGTH 2 // startingcode+command
01165
01166     if (m_ptrStream == NULL) return false; // Should not happen
01167
01168     // Receive
01169     int i=0;
01170     byte data, firstByte = 0, sum, length=0xff;
01171     word result = 0;
01172     unsigned long receiveStartMS = millis();
01173     do {

```

```

01174     byte dataReady = 0;
01175     unsigned long lastMS = millis();
01176     // Wait for response or timeout
01177     while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01178
01179     if (dataReady == 0) return RECEIVEFAILED; // Timeout
01180     data = m_ptrStream->read();
01181
01182     if (i==0) { // Begin of transmission
01183         firstByte=data;
01184         sum = 0;
01185     }
01186     if ((i == 1) && (data != COMMAND)) {
01187         // Invalid signal => reset receive
01188         i=0;
01189         firstByte = 0;
01190     }
01191     if (i == RECEIVEHEADERLENGTH) {
01192         length = data; // Length of receiving data
01193         if (length != 3) {
01194             // Invalid length => reset receive
01195             i=0;
01196             firstByte = 0;
01197         }
01198     }
01199     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01200         switch (i-RECEIVEHEADERLENGTH-1) {
01201             case 0:
01202                 hour=data;
01203                 break;
01204             case 1:
01205                 minute=data;
01206                 break;
01207             case 2:
01208                 second=data;
01209                 break;
01210         }
01211     }
01212     if (firstByte == STARTINGCODE) {
01213         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01214         i++;
01215     }
01216     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01217 } while (i<length+RECEIVEHEADERLENGTH+2);
01218
01219 return (data == sum); // Does checksum matches?
01220 }

```

#### 4.1.4.11 getStatus()

```
byte DFR0534::getStatus ()
```

Get module status.

##### Return values

<a href="#"><i>DFR0534::STOPPED</i></a>	Audio module is idle
<a href="#"><i>DFR0534::PLAYING</i></a>	Audio module is playing a file
<a href="#"><i>DFR0534::PAUSED</i></a>	Audio module is paused
<a href="#"><i>DFR0534::STATUSUNKNOWN</i></a>	Error (for example request timeout)

Definition at line 53 of file [DFR0534.cpp](#).

```

00054 {
00055     #define COMMAND 0x01
00056     #define RECEIVEBYTETIMEOUTMS 100
00057     #define RECEIVEGLOBALTIMEOUTMS 500
00058     #define RECEIVEFAILED STATUSUNKNOWN
00059     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00060
00061     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00062     sendStartingCode();
00063     sendDataByte(COMMAND);
00064     sendDataByte(0x00);
00065     sendChecksum();

```

```

00066
00067 // Receive
00068 int i=0;
00069 byte data, firstByte = 0, sum, length=0xff, result = 0;
00070 unsigned long receiveStartMS = millis();
00071 do {
00072     byte dataReady = 0;
00073     unsigned long lastMS = millis();
00074     // Wait for response or timeout
00075     while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00076
00077     if (dataReady == 0) return RECEIVEFAILED; // Timeout
00078     data = m_ptrStream->read();
00079
00080     if (i==0) { // Begin of transmission
00081         firstByte=data;
00082         sum = 0;
00083     }
00084     if ((i == 1) && (data != COMMAND)) {
00085         // Invalid signal => reset receive
00086         i=0;
00087         firstByte = 0;
00088     }
00089     if (i == RECEIVEHEADERLENGTH) {
00090         length = data; // Length of receiving data
00091         if (length != 1) {
00092             // Invalid length => reset receive
00093             i=0;
00094             firstByte = 0;
00095         }
00096     }
00097     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00098         result = data;
00099     }
00100     if (firstByte == STARTINGCODE) {
00101         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00102         i++;
00103     }
00104     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00105 } while (i<length+RECEIVEHEADERLENGTH+2);
00106
00107 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00108 return result;
00109 }

```

#### 4.1.4.12 getTotalFiles()

```
int DFR0534::getTotalFiles ()
```

Get total number of supported audio files on current drive.

##### Returns

Number of files

##### Return values

-1	Error (for example request timeout)
----	-------------------------------------

Definition at line 499 of file [DFR0534.cpp](#).

```

00500 {
00501     #define COMMAND 0x0C
00502     #define RECEIVEFAILED -1
00503     #define RECEIVEBYTETIMEOUTMS 100
00504     #define RECEIVEGLOBALTIMEOUTMS 500
00505     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00506
00507     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00508     sendStartingCode();
00509     sendDataByte(COMMAND);
00510     sendDataByte(0x00);
00511     sendChecksum();
00512

```

```

00513 // Receive
00514 int i=0;
00515 byte data, firstByte = 0, sum, length=0xff;
00516 word result = 0;
00517 unsigned long receiveStartMS = millis();
00518 do {
00519     byte dataReady = 0;
00520     unsigned long lastMS = millis();
00521     // Wait for response or timeout
00522     while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00523
00524     if (dataReady == 0) return RECEIVEFAILED; // Timeout
00525     data = m_ptrStream->read();
00526
00527     if (i==0) { // Begin of transmission
00528         firstByte=data;
00529         sum = 0;
00530     }
00531     if ((i == 1) && (data != COMMAND)) {
00532         // Invalid signal => reset receive
00533         i=0;
00534         firstByte = 0;
00535     }
00536     if (i == RECEIVEHEADERLENGTH) {
00537         length = data; // Length of receiving data
00538         if (length != 2) {
00539             // Invalid length => reset receive
00540             i=0;
00541             firstByte = 0;
00542         }
00543     }
00544     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00545         switch (i-RECEIVEHEADERLENGTH-1) {
00546             case 0:
00547                 result=data<<8;
00548                 break;
00549             case 1:
00550                 result+=data;
00551                 break;
00552         }
00553     }
00554     if (firstByte == STARTINGCODE) {
00555         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00556         i++;
00557     }
00558     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00559 } while (i<length+RECEIVEHEADERLENGTH+2);
00560
00561 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00562 return result;
00563 }

```

#### 4.1.4.13 getTotalFilesInCurrentDirectory()

```
int DFR0534::getTotalFilesInCurrentDirectory ()
```

Count all audio files for the current directory.

##### Returns

File count

##### Return values

-1	Error (for example request timeout)
----	-------------------------------------

Definition at line 667 of file [DFR0534.cpp](#).

```

00668 {
00669     #define COMMAND 0x12
00670     #define RECEIVEFAILED -1
00671     #define RECEIVEBYTETIMEOUTMS 100
00672     #define RECEIVEGLOBALTIMEOUTMS 500
00673     #define RECEIVEHEADERLENGTH 2 // startingcode+command

```

```

00674
00675     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00676     sendStartingCode();
00677     sendDataByte(COMMAND);
00678     sendDataByte(0x00);
00679     sendChecksum();
00680
00681     // Receive
00682     int i=0;
00683     byte data, firstByte = 0, sum, length=0xff;
00684     word result = 0;
00685     unsigned long receiveStartMS = millis();
00686     do {
00687         byte dataReady = 0;
00688         unsigned long lastMS = millis();
00689         // Wait for response or timeout
00690         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00691
00692         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00693         data = m_ptrStream->read();
00694
00695         if (i==0) { // Begin of transmission
00696             firstByte=data;
00697             sum = 0;
00698         }
00699         if ((i == 1) && (data != COMMAND)) {
00700             // Invalid signal => reset receive
00701             i=0;
00702             firstByte = 0;
00703         }
00704         if (i == RECEIVEHEADERLENGTH) {
00705             length = data; // Length of receiving data
00706             if (length != 2) {
00707                 // Invalid length => reset receive
00708                 i=0;
00709                 firstByte = 0;
00710             }
00711         }
00712         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00713             switch (i-RECEIVEHEADERLENGTH-1) {
00714                 case 0:
00715                     result=data<<8;
00716                     break;
00717                 case 1:
00718                     result+=data;
00719                     break;
00720             }
00721         }
00722         if (firstByte == STARTINGCODE) {
00723             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00724             i++;
00725         }
00726         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00727     } while (i<length+RECEIVEHEADERLENGTH+2);
00728
00729     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00730     return result;
00731 }

```

#### 4.1.4.14 increaseVolume()

```
void DFR0534::increaseVolume ()
```

Increase volume by one step.

Definition at line 736 of file [DFR0534.cpp](#).

```

00737 {
00738     if (m_ptrStream == NULL) return; // Should not happen
00739     sendStartingCode();
00740     sendDataByte(0x14);
00741     sendDataByte(0x00);
00742     sendChecksum();
00743 }

```

#### 4.1.4.15 insertFileByNumber()

```
void DFR0534::insertFileByNumber (
    word track,
    byte drive = DRIVEFLASH)
```

Pause current file and play another file by number.

File number order is "file copy order". Continue original file when this file stops

##### Parameters

in	<i>track</i>	File number of the audio file
in	<i>drive</i>	Drive, where file is stored: Drive <a href="#">DFR0534::DRIVEUSB</a> , <a href="#">DFR0534::DRIVESD</a> or <a href="#">DFR0534::DRIVEFLASH</a> (=default)

Definition at line 765 of file [DFR0534.cpp](#).

```
00766 {
00767     if (m_ptrStream == NULL) return; // Should not happen
00768     if (drive >= DRIVEUNKNOWN) return;
00769     sendStartingCode();
00770     sendDataByte(0x16);
00771     sendDataByte(0x03);
00772     sendDataByte(drive);
00773     sendDataByte((track >> 8) & 0xff);
00774     sendDataByte(track & 0xff);
00775     sendChecksum();
00776 }
```

#### 4.1.4.16 pause()

```
void DFR0534::pause ()
```

Pause the current file.

Definition at line 180 of file [DFR0534.cpp](#).

```
00181 {
00182     if (m_ptrStream == NULL) return; // Should not happen
00183     sendStartingCode();
00184     sendDataByte(0x03);
00185     sendDataByte(0x00);
00186     sendChecksum();
00187 }
```

#### 4.1.4.17 play()

```
void DFR0534::play ()
```

Play the current selected file.

Definition at line 168 of file [DFR0534.cpp](#).

```
00169 {
00170     if (m_ptrStream == NULL) return; // Should not happen
00171     sendStartingCode();
00172     sendDataByte(0x02);
00173     sendDataByte(0x00);
00174     sendChecksum();
00175 }
```

#### 4.1.4.18 playCombined()

```
void DFR0534::playCombined (
    char * list)
```

Combined/concatenated play of files.

Combined is like a playlist, for example `playCombined("0103")` for the two files 01 and 03. The Filenames must be two chars long and the files must be in a directory called /ZH Combined playback ignores loop mode and stops after last file.

## Parameters

in	<i>list</i>	Concatenated list of all files to play
----	-------------	--

Definition at line 858 of file [DFR0534.cpp](#).

```

00859 {
00860     if (m_ptrStream == NULL) return; // Should not happen
00861     if (list == NULL) return;
00862     if ((strlen(list) % 2) != 0) return;
00863
00864     sendStartingCode();
00865     sendDataByte(0x1B);
00866     sendDataByte(strlen(list));
00867     for (int i=0; i<strlen(list); i++) {
00868         sendDataByte(list[i]);
00869     }
00870     sendChecksum();
00871 }
```

#### 4.1.4.19 playFileByName()

```

void DFR0534::playFileByName (
    char * path,
    byte drive = DRIVEFLASH)
```

Play audio file by file name/path.

The file name/path is the full path of the audio file to be played in format which looks like a special unix 8+3 format:

- Without a dot between file name and file extension
- All characters in upper case
- Maximal 8 characters for file name
- Every file and folder whose name length is shorter then 8 chars must be filled up to the 8 chars length by space chars
- Must end with WAV or MP3
- Only WAV and MP3 files are supported
- Wildcards \* (=multiple arbitrary characters) and ? (=one single arbitrary character) are allowed and can be used to reduce the filling space chars

Valid examples:

- "/01 WAV" for a file '/01.wav'
- "/99-AFR~1MP3" for a file '/99-Africa.mp3'
- "/SUN\*MP3" for first file matching '/sun\*.mp3', for example '/sun.mp3'
- "/99-AFR\*MP3" for first file matching '/99-Afr\*.mp3'
- "/10\*" for first audio file matching '/10\*.\*'
- "/10 /20 WAV" for the file '/10/20.wav' (first means first in "file copy order")

## Parameters

in	<i>path</i>	Full path of the audio file
in	<i>drive</i>	Drive, where file is stored: Drive <a href="#">DFR0534::DRIVEUSB</a> , <a href="#">DFR0534::DRIVESD</a> or <a href="#">DFR0534::DRIVEFLASH</a> (=default)

Definition at line 252 of file [DFR0534.cpp](#).

```

00253 {
00254     if (m_ptrStream == NULL) return; // Should not happen
00255     if (path == NULL) return;
00256     if (drive >= DRIVEUNKNOWN) return;
00257     sendStartingCode();
00258     sendDataByte(0x08);
00259     sendDataByte(strlen(path)+1);
00260     sendDataByte(drive);
00261     for (int i=0; i<strlen(path); i++) {
00262         sendDataByte(path[i]);
00263     }
00264     sendChecksum();
00265 }
```

#### 4.1.4.20 playFileByNumber()

```

void DFR0534::playFileByNumber (
    word track)
```

Play audio file by number.

File number order is "file copy order": First audio file copied to the drive gets number 1, second audio file copied gets number 2... )

## Parameters

in	<i>track</i>	File number
----	--------------	-------------

Definition at line 135 of file [DFR0534.cpp](#).

```

00136 {
00137     if (m_ptrStream == NULL) return; // Should not happen
00138     if (track <=0) return;
00139     sendStartingCode();
00140     sendDataByte(0x07);
00141     sendDataByte(0x02);
00142     sendDataByte((track >> 8) & 0xff);
00143     sendDataByte(track & 0xff);
00144     sendChecksum();
00145 }
```

#### 4.1.4.21 playLastInDirectory()

```

void DFR0534::playLastInDirectory ()
```

Play last file in directory (in "file copy order")

Definition at line 568 of file [DFR0534.cpp](#).

```

00569 {
00570     if (m_ptrStream == NULL) return; // Should not happen
00571     sendStartingCode();
00572     sendDataByte(0x0E);
00573     sendDataByte(0x00);
00574     sendChecksum();
00575 }
```



#### 4.1.4.22 playNext()

```
void DFR0534::playNext ()
```

Play next file (in "file copy order")

Definition at line 216 of file [DFR0534.cpp](#).

```

00217 {
00218     if (m_ptrStream == NULL) return; // Should not happen
00219     sendStartingCode();
00220     sendDataByte(0x06);
00221     sendDataByte(0x00);
00222     sendChecksum();
00223 }
```

#### 4.1.4.23 playNextDirectory()

```
void DFR0534::playNextDirectory ()
```

Play first file in next directory (in "file copy order")

Definition at line 580 of file [DFR0534.cpp](#).

```

00581 {
00582     if (m_ptrStream == NULL) return; // Should not happen
00583     sendStartingCode();
00584     sendDataByte(0x0F);
00585     sendDataByte(0x00);
00586     sendChecksum();
00587 }
```

#### 4.1.4.24 playPrevious()

```
void DFR0534::playPrevious ()
```

Play previous file (in "file copy order")

Definition at line 204 of file [DFR0534.cpp](#).

```

00205 {
00206     if (m_ptrStream == NULL) return; // Should not happen
00207     sendStartingCode();
00208     sendDataByte(0x05);
00209     sendDataByte(0x00);
00210     sendChecksum();
00211 }
```

#### 4.1.4.25 prepareFileByNumber()

```
void DFR0534::prepareFileByNumber (
    word track)
```

Select file by number, but not start playing.

##### Parameters

in	<i>track</i>	Number for file
----	--------------	-----------------

Definition at line 972 of file [DFR0534.cpp](#).

```

00973 {
00974     if (m_ptrStream == NULL) return; // Should not happen
00975     sendStartingCode();
00976     sendDataByte(0x1F);
00977     sendDataByte(0x02);
00978     sendDataByte((track >> 8) & 0xff);
00979     sendDataByte(track & 0xff);
00980     sendChecksum();
00981 }
```

#### 4.1.4.26 repeatPart()

```
void DFR0534::repeatPart (
    byte startMinute,
    byte startSecond,
    byte stopMinute,
    byte stopSecond)
```

Repeat part of the current file.

Repeat between time start and stop position

##### Parameters

in	<i>startMinute</i>	Minute for start position
in	<i>startSecond</i>	Second for start position
in	<i>stopMinute</i>	Minute for stop position
in	<i>stopSecond</i>	Seconde for stop position

Definition at line 993 of file [DFR0534.cpp](#).

```
00994 {
00995     if (m_ptrStream == NULL) return; // Should not happen
00996     sendStartingCode();
00997     sendDataByte(0x20);
00998     sendDataByte(0x04);
00999     sendDataByte(startMinute);
01000     sendDataByte(startSecond);
01001     sendDataByte(stopMinute);
01002     sendDataByte(stopSecond);
01003     sendChecksum();
01004 }
```

#### 4.1.4.27 setChannel()

```
void DFR0534::setChannel (
    byte channel)
```

Set output for DAC to channel MP3, AUX or both.

I found not P26/P27 for AUX on my [DFR0534](#) => Only [DFR0534::CHANNELMP3](#) makes sense (and is already set by default) Perhaps this function works on other audio modules with the same chip.

##### Parameters

in	<i>channel</i>	Output channel: <a href="#">DFR0534::CHANNELMP3</a> , <a href="#">DFR0534::CHANNELAUX</a> or <a href="#">DFR0534::CHANNELMP3AUX</a>
----	----------------	---

Definition at line 893 of file [DFR0534.cpp](#).

```
00894 {
00895     if (m_ptrStream == NULL) return; // Should not happen
00896     if (channel >= CHANNELUNKNOWN) return;
00897     sendStartingCode();
00898     sendDataByte(0x1D);
00899     sendDataByte(0x01);
00900     sendDataByte(channel);
00901     sendChecksum();
00902 }
```

#### 4.1.4.28 setDirectory()

```
void DFR0534::setDirectory (
    char * path,
    byte drive = DRIVEFLASH)
```

Should set directory, but does not work for me.

## Parameters

in	<i>path</i>	Directory
in	<i>drive</i>	Drive, where directory is stored: Drive <a href="#">DFR0534::DRIVEUSB</a> , <a href="#">DFR0534::DRIVESD</a> or <a href="#">DFR0534::DRIVEFLASH</a> (=default)

Definition at line 798 of file [DFR0534.cpp](#).

```
00799 {
00800     if (m_ptrStream == NULL) return; // Should not happen
00801     if (path == NULL) return;
00802     if (drive >= DRIVEUNKNOWN) return;
00803     sendStartingCode();
00804     sendDataByte(0x17);
00805     sendDataByte(strlen(path)+1);
00806     sendDataByte(drive);
00807     for (int i=0;i<strlen(path);i++) {
00808         sendDataByte(path[i]);
00809     }
00810     sendChecksum();
00811 }
```

#### 4.1.4.29 setDrive()

```
void DFR0534::setDrive (
    byte drive)
```

Switch to drive.

## Parameters

in	<i>drive</i>	Drive <a href="#">DFR0534::DRIVEUSB</a> , <a href="#">DFR0534::DRIVESD</a> or <a href="#">DFR0534::DRIVEFLASH</a>
----	--------------	---

Definition at line 408 of file [DFR0534.cpp](#).

```
00409 {
00410     if (m_ptrStream == NULL) return; // Should not happen
00411     if (drive >= DRIVEUNKNOWN) return;
00412     sendStartingCode();
00413     sendDataByte(0x0B);
00414     sendDataByte(0x01);
00415     sendDataByte(drive);
00416     sendChecksum();
00417 }
```

#### 4.1.4.30 setEqualizer()

```
void DFR0534::setEqualizer (
    byte mode)
```

Set equalizer to NORMAL, POP, ROCK, JAZZ or CLASSIC.

## Parameters

in	<i>mode</i>	EQ mode: <a href="#">DFR0534::NORMAL</a> , <a href="#">DFR0534::POP</a> , <a href="#">DFR0534::ROCK</a> , <a href="#">DFR0534::JAZZ</a> or <a href="#">DFR0534::CLASSIC</a>
----	-------------	---

Definition at line 116 of file [DFR0534.cpp](#).

```
00117 {
00118     if (m_ptrStream == NULL) return; // Should not happen
00119     if (mode >= EQUNKNOWN) return;
00120     sendStartingCode();
00121     sendDataByte(0x1A);
00122     sendDataByte(0x01);
00123     sendDataByte(mode);
00124     sendChecksum();
00125 }
```

#### 4.1.4.31 setLoopMode()

```
void DFR0534::setLoopMode (
    byte mode)
```

Set loop mode.

##### Parameters

in	mode	Loop mode: <a href="#">DFR0534::LOOPBACKALL</a> , <a href="#">DFR0534::SINGLEAUDIOLOOP</a> , <a href="#">DFR0534::SINGLEAUDIOSTOP</a> , <a href="#">DFR0534::PLAYRANDOM</a> , <a href="#">DFR0534::DIRECTORYLOOP</a> , <a href="#">DFR0534::RANDOMINDIRECTORY</a> , <a href="#">DFR0534::SEQUENTIALINDIRECTORY</a> or <a href="#">DFR0534::SEQUENTIAL</a>
----	------	---

Definition at line 818 of file [DFR0534.cpp](#).

```
00819 {
00820     if (m_ptrStream == NULL) return; // Should not happen
00821     if (mode >= PLAYMODEUNKNOWN) return;
00822     sendStartingCode();
00823     sendDataByte(0x18);
00824     sendDataByte(0x01);
00825     sendDataByte(mode);
00826     sendChecksum();
00827 }
```

#### 4.1.4.32 setRepeatLoops()

```
void DFR0534::setRepeatLoops (
    word loops)
```

Set repeat loops.

Only valid for loop modes [DFR0534::LOOPBACKALL](#), [DFR0534::SINGLEAUDIOLOOP](#) or [DFR0534::DIRECTORYLOOP](#)

##### Parameters

in	loops	Number of loops
----	-------	-----------------

Definition at line 836 of file [DFR0534.cpp](#).

```
00837 {
00838     if (m_ptrStream == NULL) return; // Should not happen
00839     sendStartingCode();
00840     sendDataByte(0x19);
00841     sendDataByte(0x02);
00842     sendDataByte((loops >> 8) & 0xff);
00843     sendDataByte(loops & 0xff);
00844     sendChecksum();
00845 }
```

#### 4.1.4.33 setVolume()

```
void DFR0534::setVolume (
    byte volume)
```

Set volume.

Volumen levels 0-30 are allowed. Audio module starts always with level 20.

## Parameters

in	<i>volume</i>	Volume level
----	---------------	--------------

Definition at line 154 of file [DFR0534.cpp](#).

```
00155 {
00156     if (m_ptrStream == NULL) return; // Should not happen
00157     if (volume > 30) volume = 30;
00158     sendStartingCode();
00159     sendDataByte(0x13);
00160     sendDataByte(0x01);
00161     sendDataByte(volume);
00162     sendChecksum();
00163 }
```

#### 4.1.4.34 startSendingRuntime()

```
void DFR0534::startSendingRuntime ()
```

Start sending elapsed runtime every 1 second.

Definition at line 1136 of file [DFR0534.cpp](#).

```
01137 {
01138     if (m_ptrStream == NULL) return; // Should not happen
01139     sendStartingCode();
01140     sendDataByte(0x25);
01141     sendDataByte(0x00);
01142     sendChecksum();
01143 }
```

#### 4.1.4.35 stop()

```
void DFR0534::stop ()
```

Stop the current file.

Definition at line 192 of file [DFR0534.cpp](#).

```
00193 {
00194     if (m_ptrStream == NULL) return; // Should not happen
00195     sendStartingCode();
00196     sendDataByte(0x04);
00197     sendDataByte(0x00);
00198     sendChecksum();
00199 }
```

#### 4.1.4.36 stopCombined()

```
void DFR0534::stopCombined ()
```

Stop combined play (playlist)

Definition at line 876 of file [DFR0534.cpp](#).

```
00877 {
00878     if (m_ptrStream == NULL) return; // Should not happen
00879     sendStartingCode();
00880     sendDataByte(0x1C);
00881     sendDataByte(0x00);
00882     sendChecksum();
00883 }
```

#### 4.1.4.37 stopInsertedFile()

```
void DFR0534::stopInsertedFile ()
```

Stop inserted file.

Continue original file

Definition at line 783 of file [DFR0534.cpp](#).

```
00784 {  
00785     if (m_ptrStream == NULL) return; // Should not happen  
00786     sendStartingCode();  
00787     sendDataByte(0x10);  
00788     sendDataByte(0x00);  
00789     sendChecksum();  
00790 }
```

#### 4.1.4.38 stopRepeatPart()

```
void DFR0534::stopRepeatPart ()
```

Stop repeating part of the current file.

Definition at line 1009 of file [DFR0534.cpp](#).

```
01010 {  
01011     if (m_ptrStream == NULL) return; // Should not happen  
01012     sendStartingCode();  
01013     sendDataByte(0x21);  
01014     sendDataByte(0x00);  
01015     sendChecksum();  
01016 }
```

#### 4.1.4.39 stopSendingRuntime()

```
void DFR0534::stopSendingRuntime ()
```

Stop sending runtime.

Definition at line 1225 of file [DFR0534.cpp](#).

```
01226 {  
01227     if (m_ptrStream == NULL) return; // Should not happen  
01228     sendStartingCode();  
01229     sendDataByte(0x26);  
01230     sendDataByte(0x00);  
01231     sendChecksum();  
01232 }
```

The documentation for this class was generated from the following files:

- [DFR0534.h](#)
- [DFR0534.cpp](#)

# Chapter 5

## File Documentation

### 5.1 playCombined.ino

```
00001 /*
00002  * Example for using the DFR0534 for playing combined audio files like a playlist
00003  *
00004  * This example code was made for Arduino Uno/Nano/ATmega328p. For ESP32 you have to change the code
00005  * to use HardwareSerial (see https://github.com/codingABI/DFR0534#hardwareserial-for-esp32)
00006  */
00007
00008 #include <SoftwareSerial.h>
00009 #include <DFR0534.h>
00010
00011 #define TX_PIN A0
00012 #define RX_PIN A1
00013 SoftwareSerial g_serial(RX_PIN, TX_PIN);
00014 DFR0534 g_audio(g_serial);
00015
00016 void setup() {
00017     // Serial for console output
00018     Serial.begin(9600);
00019     // Software serial for communication to DFR0534 module
00020     g_serial.begin(9600);
00021
00022     // Set volume
00023     g_audio.setVolume(18);
00024
00025     /* The parameter string for the playCombined function is just
00026      * a concatenation of all files in the desired order without
00027      * path and without extension.
00028      * All files have to be in the folder /ZH and the each
00029      * file has to have a length (without extension) of two chars.
00030      *
00031      * You can get example files from
00032      * https://github.com/codingABI/DFR0534/tree/main/assets/exampleContent
00033      */
00034     /* Plays files the custom order, like a playlist and stops after the last file:
00035      * /ZH/05.wav
00036      * /ZH/04.wav
00037      * /ZH/03.wav
00038      * /ZH/02.wav
00039      * /ZH/01.wav
00040      * /ZH/0A.wav
00041      */
00042     g_audio.playCombined("05040302010A");
00043 }
00044
00045 void loop() {
00046     static unsigned long lastDisplayMS = millis();
00047     char name[12];
00048
00049     // Show information about current track every 500ms
00050     if (millis() - lastDisplayMS > 500) {
00051         Serial.print("number: ");
00052         word fileNumber = g_audio.getFileNumber();
00053         if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");
00054
00055         Serial.print(" name: ");
00056         if (g_audio.getFileName(name)) Serial.print(name);
```

```

00057
00058     Serial.print(" status: ");
00059     switch (g_audio.getStatus()) {
00060     case DFR0534::STOPPED:
00061         Serial.println("Stopped");
00062         break;
00063     case DFR0534::PAUSED:
00064         Serial.println("Paused");
00065         break;
00066     case DFR0534::PLAYING:
00067         Serial.println("Playing");
00068         break;
00069     case DFR0534::STATUSUNKNOWN:
00070         Serial.println("Unknown");
00071         break;
00072     }
00073     lastDisplayMS = millis();
00074 }
00075 }

```

## 5.2 playFileByName.ino

```

00001 /*
00002  * Example for using the DFR0534 for playing audio files by file name
00003  *
00004  * This example code was made for Arduino Uno/Nano/ATmega328p. For ESP32 you have to change the code
00005  * to use HardwareSerial
00006  * instead of SoftwareSerial (see https://github.com/codingABI/DFR0534#hardwareserial-for-esp32)
00007  */
00008 #include <SoftwareSerial.h>
00009 #include <DFR0534.h>
00010
00011 #define TX_PIN A0
00012 #define RX_PIN A1
00013 SoftwareSerial g_serial(RX_PIN, TX_PIN);
00014 DFR0534 g_audio(g_serial);
00015
00016 void setup() {
00017     // Serial for console output
00018     Serial.begin(9600);
00019     // Software serial for communication to DFR0534 module
00020     g_serial.begin(9600);
00021
00022     // Set volume
00023     g_audio.setVolume(18);
00024
00025     /* The file name/path for the function playFileByName() is the
00026      * full path of the audio file to be played in format which looks like
00027      * a special unix 8+3 format:
00028      * - Without a dot between file name and file extension
00029      * - All characters in upper case
00030      * - Maximal 8 characters for file name
00031      * - Every file and folder whose name length is shorter than 8 chars
00032      * - must be filled up to the 8 chars length by space chars
00033      * - Must end with WAV or MP3
00034      * - Only WAV and MP3 files are supported
00035      * - Wildcards * (=multiple arbitrary characters) and ? (=one single arbitrary character)
00036      * - are allowed and can be used to reduce the filling space chars
00037      *
00038      * Valid examples:
00039      * - "/01 WAV" for a file '/01.wav'
00040      * - "/99-AFR~1MP3" for a file '/99-Africa.mp3'
00041      * - "/SUN*MP3" for first file matching '/sun*.mp3', for example '/sun.mp3'
00042      * - "/99-AFR*MP3" for first file matching '/99-Afr*.mp3'
00043      * - "/10*" for first audio file matching '/10*.*'
00044      * - "/10 /20 WAV" for the file '/10/20.wav'
00045      * (first means first in "file copy order")
00046      *
00047      * You can get example files from
00048      * https://github.com/codingABI/DFR0534/tree/main/assets/exampleContent
00049      */
00050
00051     // Play the file "test.wav"
00052     g_audio.playFileByName("/TEST WAV");
00053 }
00054
00055 void loop() {
00056     static unsigned long lastDisplayMS = millis()-500;
00057     char name[12];
00058
00059     // Show information about current track once per second
00060     if (millis()-lastDisplayMS > 1000) {

```



```

00061     Serial.print("number: ");
00062     word fileNumber = g_audio.getFileNumber();
00063     if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");
00064
00065     Serial.print(" name: ");
00066     if (g_audio.getFileName(name)) Serial.print(name);
00067
00068     Serial.print(" status: ");
00069     switch (g_audio.getStatus()) {
00070     case DFR0534::STOPPED:
00071         Serial.println("Stopped");
00072         break;
00073     case DFR0534::PAUSED:
00074         Serial.println("Paused");
00075         break;
00076     case DFR0534::PLAYING:
00077         Serial.println("Playing");
00078         break;
00079     case DFR0534::STATUSUNKNOWN:
00080         Serial.println("Unknown");
00081         break;
00082     }
00083     lastDisplayMS = millis();
00084 }
00085 }

```

## 5.3 playFileByNumber.ino

```

00001 /*
00002  * Example for using the DFR0534 for playing audio files by file number
00003  *
00004  * This example code was made for Arduino Uno/Nano/ATmega328p. For ESP32 you have the change the code
00005  * to use HardwareSerial
00006  * instead of SoftwareSerial (see https://github.com/codingABI/DFR0534#hardwareserial-for-esp32)
00007  */
00008 #include <SoftwareSerial.h>
00009 #include <DFR0534.h>
00010
00011 #define TX_PIN A0
00012 #define RX_PIN A1
00013 SoftwareSerial g_serial(RX_PIN, TX_PIN);
00014 DFR0534 g_audio(g_serial);
00015
00016 void setup() {
00017     // Serial for console output
00018     Serial.begin(9600);
00019     // Software serial for communication to DFR0534 module
00020     g_serial.begin(9600);
00021
00022     // Set volume
00023     g_audio.setVolume(18);
00024
00025     // Show some device infos
00026     Serial.print("Ready drives: ");
00027     byte drive = g_audio.getDrivesStates();
00028     if ((drive >> DFR0534::DRIVEUSB) & 1) == 1) Serial.print("USB ");
00029     if ((drive >> DFR0534::DRIVESD) & 1) == 1) Serial.print("SD ");
00030     if ((drive >> DFR0534::DRIVEFLASH) & 1) == 1) Serial.print("FLASH ");
00031     Serial.println();
00032
00033     Serial.print("Current playing drive: ");
00034     switch(g_audio.getDrive()) {
00035     case DFR0534::DRIVEUSB:
00036         Serial.println("USB");
00037         break;
00038     case DFR0534::DRIVESD:
00039         Serial.println("SD");
00040         break;
00041     case DFR0534::DRIVEFLASH:
00042         Serial.println("FLASH");
00043         break;
00044     case DFR0534::DRIVENO:
00045         Serial.println("No drive");
00046         break;
00047     default:
00048         Serial.println("Unknown");
00049         break;
00050     }
00051
00052     Serial.print("Total files: ");
00053     Serial.println(g_audio.getTotalFiles());
00054     Serial.print("Total files in directory: ");

```

```

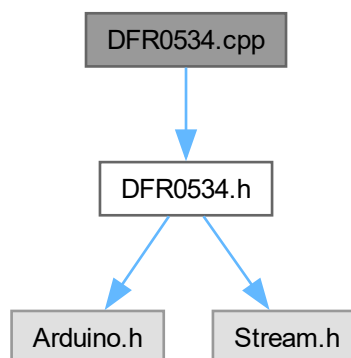
00055 Serial.println(g_audio.getTotalFilesInCurrentDirectory());
00056
00057 Serial.print("First file: ");
00058 Serial.println(g_audio.getFirstFileNumberInCurrentDirectory());
00059
00060 // Play the first audio file copied to the DFR0534
00061 // (Second file copied to the DFR0534 would be number 2...)
00062 g_audio.playFileByNumber(1);
00063 }
00064
00065 void loop() {
00066     static unsigned long lastDisplayMS = millis()-500;
00067     char name[12];
00068
00069     // Show information about current track once per second
00070     if (millis()-lastDisplayMS > 1000) {
00071         Serial.print("number: ");
00072         word fileNumber = g_audio.getFileNumber();
00073         if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");
00074
00075         Serial.print(" name: ");
00076         if (g_audio.getFileName(name)) Serial.print(name);
00077
00078         Serial.print(" status: ");
00079         switch (g_audio.getStatus()) {
00080             case DFR0534::STOPPED:
00081                 Serial.println("Stopped");
00082                 break;
00083             case DFR0534::PAUSED:
00084                 Serial.println("Paused");
00085                 break;
00086             case DFR0534::PLAYING:
00087                 Serial.println("Playing");
00088                 break;
00089             case DFR0534::STATUSUNKNOWN:
00090                 Serial.println("Unknown");
00091                 break;
00092         }
00093         lastDisplayMS = millis();
00094     }
00095 }

```

## 5.4 DFR0534.cpp File Reference

#include "DFR0534.h"

Include dependency graph for DFR0534.cpp:



### 5.4.1 Detailed Description

Class: [DFR0534](#)

Description: Class for controlling a [DFR0534](#) audio module ( [https://wiki.dfrobot.com/Voice\\_Module\\_SKU\\_\\_DFR0534](https://wiki.dfrobot.com/Voice_Module_SKU__DFR0534)) by SoftwareSerial or HardwareSerial

License: 2-Clause BSD License Copyright (c) 2024 codingABI For details see: LICENSE.txt

Notes for [DFR0534](#) audio module:

- Consumes about 20mA when idle ( $V_{cc} = 5V$ )
- Creates a short "click" noise, when  $V_{cc}$  is powered on
- Should be used with a 1k resistor on TX when your MCU runs on 5V, because the [DFR0534](#) uses 3.3V logic (and 5V on TX causes clicks/noise)
- Can be controlled by a RX/TX serial connection (9600 baud) or one-wire protocol
- Can play WAV and MP3 audiofiles
- Can "insert" audiofiles while another audiofile is running. In this case the original audiofile is paused and will be resumed after the "inserted" audiofile
- Can play files in a playlist like mode called "combined" for files stored in a directory /ZH
- Can select the file to play by a file number\* or file name\*\* \*File number is independent from file name. The first WAV or MP3 copied to the [DFR0534](#) gets file number 1 and so on. To play a file by number use `playFileByNumber()` \*\*File name is a little bit like a 8+3 file path and can be used with `playFileByName()`, but have special rules (see `playFileByName()` for details)
- Can send automatically the file runtime every second (when enabled)
- Has a NS8002 amplifier, JQ8400 Audio chip, W25Q64JVSIQ flash memory
- Has a Sleep mode 0x1B and this mode only works with one-wire protocol ( [https://github.com/arduino12/mp3\\_player\\_module\\_wire](https://github.com/arduino12/mp3_player_module_wire)) and does not work for me without additional electric modifications (e.g. disconnecting speakers) => Switching off [DFR0534](#) with a FET is a better solution

Home: <https://github.com/codingABI/DFR0534>

#### Author

codingABI <https://github.com/codingABI/>

#### Copyright

2-Clause BSD License

#### Version

1.0.4

Definition in file [DFR0534.cpp](#).

## 5.5 DFR0534.cpp

[Go to the documentation of this file.](#)

```

00001
00043 #include "DFR0534.h"
00044
00053 byte DFR0534::getStatus()
00054 {
00055     #define COMMAND 0x01
00056     #define RECEIVEBYTETIMEOUTMS 100
00057     #define RECEIVEGLOBALTIMEOUTMS 500
00058     #define RECEIVEFAILED STATUSUNKNOWN
00059     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00060
00061     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00062     sendStartingCode();
00063     sendDataByte(COMMAND);
00064     sendDataByte(0x00);
00065     sendChecksum();
00066
00067     // Receive
00068     int i=0;
00069     byte data, firstByte = 0, sum, length=0xff, result = 0;
00070     unsigned long receiveStartMS = millis();
00071     do {
00072         byte dataReady = 0;
00073         unsigned long lastMS = millis();
00074         // Wait for response or timeout
00075         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00076
00077         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00078         data = m_ptrStream->read();
00079
00080         if (i==0) { // Begin of transmission
00081             firstByte=data;
00082             sum = 0;
00083         }
00084         if ((i == 1) && (data != COMMAND)) {
00085             // Invalid signal => reset receive
00086             i=0;
00087             firstByte = 0;
00088         }
00089         if (i == RECEIVEHEADERLENGTH) {
00090             length = data; // Length of receiving data
00091             if (length != 1) {
00092                 // Invalid length => reset receive
00093                 i=0;
00094                 firstByte = 0;
00095             }
00096         }
00097         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00098             result = data;
00099         }
00100         if (firstByte == STARTINGCODE) {
00101             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00102             i++;
00103         }
00104         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00105     } while (i<length+RECEIVEHEADERLENGTH+2);
00106
00107     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00108     return result;
00109 }
00110
00116 void DFR0534::setEqualizer(byte mode)
00117 {
00118     if (m_ptrStream == NULL) return; // Should not happen
00119     if (mode >= EQUNKNOWN) return;
00120     sendStartingCode();
00121     sendDataByte(0x1A);
00122     sendDataByte(0x01);
00123     sendDataByte(mode);
00124     sendChecksum();
00125 }
00126
00135 void DFR0534::playFileByNumber(word track)
00136 {
00137     if (m_ptrStream == NULL) return; // Should not happen
00138     if (track <=0) return;
00139     sendStartingCode();
00140     sendDataByte(0x07);
00141     sendDataByte(0x02);
00142     sendDataByte((track » 8) & 0xff);
00143     sendDataByte(track & 0xff);

```

```

00144     sendChecksum();
00145 }
00146
00154 void DFR0534::setVolume(byte volume)
00155 {
00156     if (m_ptrStream == NULL) return; // Should not happen
00157     if (volume > 30) volume = 30;
00158     sendStartingCode();
00159     sendDataByte(0x13);
00160     sendDataByte(0x01);
00161     sendDataByte(volume);
00162     sendChecksum();
00163 }
00164
00168 void DFR0534::play()
00169 {
00170     if (m_ptrStream == NULL) return; // Should not happen
00171     sendStartingCode();
00172     sendDataByte(0x02);
00173     sendDataByte(0x00);
00174     sendChecksum();
00175 }
00176
00180 void DFR0534::pause()
00181 {
00182     if (m_ptrStream == NULL) return; // Should not happen
00183     sendStartingCode();
00184     sendDataByte(0x03);
00185     sendDataByte(0x00);
00186     sendChecksum();
00187 }
00188
00192 void DFR0534::stop()
00193 {
00194     if (m_ptrStream == NULL) return; // Should not happen
00195     sendStartingCode();
00196     sendDataByte(0x04);
00197     sendDataByte(0x00);
00198     sendChecksum();
00199 }
00200
00204 void DFR0534::playPrevious()
00205 {
00206     if (m_ptrStream == NULL) return; // Should not happen
00207     sendStartingCode();
00208     sendDataByte(0x05);
00209     sendDataByte(0x00);
00210     sendChecksum();
00211 }
00212
00216 void DFR0534::playNext()
00217 {
00218     if (m_ptrStream == NULL) return; // Should not happen
00219     sendStartingCode();
00220     sendDataByte(0x06);
00221     sendDataByte(0x00);
00222     sendChecksum();
00223 }
00224
00252 void DFR0534::playFileByName(char *path, byte drive)
00253 {
00254     if (m_ptrStream == NULL) return; // Should not happen
00255     if (path == NULL) return;
00256     if (drive >= DRIVEUNKNOWN) return;
00257     sendStartingCode();
00258     sendDataByte(0x08);
00259     sendDataByte(strlen(path)+1);
00260     sendDataByte(drive);
00261     for (int i=0; i<strlen(path); i++) {
00262         sendDataByte(path[i]);
00263     }
00264     sendChecksum();
00265 }
00266
00278 byte DFR0534::getDrivesStates()
00279 {
00280     #define COMMAND 0x09
00281     #define RECEIVEBYTETIMEOUTMS 100
00282     #define RECEIVEGLOBALTIMEOUTMS 500
00283     #define RECEIVEFAILED DRIVEUNKNOWN
00284     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00285
00286     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00287     sendStartingCode();
00288     sendDataByte(COMMAND);
00289     sendDataByte(0x00);
00290     sendChecksum();

```

```

00291
00292 // Receive
00293 int i=0;
00294 byte data, firstByte = 0, sum, length=0xff, result = 0;
00295 unsigned long receiveStartMS = millis();
00296 do {
00297     byte dataReady = 0;
00298     unsigned long lastMS = millis();
00299     // Wait for response or timeout
00300     while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00301
00302     if (dataReady == 0) return RECEIVEFAILED; // Timeout
00303     data = m_ptrStream->read();
00304
00305     if (i==0) { // Begin of transmission
00306         firstByte=data;
00307         sum = 0;
00308     }
00309     if ((i == 1) && (data != COMMAND)) {
00310         // Invalid signal => reset receive
00311         i=0;
00312         firstByte = 0;
00313     }
00314     if (i == RECEIVEHEADERLENGTH) {
00315         length = data; // Length of receiving data
00316         if (length != 1) {
00317             // Invalid length => reset receive
00318             i=0;
00319             firstByte = 0;
00320         }
00321     }
00322     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00323         result = data;
00324     }
00325     if (firstByte == STARTINGCODE) {
00326         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00327         i++;
00328     }
00329     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00330 } while (i<length+RECEIVEHEADERLENGTH+2);
00331
00332 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00333 return result;
00334 }
00335
00345 byte DFR0534::getDrive()
00346 {
00347     #define COMMAND 0x0A
00348     #define RECEIVEBYTETIMEOUTMS 100
00349     #define RECEIVEGLOBALTIMEOUTMS 500
00350     #define RECEIVEFAILED DRIVEUNKNOWN
00351     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00352
00353     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00354     sendStartingCode();
00355     sendDataByte(COMMAND);
00356     sendDataByte(0x00);
00357     sendChecksum();
00358
00359     // Receive
00360     int i=0;
00361     byte data, firstByte = 0, sum, length=0xff, result = 0;
00362     unsigned long receiveStartMS = millis();
00363     do {
00364         byte dataReady = 0;
00365         unsigned long lastMS = millis();
00366         // Wait for response or timeout
00367         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00368
00369         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00370         data = m_ptrStream->read();
00371
00372         if (i==0) { // Begin of transmission
00373             firstByte=data;
00374             sum = 0;
00375         }
00376         if ((i == 1) && (data != COMMAND)) {
00377             // Invalid signal => reset receive
00378             i=0;
00379             firstByte = 0;
00380         }
00381         if (i == RECEIVEHEADERLENGTH) {
00382             length = data; // Length of receiving data
00383             if (length != 1) {
00384                 // Invalid length => reset receive

```

```

00385         i=0;
00386         firstByte = 0;
00387     }
00388 }
00389 if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00390     result = data;
00391 }
00392 if (firstByte == STARTINGCODE) {
00393     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00394     i++;
00395 }
00396 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00397 } while (i<length+RECEIVEHEADERLENGTH+2);
00398
00399 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00400 return result;
00401 }
00402
00408 void DFR0534::setDrive(byte drive)
00409 {
00410     if (m_ptrStream == NULL) return; // Should not happen
00411     if (drive >= DRIVEUNKNOWN) return;
00412     sendStartingCode();
00413     sendDataByte(0x0B);
00414     sendDataByte(0x01);
00415     sendDataByte(drive);
00416     sendChecksum();
00417 }
00418
00427 word DFR0534::getFileNumber()
00428 {
00429     #define COMMAND 0x0D
00430     #define RECEIVEFAILED 0
00431     #define RECEIVEBYTETIMEOUTMS 100
00432     #define RECEIVEGLOBALTIMEOUTMS 500
00433     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00434
00435     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00436     sendStartingCode();
00437     sendDataByte(COMMAND);
00438     sendDataByte(0x00);
00439     sendChecksum();
00440
00441     // Receive
00442     int i=0;
00443     byte data, firstByte = 0, sum, length=0xff;
00444     word result = 0;
00445     unsigned long receiveStartMS = millis();
00446     do {
00447         byte dataReady = 0;
00448         unsigned long lastMS = millis();
00449         // Wait for response or timeout
00450         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00451
00452         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00453         data = m_ptrStream->read();
00454
00455         if (i==0) { // Begin of transmission
00456             firstByte=data;
00457             sum = 0;
00458         }
00459         if ((i == 1) && (data != COMMAND)) {
00460             // Invalid signal => reset receive
00461             i=0;
00462             firstByte = 0;
00463         }
00464         if (i == RECEIVEHEADERLENGTH) {
00465             length = data; // Length of receiving data
00466             if (length != 2) {
00467                 // Invalid length => reset receive
00468                 i=0;
00469                 firstByte = 0;
00470             }
00471         }
00472         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00473             switch (i-RECEIVEHEADERLENGTH-1) {
00474                 case 0:
00475                     result=data<<8;
00476                     break;
00477                 case 1:
00478                     result+=data;
00479                     break;
00480             }
00481         }
00482         if (firstByte == STARTINGCODE) {
00483             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum

```

```

00484         i++;
00485     }
00486     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00487 } while (i<length+RECEIVEHEADERLENGTH+2);
00488
00489 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00490 return result;
00491 }
00492
00493 int DFR0534::getTotalFiles()
00494 {
00495     #define COMMAND 0x0C
00496     #define RECEIVEFAILED -1
00497     #define RECEIVEBYTETIMEOUTMS 100
00498     #define RECEIVEGLOBALTIMEOUTMS 500
00499     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00500
00501     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00502     sendStartingCode();
00503     sendDataByte(COMMAND);
00504     sendDataByte(0x00);
00505     sendChecksum();
00506
00507     // Receive
00508     int i=0;
00509     byte data, firstByte = 0, sum, length=0xff;
00510     word result = 0;
00511     unsigned long receiveStartMS = millis();
00512     do {
00513         byte dataReady = 0;
00514         unsigned long lastMS = millis();
00515         // Wait for response or timeout
00516         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
00517             m_ptrStream->available();
00518
00519         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00520         data = m_ptrStream->read();
00521
00522         if (i==0) { // Begin of transmission
00523             firstByte=data;
00524             sum = 0;
00525         }
00526         if ((i == 1) && (data != COMMAND)) {
00527             // Invalid signal => reset receive
00528             i=0;
00529             firstByte = 0;
00530         }
00531         if (i == RECEIVEHEADERLENGTH) {
00532             length = data; // Length of receiving data
00533             if (length != 2) {
00534                 // Invalid length => reset receive
00535                 i=0;
00536                 firstByte = 0;
00537             }
00538             if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00539                 switch (i-RECEIVEHEADERLENGTH-1) {
00540                     case 0:
00541                         result=data<<8;
00542                         break;
00543                     case 1:
00544                         result+=data;
00545                         break;
00546                 }
00547             }
00548             if (firstByte == STARTINGCODE) {
00549                 if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00550                 i++;
00551             }
00552             if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00553         } while (i<length+RECEIVEHEADERLENGTH+2);
00554
00555         if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00556         return result;
00557     }
00558 }
00559
00560 void DFR0534::playLastInDirectory()
00561 {
00562     if (m_ptrStream == NULL) return; // Should not happen
00563     sendStartingCode();
00564     sendDataByte(0x0E);
00565     sendDataByte(0x00);
00566     sendChecksum();
00567 }
00568
00569 void DFR0534::playNextDirectory()
00570 {

```



```

00582     if (m_ptrStream == NULL) return; // Should not happen
00583     sendStartingCode();
00584     sendDataByte(0x0F);
00585     sendDataByte(0x00);
00586     sendChecksum();
00587 }
00588
00595 int DFR0534::getFirstFileNumberInCurrentDirectory()
00596 {
00597     #define COMMAND 0x11
00598     #define RECEIVEFAILED -1
00599     #define RECEIVEBYTETIMEOUTMS 100
00600     #define RECEIVEGLOBALTIMEOUTMS 500
00601     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00602
00603     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00604     sendStartingCode();
00605     sendDataByte(COMMAND);
00606     sendDataByte(0x00);
00607     sendChecksum();
00608
00609     // Receive
00610     int i=0;
00611     byte data, firstByte = 0, sum, length=0xff;
00612     word result = 0;
00613     unsigned long receiveStartMS = millis();
00614     do {
00615         byte dataReady = 0;
00616         unsigned long lastMS = millis();
00617         // Wait for response or timeout
00618         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00619
00620         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00621         data = m_ptrStream->read();
00622
00623         if (i==0) { // Begin of transmission
00624             firstByte=data;
00625             sum = 0;
00626         }
00627         if ((i == 1) && (data != COMMAND)) {
00628             // Invalid signal => reset receive
00629             i=0;
00630             firstByte = 0;
00631         }
00632         if (i == RECEIVEHEADERLENGTH) {
00633             length = data; // Length of receiving data
00634             if (length != 2) {
00635                 // Invalid length => reset receive
00636                 i=0;
00637                 firstByte = 0;
00638             }
00639         }
00640         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00641             switch (i-RECEIVEHEADERLENGTH-1) {
00642                 case 0:
00643                     result=data<<8;
00644                     break;
00645                 case 1:
00646                     result+=data;
00647                     break;
00648             }
00649         }
00650         if (firstByte == STARTINGCODE) {
00651             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00652             i++;
00653         }
00654         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00655     } while (i<length+RECEIVEHEADERLENGTH+2);
00656
00657     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00658     return result;
00659 }
00660
00667 int DFR0534::getTotalFilesInCurrentDirectory()
00668 {
00669     #define COMMAND 0x12
00670     #define RECEIVEFAILED -1
00671     #define RECEIVEBYTETIMEOUTMS 100
00672     #define RECEIVEGLOBALTIMEOUTMS 500
00673     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00674
00675     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00676     sendStartingCode();
00677     sendDataByte(COMMAND);
00678     sendDataByte(0x00);
00679     sendChecksum();

```

```

00680
00681 // Receive
00682 int i=0;
00683 byte data, firstByte = 0, sum, length=0xff;
00684 word result = 0;
00685 unsigned long receiveStartMS = millis();
00686 do {
00687     byte dataReady = 0;
00688     unsigned long lastMS = millis();
00689     // Wait for response or timeout
00690     while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00691
00692     if (dataReady == 0) return RECEIVEFAILED; // Timeout
00693     data = m_ptrStream->read();
00694
00695     if (i==0) { // Begin of transmission
00696         firstByte=data;
00697         sum = 0;
00698     }
00699     if ((i == 1) && (data != COMMAND)) {
00700         // Invalid signal => reset receive
00701         i=0;
00702         firstByte = 0;
00703     }
00704     if (i == RECEIVEHEADERLENGTH) {
00705         length = data; // Length of receiving data
00706         if (length != 2) {
00707             // Invalid length => reset receive
00708             i=0;
00709             firstByte = 0;
00710         }
00711     }
00712     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00713         switch (i-RECEIVEHEADERLENGTH-1) {
00714             case 0:
00715                 result=data<<8;
00716                 break;
00717             case 1:
00718                 result+=data;
00719                 break;
00720         }
00721     }
00722     if (firstByte == STARTINGCODE) {
00723         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00724         i++;
00725     }
00726     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00727 } while (i<length+RECEIVEHEADERLENGTH+2);
00728
00729 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00730 return result;
00731 }
00732
00736 void DFR0534::increaseVolume()
00737 {
00738     if (m_ptrStream == NULL) return; // Should not happen
00739     sendStartingCode();
00740     sendDataByte(0x14);
00741     sendDataByte(0x00);
00742     sendChecksum();
00743 }
00744
00748 void DFR0534::decreaseVolume()
00749 {
00750     if (m_ptrStream == NULL) return; // Should not happen
00751     sendStartingCode();
00752     sendDataByte(0x15);
00753     sendDataByte(0x00);
00754     sendChecksum();
00755 }
00756
00765 void DFR0534::insertFileByNumber(word track, byte drive)
00766 {
00767     if (m_ptrStream == NULL) return; // Should not happen
00768     if (drive >= DRIVEUNKNOWN) return;
00769     sendStartingCode();
00770     sendDataByte(0x16);
00771     sendDataByte(0x03);
00772     sendDataByte(drive);
00773     sendDataByte((track >> 8) & 0xff);
00774     sendDataByte(track & 0xff);
00775     sendChecksum();
00776 }
00777
00783 void DFR0534::stopInsertedFile()
00784 {

```

```

00785     if (m_ptrStream == NULL) return; // Should not happen
00786     sendStartingCode();
00787     sendDataByte(0x10);
00788     sendDataByte(0x00);
00789     sendChecksum();
00790 }
00791
00798 void DFR0534::setDirectory(char *path, byte drive)
00799 {
00800     if (m_ptrStream == NULL) return; // Should not happen
00801     if (path == NULL) return;
00802     if (drive >= DRIVEUNKNOWN) return;
00803     sendStartingCode();
00804     sendDataByte(0x17);
00805     sendDataByte(strlen(path)+1);
00806     sendDataByte(drive);
00807     for (int i=0;i<strlen(path);i++) {
00808         sendDataByte(path[i]);
00809     }
00810     sendChecksum();
00811 }
00812
00818 void DFR0534::setLoopMode(byte mode)
00819 {
00820     if (m_ptrStream == NULL) return; // Should not happen
00821     if (mode >= PLAYMODEUNKNOWN) return;
00822     sendStartingCode();
00823     sendDataByte(0x18);
00824     sendDataByte(0x01);
00825     sendDataByte(mode);
00826     sendChecksum();
00827 }
00828
00836 void DFR0534::setRepeatLoops(word loops)
00837 {
00838     if (m_ptrStream == NULL) return; // Should not happen
00839     sendStartingCode();
00840     sendDataByte(0x19);
00841     sendDataByte(0x02);
00842     sendDataByte((loops >> 8) & 0xff);
00843     sendDataByte(loops & 0xff);
00844     sendChecksum();
00845 }
00846
00858 void DFR0534::playCombined(char* list)
00859 {
00860     if (m_ptrStream == NULL) return; // Should not happen
00861     if (list == NULL) return;
00862     if ((strlen(list) % 2) != 0) return;
00863     sendStartingCode();
00864     sendDataByte(0x1B);
00865     sendDataByte(strlen(list));
00866     for (int i=0;i<strlen(list);i++) {
00867         sendDataByte(list[i]);
00868     }
00869     sendChecksum();
00870 }
00871
00872
00876 void DFR0534::stopCombined()
00877 {
00878     if (m_ptrStream == NULL) return; // Should not happen
00879     sendStartingCode();
00880     sendDataByte(0x1C);
00881     sendDataByte(0x00);
00882     sendChecksum();
00883 }
00884
00893 void DFR0534::setChannel(byte channel)
00894 {
00895     if (m_ptrStream == NULL) return; // Should not happen
00896     if (channel >= CHANNELUNKNOWN) return;
00897     sendStartingCode();
00898     sendDataByte(0x1D);
00899     sendDataByte(0x01);
00900     sendDataByte(channel);
00901     sendChecksum();
00902 }
00903
00913 bool DFR0534::getFileName(char *name)
00914 {
00915     #define COMMAND 0x1E
00916     #define RECEIVEBYTETIMEOUTMS 100
00917     #define RECEIVEGLOBALTIMEOUTMS 500
00918     #define RECEIVEFAILED false
00919     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00920

```

```

00921     if (m_ptrStream == NULL) return false; // Should not happen
00922     if (name == NULL) return false;
00923     name[0] = '\0';
00924
00925     sendStartingCode();
00926     sendDataByte(COMMAND);
00927     sendDataByte(0x00);
00928     sendChecksum();
00929
00930     // Receive
00931     int i=0;
00932     byte data, firstByte = 0, sum, length=0xff;
00933     unsigned long receiveStartMS = millis();
00934     do {
00935         byte dataReady = 0;
00936         unsigned long lastMS = millis();
00937         // Wait for response or timeout
00938         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00939
00940         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00941         data = m_ptrStream->read();
00942         if (i==0) { // Begin of transmission
00943             firstByte=data;
00944             sum = 0;
00945         }
00946         if ((i == 1) && (data != COMMAND)) {
00947             // Invalid signal => reset receive
00948             i=0;
00949             firstByte = 0;
00950         }
00951         if (i == RECEIVEHEADERLENGTH) length = data; // Length of receiving string
00952         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00953             if ((i-RECEIVEHEADERLENGTH) < 12) { // I expect no longer file names than 8+3 chars plus '\0'
00954                 name[i-RECEIVEHEADERLENGTH-1] = data;
00955                 name[i-RECEIVEHEADERLENGTH] = '\0';
00956             }
00957         }
00958         if (firstByte == STARTINGCODE) {
00959             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00960             i++;
00961         }
00962         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00963     } while (i<length+RECEIVEHEADERLENGTH+2);
00964     return (data == sum); // Does checksum matches?
00965 }
00966
00972 void DFR0534::prepareFileByNumber(word track)
00973 {
00974     if (m_ptrStream == NULL) return; // Should not happen
00975     sendStartingCode();
00976     sendDataByte(0x1F);
00977     sendDataByte(0x02);
00978     sendDataByte((track » 8) & 0xff);
00979     sendDataByte(track & 0xff);
00980     sendChecksum();
00981 }
00982
00993 void DFR0534::repeatPart(byte startMinute, byte startSecond, byte stopMinute, byte stopSecond )
00994 {
00995     if (m_ptrStream == NULL) return; // Should not happen
00996     sendStartingCode();
00997     sendDataByte(0x20);
00998     sendDataByte(0x04);
00999     sendDataByte(startMinute);
01000     sendDataByte(startSecond);
01001     sendDataByte(stopMinute);
01002     sendDataByte(stopSecond);
01003     sendChecksum();
01004 }
01005
01009 void DFR0534::stopRepeatPart()
01010 {
01011     if (m_ptrStream == NULL) return; // Should not happen
01012     sendStartingCode();
01013     sendDataByte(0x21);
01014     sendDataByte(0x00);
01015     sendChecksum();
01016 }
01017
01025 void DFR0534::fastBackwardDuration(word seconds)
01026 {
01027     if (m_ptrStream == NULL) return; // Should not happen
01028     sendStartingCode();
01029     sendDataByte(0x22);
01030     sendDataByte(0x02);
01031     sendDataByte((seconds » 8) & 0xff);

```

```

01032     sendDataByte(seconds & 0xff);
01033     sendChecksum();
01034 }
01035
01042 void DFR0534::fastForwardDuration(word seconds)
01043 {
01044     if (m_ptrStream == NULL) return; // Should not happen
01045     sendStartingCode();
01046     sendDataByte(0x23);
01047     sendDataByte(0x02);
01048     sendDataByte((seconds >> 8) & 0xff);
01049     sendDataByte(seconds & 0xff);
01050     sendChecksum();
01051 }
01052
01065 bool DFR0534::getDuration(byte &hour, byte &minute, byte &second)
01066 {
01067     #define COMMAND 0x24
01068     #define RECEIVEFAILED false
01069     #define RECEIVEBYTETIMEOUTMS 100
01070     #define RECEIVEGLOBALTIMEOUTMS 500
01071     #define RECEIVEHEADERLENGTH 2 // startingcode+command
01072
01073     if (m_ptrStream == NULL) return false; // Should not happen
01074     sendStartingCode();
01075     sendDataByte(COMMAND);
01076     sendDataByte(0x00);
01077     sendChecksum();
01078
01079     // Receive
01080     int i=0;
01081     byte data, firstByte = 0, sum, length=0xff;
01082     word result = 0;
01083     unsigned long receiveStartMS = millis();
01084     do {
01085         byte dataReady = 0;
01086         unsigned long lastMS = millis();
01087         // Wait for response or timeout
01088         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01089
01090         if (dataReady == 0) return RECEIVEFAILED; // Timeout
01091         data = m_ptrStream->read();
01092
01093         if (i==0) { // Begin of transmission
01094             firstByte=data;
01095             sum = 0;
01096         }
01097         if ((i == 1) && (data != COMMAND)) {
01098             // Invalid signal => reset receive
01099             i=0;
01100             firstByte = 0;
01101         }
01102         if (i == RECEIVEHEADERLENGTH) {
01103             length = data; // Length of receiving data
01104             if (length != 3) {
01105                 // Invalid length => reset receive
01106                 i=0;
01107                 firstByte = 0;
01108             }
01109         }
01110         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01111             switch (i-RECEIVEHEADERLENGTH-1) {
01112                 case 0:
01113                     hour=data;
01114                     break;
01115                 case 1:
01116                     minute=data;
01117                     break;
01118                 case 2:
01119                     second=data;
01120                     break;
01121             }
01122         }
01123         if (firstByte == STARTINGCODE) {
01124             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01125             i++;
01126         }
01127         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01128     } while (i<length+RECEIVEHEADERLENGTH+2);
01129
01130     return (data == sum); // Does checksum matches?
01131 }
01132
01136 void DFR0534::startSendingRuntime()
01137 {
01138     if (m_ptrStream == NULL) return; // Should not happen

```

```

01139     sendStartingCode();
01140     sendDataByte(0x25);
01141     sendDataByte(0x00);
01142     sendChecksum();
01143 }
01144
01158 bool DFR0534::getRuntime(byte &hour, byte &minute, byte &second)
01159 {
01160     #define COMMAND 0x25
01161     #define RECEIVEFAILED false
01162     #define RECEIVEBYTETIMEOUTMS 100
01163     #define RECEIVEGLOBALTIMEOUTMS 500
01164     #define RECEIVEHEADERLENGTH 2 // startingcode+command
01165
01166     if (m_ptrStream == NULL) return false; // Should not happen
01167
01168     // Receive
01169     int i=0;
01170     byte data, firstByte = 0, sum, length=0xff;
01171     word result = 0;
01172     unsigned long receiveStartMS = millis();
01173     do {
01174         byte dataReady = 0;
01175         unsigned long lastMS = millis();
01176         // Wait for response or timeout
01177         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01178
01179         if (dataReady == 0) return RECEIVEFAILED; // Timeout
01180         data = m_ptrStream->read();
01181
01182         if (i==0) { // Begin of transmission
01183             firstByte=data;
01184             sum = 0;
01185         }
01186         if ((i == 1) && (data != COMMAND)) {
01187             // Invalid signal => reset receive
01188             i=0;
01189             firstByte = 0;
01190         }
01191         if (i == RECEIVEHEADERLENGTH) {
01192             length = data; // Length of receiving data
01193             if (length != 3) {
01194                 // Invalid length => reset receive
01195                 i=0;
01196                 firstByte = 0;
01197             }
01198         }
01199         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01200             switch (i-RECEIVEHEADERLENGTH-1) {
01201                 case 0:
01202                     hour=data;
01203                     break;
01204                 case 1:
01205                     minute=data;
01206                     break;
01207                 case 2:
01208                     second=data;
01209                     break;
01210             }
01211         }
01212         if (firstByte == STARTINGCODE) {
01213             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01214             i++;
01215         }
01216         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01217     } while (i<length+RECEIVEHEADERLENGTH+2);
01218
01219     return (data == sum); // Does checksum matches?
01220 }
01221
01225 void DFR0534::stopSendingRuntime()
01226 {
01227     if (m_ptrStream == NULL) return; // Should not happen
01228     sendStartingCode();
01229     sendDataByte(0x26);
01230     sendDataByte(0x00);
01231     sendChecksum();
01232 }

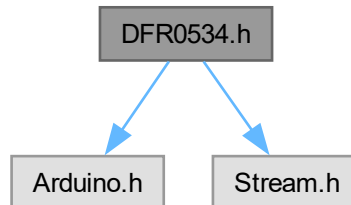
```

## 5.6 DFR0534.h File Reference

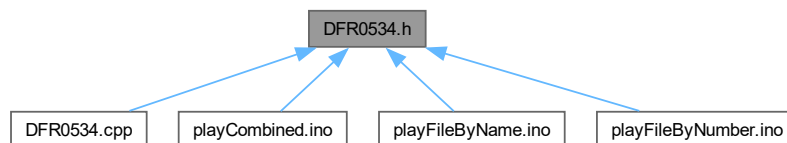
```
#include <Arduino.h>
```

```
#include <Stream.h>
```

Include dependency graph for DFR0534.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [DFR0534](#)  
*Class for a [DFR0534](#) audio module.*

### Macros

- `#define DFR0534\_VERSION "1.0.4"`

### 5.6.1 Detailed Description

Class: [DFR0534](#)

Description: Class for controlling a [DFR0534](#) audio module ( [https://wiki.dfrobot.com/Voice\\_Module\\_SKU\\_\\_DFR0534](https://wiki.dfrobot.com/Voice_Module_SKU__DFR0534)) by SoftwareSerial or HardwareSerial

License: 2-Clause BSD License Copyright (c) 2024 codingABI For details see: LICENSE.txt

Home: <https://github.com/codingABI/DFR0534>

**Author**

codingABI <https://github.com/codingABI/>

**Copyright**

2-Clause BSD License

**Version**

1.0.4

Definition in file [DFR0534.h](#).

## 5.6.2 Macro Definition Documentation

### 5.6.2.1 DFR0534\_VERSION

```
#define DFR0534_VERSION "1.0.4"
```

Library version

Definition at line 22 of file [DFR0534.h](#).

## 5.7 DFR0534.h

[Go to the documentation of this file.](#)

```
00001
00019 #pragma once
00020
00022 #define DFR0534_VERSION "1.0.4"
00023
00024 #include <Arduino.h>
00025 #include <Stream.h>
00026
00027 #define STARTINGCODE 0xAA
00028
00032 class DFR0534 {
00033 public:
00035     enum DFR0534CHANNELS
00036     {
00037         CHANNELMP3,
00038         CHANNELAUX,
00039         CHANNELMP3AUX,
00040         CHANNELUNKNOWN
00041     };
00043     enum DFR0534DRIVE
00044     {
00045         DRIVEUSB,
00046         DRIVESD,
00047         DRIVEFLASH,
00048         DRIVEUNKNOWN,
00049         DRIVENO = 0xFF
00050     };
00052     enum DFR0534LOOPMODE
00053     {
00054         LOOPBACKALL,
00055         SINGLEAUDIOLOOP,
00056         SINGLEAUDIOSTOP,
00057         PLAYRANDOM,
00058         DIRECTORYLOOP,
00059         RANDOMINDIRECTORY,
00060         SEQUENTIALINDIRECTORY,
00061         SEQUENTIAL,
```



```

00062     PLAYMODEUNKNOWN
00063 };
00065 enum DFR0534EQ
00066 {
00067     NORMAL,
00068     POP,
00069     ROCK,
00070     JAZZ ,
00071     CLASSIC,
00072     EQUNKNOWN
00073 };
00075 enum DFR0534STATUS
00076 {
00077     STOPPED,
00078     PLAYING,
00079     PAUSED,
00080     STATUSUNKNOWN
00081 };
00087 DFR0534(Stream &stream)
00088 {
00089     m_ptrStream = &stream;
00090 }
00091 void decreaseVolume();
00092 void fastBackwardDuration(word seconds);
00093 void fastForwardDuration(word seconds);
00094 byte getDrive();
00095 byte getDrivesStates();
00096 bool getDuration(byte &hour, byte &minute, byte &second);
00097 bool getFileName(char *name);
00098 word getFileNameNumber();
00099 int getFirstFileNameNumberInCurrentDirectory();
00100 bool getRuntime(byte &hour, byte &minute, byte &second);
00101 byte getStatus();
00102 int getTotalFiles();
00103 int getTotalFilesInCurrentDirectory();
00104 void increaseVolume();
00105 void insertFileByNumber(word track, byte drive=DRIVEFLASH);
00106 void pause();
00107 void play();
00108 void playCombined(char* list);
00109 void playFileByName(char *path, byte drive=DRIVEFLASH);
00110 void playFileByNumber(word track);
00111 void playLastInDirectory();
00112 void playNext();
00113 void playNextDirectory();
00114 void playPrevious();
00115 void prepareFileByNumber(word track);
00116 void repeatPart(byte startMinute, byte startSecond, byte stopMinute, byte stopSecond);
00117 void setChannel(byte channel);
00118 void setDirectory(char *path, byte drive=DRIVEFLASH);
00119 void setDrive(byte drive);
00120 void setEqualizer(byte mode);
00121 void setLoopMode(byte mode);
00122 void setRepeatLoops(word loops);
00123 void setVolume(byte volume);
00124 void stop();
00125 void stopInsertedFile();
00126 void startSendingRuntime();
00127 void stopCombined();
00128 void stopRepeatPart();
00129 void stopSendingRuntime();
00130 private:
00131 void sendStartingCode() {
00132     m_checksum=STARTINGCODE;
00133     m_ptrStream->write((byte) STARTINGCODE);
00134 }
00135 void sendDataByte(byte data) {
00136     m_checksum +=data;
00137     m_ptrStream->write((byte) data);
00138 }
00139 void sendCheckSum() {
00140     m_ptrStream->write((byte)m_checksum);
00141 }
00142 byte m_checksum;
00143 Stream *m_ptrStream = NULL;
00144 };

```



# Index

CHANNELAUX  
    DFR0534, [11](#)  
CHANNELMP3  
    DFR0534, [11](#)  
CHANNELMP3AUX  
    DFR0534, [11](#)  
CHANNELUNKNOWN  
    DFR0534, [11](#)  
  
decreaseVolume  
    DFR0534, [14](#)  
DFR0534, [1](#), [9](#)  
    CHANNELAUX, [11](#)  
    CHANNELMP3, [11](#)  
    CHANNELMP3AUX, [11](#)  
    CHANNELUNKNOWN, [11](#)  
    decreaseVolume, [14](#)  
    DFR0534, [13](#)  
    DFR0534CHANNELS, [11](#)  
    DFR0534DRIVE, [11](#)  
    DFR0534EQ, [12](#)  
    DFR0534LOOPMODE, [12](#)  
    DFR0534STATUS, [13](#)  
    DIRECTORYLOOP, [12](#)  
    DRIVEFLASH, [12](#)  
    DRIVENO, [12](#)  
    DRIVESD, [12](#)  
    DRIVEUNKNOWN, [12](#)  
    DRIVEUSB, [12](#)  
    fastBackwardDuration, [14](#)  
    fastForwardDuration, [14](#)  
    getDrive, [14](#)  
    getDrivesStates, [15](#)  
    getDuration, [16](#)  
    getFileName, [18](#)  
    getFileName, [19](#)  
    getFirstFileNameInCurrentDirectory, [20](#)  
    getRuntime, [21](#)  
    getStatus, [22](#)  
    getTotalFiles, [23](#)  
    getTotalFilesInCurrentDirectory, [24](#)  
    increaseVolume, [25](#)  
    insertFileByNumber, [25](#)  
    LOOPBACKALL, [12](#)  
    NORMAL, [12](#)  
    pause, [26](#)  
    PAUSED, [13](#)  
    play, [26](#)  
    playCombined, [26](#)  
    playFileByName, [27](#)  
    playFileByNumber, [28](#)  
    PLAYING, [13](#)  
    playLastInDirectory, [28](#)  
    PLAYMODEUNKNOWN, [12](#)  
    playNext, [28](#)  
    playNextDirectory, [29](#)  
    playPrevious, [29](#)  
    PLAYRANDOM, [12](#)  
    prepareFileByNumber, [29](#)  
    RANDOMINDIRECTORY, [12](#)  
    repeatPart, [29](#)  
    SEQUENTIAL, [12](#)  
    SEQUENTIALINDIRECTORY, [12](#)  
    setChannel, [30](#)  
    setDirectory, [30](#)  
    setDrive, [31](#)  
    setEqualizer, [31](#)  
    setLoopMode, [31](#)  
    setRepeatLoops, [32](#)  
    setVolume, [32](#)  
    SINGLEAUDIOLOOP, [12](#)  
    SINGLEAUDIOSTOP, [12](#)  
    startSendingRuntime, [33](#)  
    STATUSUNKNOWN, [13](#)  
    stop, [33](#)  
    stopCombined, [33](#)  
    stopInsertedFile, [33](#)  
    STOPPED, [13](#)  
    stopRepeatPart, [34](#)  
    stopSendingRuntime, [34](#)  
DFR0534.cpp, [38](#), [40](#)  
DFR0534.h, [51](#), [52](#)  
    DFR0534\_VERSION, [52](#)  
DFR0534\_VERSION  
    DFR0534.h, [52](#)  
DFR0534CHANNELS  
    DFR0534, [11](#)  
DFR0534DRIVE  
    DFR0534, [11](#)  
DFR0534EQ  
    DFR0534, [12](#)  
DFR0534LOOPMODE  
    DFR0534, [12](#)  
DFR0534STATUS  
    DFR0534, [13](#)  
DIRECTORYLOOP  
    DFR0534, [12](#)  
DRIVEFLASH  
    DFR0534, [12](#)

DRIVENO  
     DFR0534, [12](#)  
 DRIVESD  
     DFR0534, [12](#)  
 DRIVEUNKNOWN  
     DFR0534, [12](#)  
 DRIVEUSB  
     DFR0534, [12](#)  
  
 fastBackwardDuration  
     DFR0534, [14](#)  
 fastForwardDuration  
     DFR0534, [14](#)  
  
 getDrive  
     DFR0534, [14](#)  
 getDrivesStates  
     DFR0534, [15](#)  
 getDuration  
     DFR0534, [16](#)  
 getFileName  
     DFR0534, [18](#)  
 getFileNumber  
     DFR0534, [19](#)  
 getFirstFileNumberInCurrentDirectory  
     DFR0534, [20](#)  
 getRuntime  
     DFR0534, [21](#)  
 getStatus  
     DFR0534, [22](#)  
 getTotalFiles  
     DFR0534, [23](#)  
 getTotalFilesInCurrentDirectory  
     DFR0534, [24](#)  
  
 increaseVolume  
     DFR0534, [25](#)  
 insertFileByNumber  
     DFR0534, [25](#)  
  
 LOOPBACKALL  
     DFR0534, [12](#)  
  
 NORMAL  
     DFR0534, [12](#)  
  
 pause  
     DFR0534, [26](#)  
 PAUSED  
     DFR0534, [13](#)  
 play  
     DFR0534, [26](#)  
 playCombined  
     DFR0534, [26](#)  
 playCombined.ino, [35](#)  
 playFileByName  
     DFR0534, [27](#)  
 playFileByName.ino, [36](#)  
 playFileByNumber  
     DFR0534, [28](#)  
  
 playFileByNumber.ino, [37](#)  
 PLAYING  
     DFR0534, [13](#)  
 playLastInDirectory  
     DFR0534, [28](#)  
 PLAYMODEUNKNOWN  
     DFR0534, [12](#)  
 playNext  
     DFR0534, [28](#)  
 playNextDirectory  
     DFR0534, [29](#)  
 playPrevious  
     DFR0534, [29](#)  
 PLAYRANDOM  
     DFR0534, [12](#)  
 prepareFileByNumber  
     DFR0534, [29](#)  
  
 RANDOMINDIRECTORY  
     DFR0534, [12](#)  
 repeatPart  
     DFR0534, [29](#)  
  
 SEQUENTIAL  
     DFR0534, [12](#)  
 SEQUENTIALINDIRECTORY  
     DFR0534, [12](#)  
 setChannel  
     DFR0534, [30](#)  
 setDirectory  
     DFR0534, [30](#)  
 setDrive  
     DFR0534, [31](#)  
 setEqualizer  
     DFR0534, [31](#)  
 setLoopMode  
     DFR0534, [31](#)  
 setRepeatLoops  
     DFR0534, [32](#)  
 setVolume  
     DFR0534, [32](#)  
 SINGLEAUDIOLOOP  
     DFR0534, [12](#)  
 SINGLEAUDIOSTOP  
     DFR0534, [12](#)  
 startSendingRuntime  
     DFR0534, [33](#)  
 STATUSUNKNOWN  
     DFR0534, [13](#)  
 stop  
     DFR0534, [33](#)  
 stopCombined  
     DFR0534, [33](#)  
 stopInsertedFile  
     DFR0534, [33](#)  
 STOPPED  
     DFR0534, [13](#)  
 stopRepeatPart  
     DFR0534, [34](#)

stopSendingRuntime  
DFR0534, [34](#)