

DFR0534

1.0.1

Generated by Doxygen 1.10.0



<b>1 DFR0534</b>	<b>1</b>
1.1 License and copyright	1
1.2 Appendix	2
1.2.1 DFR0534 pinout	2
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 DFR0534 Class Reference	7
4.1.1 Detailed Description	9
4.1.2 Member Enumeration Documentation	9
4.1.2.1 DFR0534CHANNELS	9
4.1.2.2 DFR0534DRIVE	9
4.1.2.3 DFR0534EQ	10
4.1.2.4 DFR0534LOOPMODE	10
4.1.2.5 DFR0534STATUS	11
4.1.3 Constructor & Destructor Documentation	11
4.1.3.1 DFR0534()	11
4.1.4 Member Function Documentation	11
4.1.4.1 decreaseVolume()	11
4.1.4.2 fastBackwardDuration()	12
4.1.4.3 fastForwardDuration()	12
4.1.4.4 getDrive()	12
4.1.4.5 getDrivesStates()	13
4.1.4.6 getDuration()	15
4.1.4.7 getFileName()	16
4.1.4.8 getFileNameNumber()	17
4.1.4.9 getFirstFileNameNumberInCurrentDirectory()	18
4.1.4.10 getRuntime()	19
4.1.4.11 getStatus()	20
4.1.4.12 getTotalFiles()	21
4.1.4.13 getTotalFilesInCurrentDirectory()	22
4.1.4.14 increaseVolume()	23
4.1.4.15 insertFileByNumber()	24
4.1.4.16 pause()	24
4.1.4.17 play()	24
4.1.4.18 playCombined()	25
4.1.4.19 playFileByName()	25
4.1.4.20 playFileByNumber()	26

4.1.4.21 playLastInDirectory()	26
4.1.4.22 playNext()	27
4.1.4.23 playNextDirectory()	27
4.1.4.24 playPrevious()	27
4.1.4.25 prepareFileByNumber()	27
4.1.4.26 repeatPart()	28
4.1.4.27 setChannel()	28
4.1.4.28 setDirectory()	29
4.1.4.29 setDrive()	29
4.1.4.30 setEqualizer()	30
4.1.4.31 setLoopMode()	30
4.1.4.32 setRepeatLoops()	30
4.1.4.33 setVolume()	31
4.1.4.34 startSendingRuntime()	31
4.1.4.35 stop()	31
4.1.4.36 stopCombined()	32
4.1.4.37 stopInsertedFile()	32
4.1.4.38 stopRepeatPart()	32
4.1.4.39 stopSendingRuntime()	33
<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	39
5.6 DFR0534.h File Reference	50
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 library for a [DFR0534](#) audio module. The library works with SoftwareSerial and is very similar to [https://github.com/sleemanj/JQ8400\\_Serial](https://github.com/sleemanj/JQ8400_Serial), but is no fork.

To create a [DFR0534](#) object pass the existing SoftwareSerial object as 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);
...
```

Examples how to use the library

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

### 1.1 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.

## 1.2 Appendix

### 1.2.1 DF0534 pinout

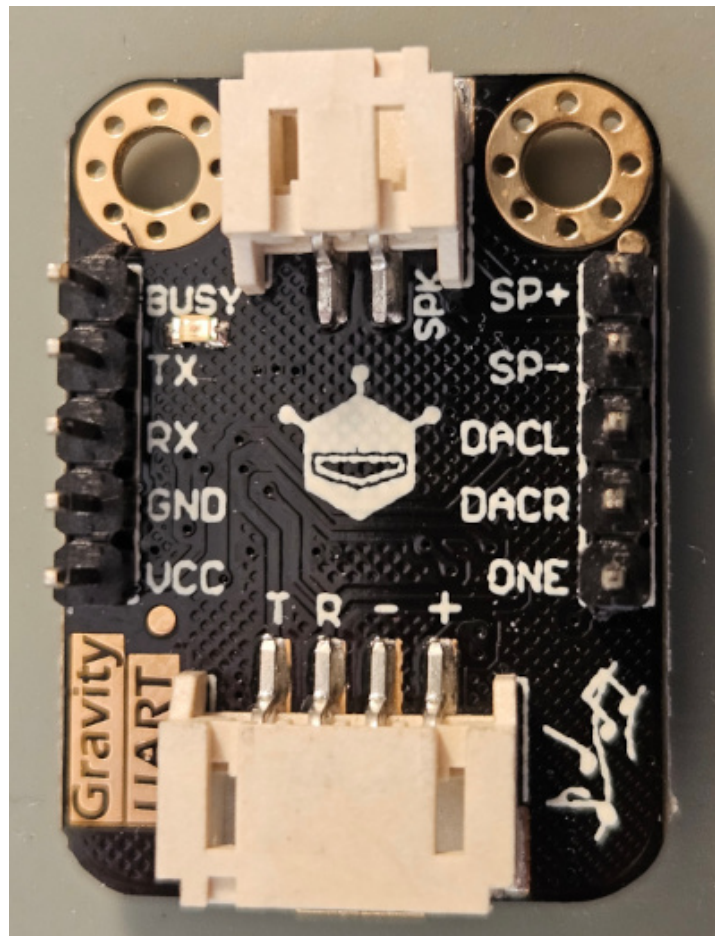


Figure 1.1 DFR0534

Minimal schematic to use this library

Pin	Connected to
TX	Used SoftwareSerial RX
RX	Used SoftwareSerial 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 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	7





## 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>	50



# 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

CHANNELMP3	Use MP3 input channel for DAC output (=default after device startup)
CHANNELAUX	Use AUX input (P26 and P27) for DAC output
CHANNELMP3AUX	Combines MP3 and AUX audio from P26 and P27 for DAC output
CHANNELUNKNOWN	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
----	---------------	---

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 742 of file [DFR0534.cpp](#).

```
00743 {
00744     if (m_ptrStream == NULL) return; // Should not happen
00745     sendStartingCode();
00746     sendDataByte(0x15);
00747     sendDataByte(0x00);
00748     sendChecksum();
00749 }
```

#### 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 1019 of file [DFR0534.cpp](#).

```
01020 {
01021     if (m_ptrStream == NULL) return; // Should not happen
01022     sendStartingCode();
01023     sendDataByte(0x22);
01024     sendDataByte(0x02);
01025     sendDataByte((seconds >> 8) & 0xff);
01026     sendDataByte(seconds & 0xff);
01027     sendChecksum();
01028 }
```

#### 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 1036 of file [DFR0534.cpp](#).

```
01037 {
01038     if (m_ptrStream == NULL) return; // Should not happen
01039     sendStartingCode();
01040     sendDataByte(0x23);
01041     sendDataByte(0x02);
01042     sendDataByte((seconds >> 8) & 0xff);
01043     sendDataByte(seconds & 0xff);
01044     sendChecksum();
01045 }
```

#### 4.1.4.4 getDrive()

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



Get current drive.

Return values

<code>DFR0534::DRIVEUSB</code>	USB drive
<code>DFR0534::DRIVESD</code>	SD card
<code>DFR0534::DRIVEFLASH</code>	Flash memory chip
<code>DFR0534::DRIVENO</code>	No drive found
<code>DFR0534::DRIVEUNKNOWN</code>	Error (for example request timeout)

Definition at line 339 of file `DFR0534.cpp`.

```

00340 {
00341     #define COMMAND 0x0A
00342     #define RECEIVEBYTETIMEOUTMS 100
00343     #define RECEIVEGLOBALTIMEOUTMS 500
00344     #define RECEIVEFAILED DRIVEUNKNOWN
00345     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00346
00347     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00348     sendStartingCode();
00349     sendDataByte(COMMAND);
00350     sendDataByte(0x00);
00351     sendChecksum();
00352
00353     // Receive
00354     int i=0;
00355     byte data, firstByte = 0, sum, length=0xff, result = 0;
00356     unsigned long receiveStartMS = millis();
00357     do {
00358         byte dataReady = 0;
00359         unsigned long lastMS = millis();
00360         // Wait for response or timeout
00361         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00362
00363         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00364         data = m_ptrStream->read();
00365
00366         if (i==0) { // Begin of transmission
00367             firstByte=data;
00368             sum = 0;
00369         }
00370         if ((i == 1) && (data != COMMAND)) {
00371             // Invalid signal => reset receive
00372             i=0;
00373             firstByte = 0;
00374         }
00375         if (i == RECEIVEHEADERLENGTH) {
00376             length = data; // Length of receiving data
00377             if (length != 1) {
00378                 // Invalid length => reset receive
00379                 i=0;
00380                 firstByte = 0;
00381             }
00382         }
00383         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00384             result = data;
00385         }
00386         if (firstByte == STARTINGCODE) {
00387             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00388             i++;
00389         }
00390         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00391     } while (i<length+RECEIVEHEADERLENGTH+2);
00392
00393     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00394     return result;
00395 }

```

#### 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 272 of file [DFR0534.cpp](#).

```

00273 {
00274     #define COMMAND 0x09
00275     #define RECEIVEBYTETIMEOUTMS 100
00276     #define RECEIVEGLOBALTIMEOUTMS 500
00277     #define RECEIVEFAILED DRIVEUNKNOWN
00278     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00279
00280     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00281     sendStartingCode();
00282     sendDataByte(COMMAND);
00283     sendDataByte(0x00);
00284     sendChecksum();
00285
00286     // Receive
00287     int i=0;
00288     byte data, firstByte = 0, sum, length=0xff, result = 0;
00289     unsigned long receiveStartMS = millis();
00290     do {
00291         byte dataReady = 0;
00292         unsigned long lastMS = millis();
00293         // Wait for response or timeout
00294         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00295
00296         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00297         data = m_ptrStream->read();
00298
00299         if (i==0) { // Begin of transmission
00300             firstByte=data;
00301             sum = 0;
00302         }
00303         if ((i == 1) && (data != COMMAND)) {
00304             // Invalid signal => reset receive
00305             i=0;
00306             firstByte = 0;
00307         }
00308         if (i == RECEIVEHEADERLENGTH) {
00309             length = data; // Length of receiving data
00310             if (length != 1) {
00311                 // Invalid length => reset receive
00312                 i=0;
00313                 firstByte = 0;
00314             }
00315         }
00316         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00317             result = data;
00318         }
00319         if (firstByte == STARTINGCODE) {
00320             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00321             i++;
00322         }
00323         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00324     } while (i<length+RECEIVEHEADERLENGTH+2);
00325
00326     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00327     return result;
00328 }
```

## 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 1059 of file DFR0534.cpp.

```
01060 {
01061     #define COMMAND 0x24
01062     #define RECEIVEFAILED false
01063     #define RECEIVEBYTETIMEOUTMS 100
01064     #define RECEIVEGLOBALTIMEOUTMS 500
01065     #define RECEIVEHEADERLENGTH 2 // startingcode+command
01066
01067     if (m_ptrStream == NULL) return false; // Should not happen
01068     sendStartingCode();
01069     sendDataByte(COMMAND);
01070     sendDataByte(0x00);
01071     sendChecksum();
01072
01073     // Receive
01074     int i=0;
01075     byte data, firstByte = 0, sum, length=0xff;
01076     word result = 0;
01077     unsigned long receiveStartMS = millis();
01078     do {
01079         byte dataReady = 0;
01080         unsigned long lastMS = millis();
01081         // Wait for response or timeout
01082         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01083
01084         if (dataReady == 0) return RECEIVEFAILED; // Timeout
01085         data = m_ptrStream->read();
01086
01087         if (i==0) { // Begin of transmission
01088             firstByte=data;
01089             sum = 0;
01090         }
01091         if ((i == 1) && (data != COMMAND)) {
01092             // Invalid signal => reset receive
01093             i=0;
01094             firstByte = 0;
01095         }
01096         if (i == RECEIVEHEADERLENGTH) {
01097             length = data; // Length of receiving data
01098             if (length != 3) {
01099                 // Invalid length => reset receive
01100                 i=0;
01101                 firstByte = 0;
01102             }
01103         }
01104         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01105             switch (i-RECEIVEHEADERLENGTH-1) {
01106                 case 0:
```

```

01107         hour=data;
01108         break;
01109     case 1:
01110         minute=data;
01111         break;
01112     case 2:
01113         second=data;
01114         break;
01115     }
01116 }
01117 if (firstByte == STARTINGCODE) {
01118     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01119     i++;
01120 }
01121 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01122 } while (i<length+RECEIVEHEADERLENGTH+2);
01123
01124 return (data == sum); // Does checksum matches?
01125 }

```

#### 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 907 of file [DFR0534.cpp](#).

```

00908 {
00909     #define COMMAND 0x1E
00910     #define RECEIVEBYTETIMEOUTMS 100
00911     #define RECEIVEGLOBALTIMEOUTMS 500
00912     #define RECEIVEFAILED false
00913     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00914
00915     if (m_ptrStream == NULL) return false; // Should not happen
00916     if (name == NULL) return false;
00917     name[0] = '\0';
00918
00919     sendStartingCode();
00920     sendDataByte(COMMAND);
00921     sendDataByte(0x00);
00922     sendCheckSum();
00923
00924     // Receive
00925     int i=0;
00926     byte data, firstByte = 0, sum, length=0xff;
00927     unsigned long receiveStartMS = millis();
00928     do {
00929         byte dataReady = 0;
00930         unsigned long lastMS = millis();
00931         // Wait for response or timeout
00932         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00933
00934         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00935         data = m_ptrStream->read();
00936         if (i==0) { // Begin of transmission
00937             firstByte=data;
00938             sum = 0;
00939         }
00940         if ((i == 1) && (data != COMMAND)) {
00941             // Invalid signal => reset receive
00942             i=0;
00943             firstByte = 0;
00944         }
00945         if (i == RECEIVEHEADERLENGTH) length = data; // Length of receiving string

```

```

00946     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00947         if ((i-RECEIVEHEADERLENGTH) < 12) { // I expect no longer file names than 8+3 chars plus '\0'
00948             name[i-RECEIVEHEADERLENGTH-1] = data;
00949             name[i-RECEIVEHEADERLENGTH] = '\0';
00950         }
00951     }
00952     if (firstByte == STARTINGCODE) {
00953         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00954         i++;
00955     }
00956     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00957 } while (i<length+RECEIVEHEADERLENGTH+2);
00958 return (data == sum); // Does checksum matches?
00959 }

```

#### 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 421 of file [DFR0534.cpp](#).

```

00422 {
00423     #define COMMAND 0x0D
00424     #define RECEIVEFAILED 0
00425     #define RECEIVEBYTETIMEOUTMS 100
00426     #define RECEIVEGLOBALTIMEOUTMS 500
00427     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00428
00429     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00430     sendStartingCode();
00431     sendDataByte(COMMAND);
00432     sendDataByte(0x00);
00433     sendChecksum();
00434
00435     // Receive
00436     int i=0;
00437     byte data, firstByte = 0, sum, length=0xff;
00438     word result = 0;
00439     unsigned long receiveStartMS = millis();
00440     do {
00441         byte dataReady = 0;
00442         unsigned long lastMS = millis();
00443         // Wait for response or timeout
00444         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00445
00446         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00447         data = m_ptrStream->read();
00448
00449         if (i==0) { // Begin of transmission
00450             firstByte=data;
00451             sum = 0;
00452         }
00453         if ((i == 1) && (data != COMMAND)) {
00454             // Invalid signal => reset receive
00455             i=0;
00456             firstByte = 0;
00457         }
00458         if (i == RECEIVEHEADERLENGTH) {
00459             length = data; // Length of receiving data
00460             if (length != 2) {

```

```

00461         // Invalid length => reset receive
00462         i=0;
00463         firstByte = 0;
00464     }
00465 }
00466 if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00467     switch (i-RECEIVEHEADERLENGTH-1) {
00468         case 0:
00469             result=data<<8;
00470             break;
00471         case 1:
00472             result+=data;
00473             break;
00474     }
00475 }
00476 if (firstByte == STARTINGCODE) {
00477     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00478     i++;
00479 }
00480 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00481 } while (i<length+RECEIVEHEADERLENGTH+2);
00482
00483 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00484 return result;
00485 }

```

#### 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 589 of file DFR0534.cpp.

```

00590 {
00591     #define COMMAND 0x11
00592     #define RECEIVEFAILED -1
00593     #define RECEIVEBYTETIMEOUTMS 100
00594     #define RECEIVEGLOBALTIMEOUTMS 500
00595     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00596
00597     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00598     sendStartingCode();
00599     sendDataByte(COMMAND);
00600     sendDataByte(0x00);
00601     sendChecksum();
00602
00603     // Receive
00604     int i=0;
00605     byte data, firstByte = 0, sum, length=0xff;
00606     word result = 0;
00607     unsigned long receiveStartMS = millis();
00608     do {
00609         byte dataReady = 0;
00610         unsigned long lastMS = millis();
00611         // Wait for response or timeout
00612         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00613
00614         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00615         data = m_ptrStream->read();
00616
00617         if (i==0) { // Begin of transmission
00618             firstByte=data;
00619             sum = 0;
00620         }

```

```

00621     if ((i == 1) && (data != COMMAND)) {
00622         // Invalid signal => reset receive
00623         i=0;
00624         firstByte = 0;
00625     }
00626     if (i == RECEIVEHEADERLENGTH) {
00627         length = data; // Length of receiving data
00628         if (length != 2) {
00629             // Invalid length => reset receive
00630             i=0;
00631             firstByte = 0;
00632         }
00633     }
00634     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00635         switch (i-RECEIVEHEADERLENGTH-1) {
00636             case 0:
00637                 result=data<<8;
00638                 break;
00639             case 1:
00640                 result+=data;
00641                 break;
00642         }
00643     }
00644     if (firstByte == STARTINGCODE) {
00645         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00646         i++;
00647     }
00648     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00649 } while (i<length+RECEIVEHEADERLENGTH+2);
00650
00651 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00652 return result;
00653 }

```

#### 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 1152 of file [DFR0534.cpp](#).

```

1153 {
1154     #define COMMAND 0x25
1155     #define RECEIVEFAILED false
1156     #define RECEIVEBYTETIMEOUTMS 100
1157     #define RECEIVEGLOBALTIMEOUTMS 500
1158     #define RECEIVEHEADERLENGTH 2 // startingcode+command
1159
1160     if (m_ptrStream == NULL) return false; // Should not happen
1161
1162     // Receive
1163     int i=0;

```

```

01164     byte data, firstByte = 0, sum, length=0xff;
01165     word result = 0;
01166     unsigned long receiveStartMS = millis();
01167     do {
01168         byte dataReady = 0;
01169         unsigned long lastMS = millis();
01170         // Wait for response or timeout
01171         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01172
01173         if (dataReady == 0) return RECEIVEFAILED; // Timeout
01174         data = m_ptrStream->read();
01175
01176         if (i==0) { // Begin of transmission
01177             firstByte=data;
01178             sum = 0;
01179         }
01180         if ((i == 1) && (data != COMMAND)) {
01181             // Invalid signal => reset receive
01182             i=0;
01183             firstByte = 0;
01184         }
01185         if (i == RECEIVEHEADERLENGTH) {
01186             length = data; // Length of receiving data
01187             if (length != 3) {
01188                 // Invalid length => reset receive
01189                 i=0;
01190                 firstByte = 0;
01191             }
01192         }
01193         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01194             switch (i-RECEIVEHEADERLENGTH-1) {
01195                 case 0:
01196                     hour=data;
01197                     break;
01198                 case 1:
01199                     minute=data;
01200                     break;
01201                 case 2:
01202                     second=data;
01203                     break;
01204             }
01205         }
01206         if (firstByte == STARTINGCODE) {
01207             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01208             i++;
01209         }
01210         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01211     } while (i<length+RECEIVEHEADERLENGTH+2);
01212
01213     return (data == sum); // Does checksum matches?
01214 }

```

#### 4.1.4.11 getStatus()

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

Get module status.

##### Return values

<i>DFR0534::STOPPED</i>	Audio module is idle
<i>DFR0534::PLAYING</i>	Audio module is playing a file
<i>DFR0534::PAUSED</i>	Audio module is paused
<i>DFR0534::STATUSUNKNOWN</i>	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 493 of file [DFR0534.cpp](#).

```

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 =
m_ptrStream->available();
00517
00518      if (dataReady == 0) return RECEIVEFAILED; // Timeout
00519      data = m_ptrStream->read();
00520
00521      if (i==0) { // Begin of transmission
00522          firstByte=data;
00523          sum = 0;
00524      }
00525      if ((i == 1) && (data != COMMAND)) {
00526          // Invalid signal => reset receive
00527          i=0;
00528          firstByte = 0;
00529      }
00530      if (i == RECEIVEHEADERLENGTH) {
00531          length = data; // Length of receiving data
00532          if (length != 2) {
00533              // Invalid length => reset receive
00534              i=0;
00535              firstByte = 0;
00536          }
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 }

```

#### 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 661 of file [DFR0534.cpp](#).

```

00662 {
00663     #define COMMAND 0x12
00664     #define RECEIVEFAILED -1
00665     #define RECEIVEBYTETIMEOUTMS 100
00666     #define RECEIVEGLOBALTIMEOUTMS 500
00667     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00668
00669     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00670     sendStartingCode();
00671     sendDataByte(COMMAND);
00672     sendDataByte(0x00);
00673     sendChecksum();
00674
00675     // Receive
00676     int i=0;
00677     byte data, firstByte = 0, sum, length=0xff;
00678     word result = 0;
00679     unsigned long receiveStartMS = millis();
00680     do {
00681         byte dataReady = 0;
00682         unsigned long lastMS = millis();
00683         // Wait for response or timeout
00684         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00685
00686         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00687         data = m_ptrStream->read();
00688
00689         if (i==0) { // Begin of transmission
00690             firstByte=data;
00691             sum = 0;
00692         }
00693         if ((i == 1) && (data != COMMAND)) {
00694             // Invalid signal => reset receive
00695             i=0;
00696             firstByte = 0;
00697         }
00698         if (i == RECEIVEHEADERLENGTH) {
00699             length = data; // Length of receiving data
00700             if (length != 2) {
00701                 // Invalid length => reset receive
00702                 i=0;
00703                 firstByte = 0;
00704             }
00705         }
00706         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00707             switch (i-RECEIVEHEADERLENGTH-1) {
00708                 case 0:
00709                     result=data<<8;
00710                     break;
00711                 case 1:
00712                     result+=data;
00713                     break;
00714             }
00715         }
00716         if (firstByte == STARTINGCODE) {
00717             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00718             i++;
00719         }
00720         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00721     } while (i<length+RECEIVEHEADERLENGTH+2);
00722
00723     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00724     return result;
00725 }

```

#### 4.1.4.14 increaseVolume()

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

Increase volume by one step.

Definition at line 730 of file [DFR0534.cpp](#).

```

00731 {
00732     if (m_ptrStream == NULL) return; // Should not happen
00733     sendStartingCode();
00734     sendDataByte(0x14);
00735     sendDataByte(0x00);
00736     sendChecksum();
00737 }

```

#### 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 [759](#) of file [DFR0534.cpp](#).

```
00760 {
00761     if (m_ptrStream == NULL) return; // Should not happen
00762     if (drive >= DRIVEUNKNOWN) return;
00763     sendStartingCode();
00764     sendDataByte(0x16);
00765     sendDataByte(0x03);
00766     sendDataByte(drive);
00767     sendDataByte((track >> 8) & 0xff);
00768     sendDataByte(track & 0xff);
00769     sendChecksum();
00770 }
```

#### 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	list	Concatenated list of all files to play
----	------	--

Definition at line 852 of file [DFR0534.cpp](#).

```
00853 {
00854     if (m_ptrStream == NULL) return; // Should not happen
00855     if (list == NULL) return;
00856     if ((strlen(list) % 2) != 0) return;
00857
00858     sendStartingCode();
00859     sendDataByte(0x1B);
00860     sendDataByte(strlen(list));
00861     for (int i=0; i<strlen(list); i++) {
00862         sendDataByte(list[i]);
00863     }
00864     sendChecksum();
00865 }
```

#### 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 the dot for the file extension
- All characters in upper case
- Every file and folder whose length is shorter then 8 chars must be filled up to the 8 chars length by spaces.
- Only WAV and MP3 files are supported Wildcards \* (=multiple arbitrary characters) and ? (=one single arbitrary character) are allowed and can be used to reduce filling spaces.

Valid examples:

- "/01 WAV" for file 01.wav
- "/99-AFR~1MP3" for a file /99-Africa.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

## 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 246 of file [DFR0534.cpp](#).

```
00247 {
00248     if (m_ptrStream == NULL) return; // Should not happen
00249     if (path == NULL) return;
00250     if (drive >= DRIVEUNKNOWN) return;
00251     sendStartingCode();
00252     sendDataByte(0x08);
00253     sendDataByte(strlen(path)+1);
00254     sendDataByte(drive);
00255     for (int i=0; i<strlen(path); i++) {
00256         sendDataByte(path[i]);
00257     }
00258     sendChecksum();
00259 }
```

#### 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 562 of file [DFR0534.cpp](#).

```
00563 {
00564     if (m_ptrStream == NULL) return; // Should not happen
00565     sendStartingCode();
00566     sendDataByte(0x0E);
00567     sendDataByte(0x00);
00568     sendChecksum();
00569 }
```

#### 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 574 of file [DFR0534.cpp](#).

```
00575 {  
00576     if (m_ptrStream == NULL) return; // Should not happen  
00577     sendStartingCode();  
00578     sendDataByte(0x0F);  
00579     sendDataByte(0x00);  
00580     sendChecksum();  
00581 }
```

#### 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 966 of file [DFR0534.cpp](#).

```

00967 {
00968     if (m_ptrStream == NULL) return; // Should not happen
00969     sendStartingCode();
00970     sendDataByte(0x1F);
00971     sendDataByte(0x02);
00972     sendDataByte((track » 8) & 0xff);
00973     sendDataByte(track & 0xff);
00974     sendChecksum();
00975 }

```

#### 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 987 of file [DFR0534.cpp](#).

```

00988 {
00989     if (m_ptrStream == NULL) return; // Should not happen
00990     sendStartingCode();
00991     sendDataByte(0x20);
00992     sendDataByte(0x04);
00993     sendDataByte(startMinute);
00994     sendDataByte(startSecond);
00995     sendDataByte(stopMinute);
00996     sendDataByte(stopSecond);
00997     sendChecksum();
00998 }

```

#### 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 887 of file [DFR0534.cpp](#).



```

00888 {
00889     if (m_ptrStream == NULL) return; // Should not happen
00890     if (channel >= CHANNELUNKNOWN) return;
00891     sendStartingCode();
00892     sendDataByte(0x1D);
00893     sendDataByte(0x01);
00894     sendDataByte(channel);
00895     sendChecksum();
00896 }

```

#### 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 792 of file [DFR0534.cpp](#).

```

00793 {
00794     if (m_ptrStream == NULL) return; // Should not happen
00795     if (path == NULL) return;
00796     if (drive >= DRIVEUNKNOWN) return;
00797     sendStartingCode();
00798     sendDataByte(0x17);
00799     sendDataByte(strlen(path)+1);
00800     sendDataByte(drive);
00801     for (int i=0;i<strlen(path);i++) {
00802         sendDataByte(path[i]);
00803     }
00804     sendChecksum();
00805 }

```

#### 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 402 of file [DFR0534.cpp](#).

```

00403 {
00404     if (m_ptrStream == NULL) return; // Should not happen
00405     if (drive >= DRIVEUNKNOWN) return;
00406     sendStartingCode();
00407     sendDataByte(0x0B);
00408     sendDataByte(0x01);
00409     sendDataByte(drive);
00410     sendChecksum();
00411 }

```

#### 4.1.4.30 setEqualizer()

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

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

##### Parameters

in	mode	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 812 of file [DFR0534.cpp](#).

```
00813 {
00814     if (m_ptrStream == NULL) return; // Should not happen
00815     if (mode >= PLAYMODEUNKNOWN) return;
00816     sendStartingCode();
00817     sendDataByte(0x18);
00818     sendDataByte(0x01);
00819     sendDataByte(mode);
00820     sendChecksum();
00821 }
```

#### 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	<i>loops</i>	Number of loops
----	--------------	-----------------

Definition at line 830 of file [DFR0534.cpp](#).

```
00831 {
00832     if (m_ptrStream == NULL) return; // Should not happen
00833     sendStartingCode();
00834     sendDataByte(0x19);
00835     sendDataByte(0x02);
00836     sendDataByte((loops >> 8) & 0xff);
00837     sendDataByte(loops & 0xff);
00838     sendChecksum();
00839 }
```

## 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 1130 of file [DFR0534.cpp](#).

```
01131 {
01132     if (m_ptrStream == NULL) return; // Should not happen
01133     sendStartingCode();
01134     sendDataByte(0x25);
01135     sendDataByte(0x00);
01136     sendChecksum();
01137 }
```

## 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 870 of file [DFR0534.cpp](#).

```
00871 {  
00872     if (m_ptrStream == NULL) return; // Should not happen  
00873     sendStartingCode();  
00874     sendDataByte(0x1C);  
00875     sendDataByte(0x00);  
00876     sendChecksum();  
00877 }
```

#### 4.1.4.37 stopInsertedFile()

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

Stop inserted file.

Continue original file

Definition at line 777 of file [DFR0534.cpp](#).

```
00778 {  
00779     if (m_ptrStream == NULL) return; // Should not happen  
00780     sendStartingCode();  
00781     sendDataByte(0x10);  
00782     sendDataByte(0x00);  
00783     sendChecksum();  
00784 }
```

#### 4.1.4.38 stopRepeatPart()

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

Stop repeating part of the current file.

Definition at line 1003 of file [DFR0534.cpp](#).

```
01004 {  
01005     if (m_ptrStream == NULL) return; // Should not happen  
01006     sendStartingCode();  
01007     sendDataByte(0x21);  
01008     sendDataByte(0x00);  
01009     sendChecksum();  
01010 }
```

#### 4.1.4.39 stopSendingRuntime()

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

Stop sending runtime.

Definition at line 1219 of file [DFR0534.cpp](#).

```
01220 {  
01221     if (m_ptrStream == NULL) return; // Should not happen  
01222     sendStartingCode();  
01223     sendDataByte(0x26);  
01224     sendDataByte(0x00);  
01225     sendChecksum();  
01226 }
```

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
00005 #include <SoftwareSerial.h>
00006 #include <DFR0534.h>
00007
00008 #define TX_PIN A0
00009 #define RX_PIN A1
00010 SoftwareSerial g_serial(RX_PIN, TX_PIN);
00011 DFR0534 g_audio(g_serial);
00012
00013 void setup() {
00014     // Serial for console output
00015     Serial.begin(9600);
00016     // Software serial for communication to DFR0534 module
00017     g_serial.begin(9600);
00018
00019     // Set volume
00020     g_audio.setVolume(18);
00021
00022     /* The parameter string for the playCombined function is just
00023      * a concatenation of all files in the desired order without
00024      * path and without extension.
00025      * All files have to be in the folder /ZH and the each
00026      * file has to have a length (without extension) of two chars.
00027      *
00028      * You can get example files from
00029      * https://github.com/codingABI/DFR0534/tree/main/assets/exampleContent
00030      */
00031     /* Plays files the custom order, like a playlist and stops after the last file:
00032      * /ZH/05.wav
00033      * /ZH/04.wav
00034      * /ZH/03.wav
00035      * /ZH/02.wav
00036      * /ZH/01.wav
00037      * /ZH/0A.wav
00038      */
00039     g_audio.playCombined("05040302010A");
00040 }
00041
00042 void loop() {
00043     static unsigned long lastDisplayMS = millis();
00044     char name[12];
00045
00046     // Show information about current track every 500ms
00047     if (millis() - lastDisplayMS > 500) {
00048         Serial.print("number: ");
00049         word fileNumber = g_audio.getFileNumber();
00050         if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("---");
00051
00052         Serial.print(" name: ");
00053         if (g_audio.getFileName(name)) Serial.print(name);
00054
00055         Serial.print(" status: ");
00056         switch (g_audio.getStatus()) {
00057             case DFR0534::STOPPED:
```

```

00058     Serial.println("Stopped");
00059     break;
00060     case DFR0534::PAUSED:
00061         Serial.println("Paused");
00062         break;
00063     case DFR0534::PLAYING:
00064         Serial.println("Playing");
00065         break;
00066     case DFR0534::STATUSUNKNOWN:
00067         Serial.println("Unknown");
00068         break;
00069     }
00070     lastDisplayMS = millis();
00071 }
00072 }

```

## 5.2 playFileByName.ino

```

00001 /*
00002  * Example for using the DFR0534 for playing audio files by file name
00003  */
00004
00005 #include <SoftwareSerial.h>
00006 #include <DFR0534.h>
00007
00008 #define TX_PIN A0
00009 #define RX_PIN A1
00010 SoftwareSerial g_serial(RX_PIN, TX_PIN);
00011 DFR0534 g_audio(g_serial);
00012
00013 void setup() {
00014     // Serial for console output
00015     Serial.begin(9600);
00016     // Software serial for communication to DFR0534 module
00017     g_serial.begin(9600);
00018
00019     // Set volume
00020     g_audio.setVolume(18);
00021
00022     /* The file name/path for the function playFileByName() is the full path of the audio file to be
00023        played
00024        * in format which looks like a special unix 8+3 format:
00025        * - Without the dot for the file extension
00026        * - All characters in upper case
00027        * - Every file and folder whose length is shorter then 8 chars must be filled up to the 8 chars
00028        *   length by spaces.
00029        * - Only WAV and MP3 files are supported
00030        * Wildcards * (=multiple arbitrary characters) and ? (=one single arbitrary character) are allowed
00031        * and can be used to reduce filling spaces.
00032
00033        * Valid examples:
00034        * - "/01      WAV" for file 01.wav
00035        * - "/99-AFR~1MP3" for a file /99-Africa.mp3
00036        * - "/99-AFR*MP3" for first file matching /99-Afr*.mp3
00037        * - "/10*" for first audio file matching /10*.*
00038        * - "/10      /20      WAV" for the file /10/20.wav
00039
00040        * You can get example files from
00041        https://github.com/codingABI/DFR0534/tree/main/assets/exampleContent
00042
00043        * Valid examples:
00044        * - "/01      WAV" for file 01.wav
00045        * - "/99-AFR~1MP3" for a file /99-Africa.mp3
00046        * - "/99-AFR*MP3" for first file matching /99-Afr*.mp3
00047        * - "/10*" for first audio file matching /10*.*
00048        * - "/10      /20      WAV" for the file /10/20.wav
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: ");

```



```

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

```

## 5.3 playFileByNumber.ino

```

00001 /*
00002  * Example for using the DFR0534 for playing audio files by file number
00003  */
00004
00005 #include <SoftwareSerial.h>
00006 #include <DFR0534.h>
00007
00008 #define TX_PIN A0
00009 #define RX_PIN A1
00010 SoftwareSerial g_serial(RX_PIN, TX_PIN);
00011 DFR0534 g_audio(g_serial);
00012
00013 void setup() {
00014     // Serial for console output
00015     Serial.begin(9600);
00016     // Software serial for communication to DFR0534 module
00017     g_serial.begin(9600);
00018
00019     // Set volume
00020     g_audio.setVolume(18);
00021
00022     // Show some device infos
00023     Serial.print("Ready drives: ");
00024     byte drive = g_audio.getDrivesStates();
00025     if ((drive > DFR0534::DRIVEUSB) & 1) == 1) Serial.print("USB ");
00026     if ((drive > DFR0534::DRIVESD) & 1) == 1) Serial.print("SD ");
00027     if ((drive > DFR0534::DRIVEFLASH) & 1) == 1) Serial.print("FLASH ");
00028     Serial.println();
00029
00030     Serial.print("Current playing drive: ");
00031     switch(g_audio.getDrive()) {
00032     case DFR0534::DRIVEUSB:
00033         Serial.println("USB");
00034         break;
00035     case DFR0534::DRIVESD:
00036         Serial.println("SD");
00037         break;
00038     case DFR0534::DRIVEFLASH:
00039         Serial.println("FLASH");
00040         break;
00041     case DFR0534::DRIVENO:
00042         Serial.println("No drive");
00043         break;
00044     default:
00045         Serial.println("Unknown");
00046         break;
00047     }
00048
00049     Serial.print("Total files: ");
00050     Serial.println(g_audio.getTotalFiles());
00051     Serial.print("Total files in directory: ");
00052     Serial.println(g_audio.getTotalFilesInCurrentDirectory());
00053
00054     Serial.print("First file: ");
00055     Serial.println(g_audio.getFirstFileNumberInCurrentDirectory());
00056
00057     // Play the first audio file copied to the DFR0534
00058     // (Second file copied to the DFR0534 would be number 2...)
00059     g_audio.playFileByNumber(1);
00060 }

```

```

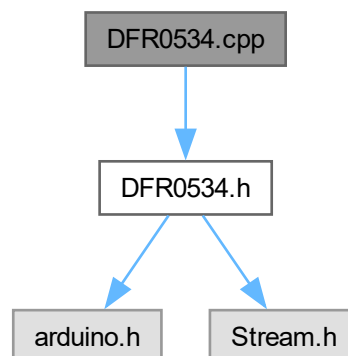
00061
00062 void loop() {
00063     static unsigned long lastDisplayMS = millis()-500;
00064     char name[12];
00065
00066     // Show information about current track once per second
00067     if (millis()-lastDisplayMS > 1000) {
00068         Serial.print("number: ");
00069         word fileNumber = g_audio.getFileNumber();
00070         if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");
00071
00072         Serial.print(" name: ");
00073         if (g_audio.getFileName(name)) Serial.print(name);
00074
00075         Serial.print(" status: ");
00076         switch (g_audio.getStatus()) {
00077             case DFR0534::STOPPED:
00078                 Serial.println("Stopped");
00079                 break;
00080             case DFR0534::PAUSED:
00081                 Serial.println("Paused");
00082                 break;
00083             case DFR0534::PLAYING:
00084                 Serial.println("Playing");
00085                 break;
00086             case DFR0534::STATUSUNKNOWN:
00087                 Serial.println("Unknown");
00088                 break;
00089         }
00090         lastDisplayMS = millis();
00091     }
00092 }

```

## 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

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, W25Q64JVS1Q 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

1.0.1

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
00246 void DFR0534::playFileByName(char *path, byte drive)
00247 {
00248     if (m_ptrStream == NULL) return; // Should not happen
00249     if (path == NULL) return;
00250     if (drive >= DRIVEUNKNOWN) return;
00251     sendStartingCode();
00252     sendDataByte(0x08);
00253     sendDataByte(strlen(path)+1);
00254     sendDataByte(drive);
00255     for (int i=0; i<strlen(path); i++) {
00256         sendDataByte(path[i]);
00257     }
00258     sendChecksum();
00259 }
00260
00272 byte DFR0534::getDrivesStates()
00273 {
00274     #define COMMAND 0x09
00275     #define RECEIVEBYTETIMEOUTMS 100
00276     #define RECEIVEGLOBALTIMEOUTMS 500
00277     #define RECEIVEFAILED DRIVEUNKNOWN
00278     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00279
00280     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00281     sendStartingCode();
00282     sendDataByte(COMMAND);
00283     sendDataByte(0x00);
00284     sendChecksum();
00285
00286     // Receive
00287     int i=0;
00288     byte data, firstByte = 0, sum, length=0xff, result = 0;
00289     unsigned long receiveStartMS = millis();
00290     do {
00291         byte dataReady = 0;
00292         unsigned long lastMS = millis();
00293         // Wait for response or timeout
00294         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00295
00296         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00297         data = m_ptrStream->read();
00298
00299         if (i==0) { // Begin of transmission
00300             firstByte=data;
00301             sum = 0;

```

```

00302     }
00303     if ((i == 1) && (data != COMMAND)) {
00304         // Invalid signal => reset receive
00305         i=0;
00306         firstByte = 0;
00307     }
00308     if (i == RECEIVEHEADERLENGTH) {
00309         length = data; // Length of receiving data
00310         if (length != 1) {
00311             // Invalid length => reset receive
00312             i=0;
00313             firstByte = 0;
00314         }
00315     }
00316     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00317         result = data;
00318     }
00319     if (firstByte == STARTINGCODE) {
00320         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00321         i++;
00322     }
00323     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00324 } while (i<length+RECEIVEHEADERLENGTH+2);
00325
00326 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00327 return result;
00328 }
00329
00330 byte DFR0534::getDrive()
00331 {
00332     #define COMMAND 0x0A
00333     #define RECEIVEBYTETIMEOUTMS 100
00334     #define RECEIVEGLOBALTIMEOUTMS 500
00335     #define RECEIVEFAILED DRIVEUNKNOWN
00336     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00337
00338     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00339     sendStartingCode();
00340     sendDataByte(COMMAND);
00341     sendDataByte(0x00);
00342     sendChecksum();
00343
00344     // Receive
00345     int i=0;
00346     byte data, firstByte = 0, sum, length=0xff, result = 0;
00347     unsigned long receiveStartMS = millis();
00348     do {
00349         byte dataReady = 0;
00350         unsigned long lastMS = millis();
00351         // Wait for response or timeout
00352         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
00353             m_ptrStream->available();
00354
00355         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00356         data = m_ptrStream->read();
00357
00358         if (i==0) { // Begin of transmission
00359             firstByte=data;
00360             sum = 0;
00361         }
00362         if ((i == 1) && (data != COMMAND)) {
00363             // Invalid signal => reset receive
00364             i=0;
00365             firstByte = 0;
00366         }
00367         if (i == RECEIVEHEADERLENGTH) {
00368             length = data; // Length of receiving data
00369             if (length != 1) {
00370                 // Invalid length => reset receive
00371                 i=0;
00372                 firstByte = 0;
00373             }
00374         }
00375         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00376             result = data;
00377         }
00378         if (firstByte == STARTINGCODE) {
00379             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00380             i++;
00381         }
00382         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00383     } while (i<length+RECEIVEHEADERLENGTH+2);
00384
00385     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00386     return result;
00387 }
00388
00389
00390

```

```

00402 void DFR0534::setDrive(byte drive)
00403 {
00404     if (m_ptrStream == NULL) return; // Should not happen
00405     if (drive >= DRIVEUNKNOWN) return;
00406     sendStartingCode();
00407     sendDataByte(0x0B);
00408     sendDataByte(0x01);
00409     sendDataByte(drive);
00410     sendChecksum();
00411 }
00412
00421 word DFR0534::getFileNumber()
00422 {
00423     #define COMMAND 0x0D
00424     #define RECEIVEFAILED 0
00425     #define RECEIVEBYTETIMEOUTMS 100
00426     #define RECEIVEGLOBALTIMEOUTMS 500
00427     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00428
00429     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00430     sendStartingCode();
00431     sendDataByte(COMMAND);
00432     sendDataByte(0x00);
00433     sendChecksum();
00434
00435     // Receive
00436     int i=0;
00437     byte data, firstByte = 0, sum, length=0xff;
00438     word result = 0;
00439     unsigned long receiveStartMS = millis();
00440     do {
00441         byte dataReady = 0;
00442         unsigned long lastMS = millis();
00443         // Wait for response or timeout
00444         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00445
00446         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00447         data = m_ptrStream->read();
00448
00449         if (i==0) { // Begin of transmission
00450             firstByte=data;
00451             sum = 0;
00452         }
00453         if ((i == 1) && (data != COMMAND)) {
00454             // Invalid signal => reset receive
00455             i=0;
00456             firstByte = 0;
00457         }
00458         if (i == RECEIVEHEADERLENGTH) {
00459             length = data; // Length of receiving data
00460             if (length != 2) {
00461                 // Invalid length => reset receive
00462                 i=0;
00463                 firstByte = 0;
00464             }
00465         }
00466         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00467             switch (i-RECEIVEHEADERLENGTH-1) {
00468                 case 0:
00469                     result=data<<8;
00470                     break;
00471                 case 1:
00472                     result+=data;
00473                     break;
00474             }
00475         }
00476         if (firstByte == STARTINGCODE) {
00477             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00478             i++;
00479         }
00480         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00481     } while (i<length+RECEIVEHEADERLENGTH+2);
00482
00483     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00484     return result;
00485 }
00486
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 =
m_ptrStream->available();
00517
00518         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00519         data = m_ptrStream->read();
00520
00521         if (i==0) { // Begin of transmission
00522             firstByte=data;
00523             sum = 0;
00524         }
00525         if ((i == 1) && (data != COMMAND)) {
00526             // Invalid signal => reset receive
00527             i=0;
00528             firstByte = 0;
00529         }
00530         if (i == RECEIVEHEADERLENGTH) {
00531             length = data; // Length of receiving data
00532             if (length != 2) {
00533                 // Invalid length => reset receive
00534                 i=0;
00535                 firstByte = 0;
00536             }
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
00562 void DFR0534::playLastInDirectory()
00563 {
00564     if (m_ptrStream == NULL) return; // Should not happen
00565     sendStartingCode();
00566     sendDataByte(0x0E);
00567     sendDataByte(0x00);
00568     sendChecksum();
00569 }
00570
00574 void DFR0534::playNextDirectory()
00575 {
00576     if (m_ptrStream == NULL) return; // Should not happen
00577     sendStartingCode();
00578     sendDataByte(0x0F);
00579     sendDataByte(0x00);
00580     sendChecksum();
00581 }
00582
00589 int DFR0534::getFirstFileNumberInCurrentDirectory()
00590 {
00591     #define COMMAND 0x11
00592     #define RECEIVEFAILED -1
00593     #define RECEIVEBYTETIMEOUTMS 100
00594     #define RECEIVEGLOBALTIMEOUTMS 500
00595     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00596
00597     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00598     sendStartingCode();
00599     sendDataByte(COMMAND);

```



```

00600     sendDataByte(0x00);
00601     sendChecksum();
00602
00603     // Receive
00604     int i=0;
00605     byte data, firstByte = 0, sum, length=0xff;
00606     word result = 0;
00607     unsigned long receiveStartMS = millis();
00608     do {
00609         byte dataReady = 0;
00610         unsigned long lastMS = millis();
00611         // Wait for response or timeout
00612         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00613
00614         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00615         data = m_ptrStream->read();
00616
00617         if (i==0) { // Begin of transmission
00618             firstByte=data;
00619             sum = 0;
00620         }
00621         if ((i == 1) && (data != COMMAND)) {
00622             // Invalid signal => reset receive
00623             i=0;
00624             firstByte = 0;
00625         }
00626         if (i == RECEIVEHEADERLENGTH) {
00627             length = data; // Length of receiving data
00628             if (length != 2) {
00629                 // Invalid length => reset receive
00630                 i=0;
00631                 firstByte = 0;
00632             }
00633         }
00634         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00635             switch (i-RECEIVEHEADERLENGTH-1) {
00636                 case 0:
00637                     result=data<<8;
00638                     break;
00639                 case 1:
00640                     result+=data;
00641                     break;
00642             }
00643         }
00644         if (firstByte == STARTINGCODE) {
00645             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00646             i++;
00647         }
00648         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00649     } while (i<length+RECEIVEHEADERLENGTH+2);
00650
00651     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00652     return result;
00653 }
00654
00661 int DFR0534::getTotalFilesInCurrentDirectory()
00662 {
00663     #define COMMAND 0x12
00664     #define RECEIVEFAILED -1
00665     #define RECEIVEBYTETIMEOUTMS 100
00666     #define RECEIVEGLOBALTIMEOUTMS 500
00667     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00668
00669     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00670     sendStartingCode();
00671     sendDataByte(COMMAND);
00672     sendDataByte(0x00);
00673     sendChecksum();
00674
00675     // Receive
00676     int i=0;
00677     byte data, firstByte = 0, sum, length=0xff;
00678     word result = 0;
00679     unsigned long receiveStartMS = millis();
00680     do {
00681         byte dataReady = 0;
00682         unsigned long lastMS = millis();
00683         // Wait for response or timeout
00684         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00685
00686         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00687         data = m_ptrStream->read();
00688
00689         if (i==0) { // Begin of transmission
00690             firstByte=data;

```

```

00691     sum = 0;
00692 }
00693 if ((i == 1) && (data != COMMAND)) {
00694     // Invalid signal => reset receive
00695     i=0;
00696     firstByte = 0;
00697 }
00698 if (i == RECEIVEHEADERLENGTH) {
00699     length = data; // Length of receiving data
00700     if (length != 2) {
00701         // Invalid length => reset receive
00702         i=0;
00703         firstByte = 0;
00704     }
00705 }
00706 if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00707     switch (i-RECEIVEHEADERLENGTH-1) {
00708         case 0:
00709             result=data<<8;
00710             break;
00711         case 1:
00712             result+=data;
00713             break;
00714     }
00715 }
00716 if (firstByte == STARTINGCODE) {
00717     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00718     i++;
00719 }
00720 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00721 } while (i<length+RECEIVEHEADERLENGTH+2);
00722
00723 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00724 return result;
00725 }
00726
00730 void DFR0534::increaseVolume()
00731 {
00732     if (m_ptrStream == NULL) return; // Should not happen
00733     sendStartingCode();
00734     sendDataByte(0x14);
00735     sendDataByte(0x00);
00736     sendChecksum();
00737 }
00738
00742 void DFR0534::decreaseVolume()
00743 {
00744     if (m_ptrStream == NULL) return; // Should not happen
00745     sendStartingCode();
00746     sendDataByte(0x15);
00747     sendDataByte(0x00);
00748     sendChecksum();
00749 }
00750
00759 void DFR0534::insertFileByNumber(word track, byte drive)
00760 {
00761     if (m_ptrStream == NULL) return; // Should not happen
00762     if (drive >= DRIVEUNKNOWN) return;
00763     sendStartingCode();
00764     sendDataByte(0x16);
00765     sendDataByte(0x03);
00766     sendDataByte(drive);
00767     sendDataByte((track >> 8) & 0xff);
00768     sendDataByte(track & 0xff);
00769     sendChecksum();
00770 }
00771
00777 void DFR0534::stopInsertedFile()
00778 {
00779     if (m_ptrStream == NULL) return; // Should not happen
00780     sendStartingCode();
00781     sendDataByte(0x10);
00782     sendDataByte(0x00);
00783     sendChecksum();
00784 }
00785
00792 void DFR0534::setDirectory(char *path, byte drive)
00793 {
00794     if (m_ptrStream == NULL) return; // Should not happen
00795     if (path == NULL) return;
00796     if (drive >= DRIVEUNKNOWN) return;
00797     sendStartingCode();
00798     sendDataByte(0x17);
00799     sendDataByte(strlen(path)+1);
00800     sendDataByte(drive);
00801     for (int i=0;i<strlen(path);i++) {
00802         sendDataByte(path[i]);

```

```

00803     }
00804     sendChecksum();
00805 }
00806
00812 void DFR0534::setLoopMode(byte mode)
00813 {
00814     if (m_ptrStream == NULL) return; // Should not happen
00815     if (mode >= PLAYMODEUNKNOWN) return;
00816     sendStartingCode();
00817     sendDataByte(0x18);
00818     sendDataByte(0x01);
00819     sendDataByte(mode);
00820     sendChecksum();
00821 }
00822
00830 void DFR0534::setRepeatLoops(word loops)
00831 {
00832     if (m_ptrStream == NULL) return; // Should not happen
00833     sendStartingCode();
00834     sendDataByte(0x19);
00835     sendDataByte(0x02);
00836     sendDataByte((loops >> 8) & 0xff);
00837     sendDataByte(loops & 0xff);
00838     sendChecksum();
00839 }
00840
00852 void DFR0534::playCombined(char* list)
00853 {
00854     if (m_ptrStream == NULL) return; // Should not happen
00855     if (list == NULL) return;
00856     if ((strlen(list) % 2) != 0) return;
00857     sendStartingCode();
00858     sendDataByte(0x1B);
00859     sendDataByte(strlen(list));
00860     for (int i=0; i<strlen(list); i++) {
00861         sendDataByte(list[i]);
00862     }
00863     sendChecksum();
00864 }
00865
00866
00870 void DFR0534::stopCombined()
00871 {
00872     if (m_ptrStream == NULL) return; // Should not happen
00873     sendStartingCode();
00874     sendDataByte(0x1C);
00875     sendDataByte(0x00);
00876     sendChecksum();
00877 }
00878
00887 void DFR0534::setChannel(byte channel)
00888 {
00889     if (m_ptrStream == NULL) return; // Should not happen
00890     if (channel >= CHANNELUNKNOWN) return;
00891     sendStartingCode();
00892     sendDataByte(0x1D);
00893     sendDataByte(0x01);
00894     sendDataByte(channel);
00895     sendChecksum();
00896 }
00897
00907 bool DFR0534::getFileName(char *name)
00908 {
00909     #define COMMAND 0x1E
00910     #define RECEIVEBYTETIMEOUTMS 100
00911     #define RECEIVEGLOBALTIMEOUTMS 500
00912     #define RECEIVEFAILED false
00913     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00914
00915     if (m_ptrStream == NULL) return false; // Should not happen
00916     if (name == NULL) return false;
00917     name[0] = '\0';
00918
00919     sendStartingCode();
00920     sendDataByte(COMMAND);
00921     sendDataByte(0x00);
00922     sendChecksum();
00923
00924     // Receive
00925     int i=0;
00926     byte data, firstByte = 0, sum, length=0xff;
00927     unsigned long receiveStartMS = millis();
00928     do {
00929         byte dataReady = 0;
00930         unsigned long lastMS = millis();
00931         // Wait for response or timeout
00932         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =

```

```

    m_ptrStream->available();
00933
00934     if (dataReady == 0) return RECEIVEFAILED; // Timeout
00935     data = m_ptrStream->read();
00936     if (i==0) { // Begin of transmission
00937         firstByte=data;
00938         sum = 0;
00939     }
00940     if ((i == 1) && (data != COMMAND)) {
00941         // Invalid signal => reset receive
00942         i=0;
00943         firstByte = 0;
00944     }
00945     if (i == RECEIVEHEADERLENGTH) length = data; // Length of receiving string
00946     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00947         if ((i-RECEIVEHEADERLENGTH) < 12) { // I expect no longer file names than 8+3 chars plus '\0'
00948             name[i-RECEIVEHEADERLENGTH-1] = data;
00949             name[i-RECEIVEHEADERLENGTH] = '\0';
00950         }
00951     }
00952     if (firstByte == STARTINGCODE) {
00953         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00954         i++;
00955     }
00956     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00957 } while (i<length+RECEIVEHEADERLENGTH+2);
00958 return (data == sum); // Does checksum matches?
00959 }
00960
00961 void DFR0534::prepareFileByNumber(word track)
00962 {
00963     if (m_ptrStream == NULL) return; // Should not happen
00964     sendStartingCode();
00965     sendDataByte(0x1F);
00966     sendDataByte(0x02);
00967     sendDataByte((track » 8) & 0xff);
00968     sendDataByte(track & 0xff);
00969     sendChecksum();
00970 }
00971
00972 void DFR0534::repeatPart(byte startMinute, byte startSecond, byte stopMinute, byte stopSecond )
00973 {
00974     if (m_ptrStream == NULL) return; // Should not happen
00975     sendStartingCode();
00976     sendDataByte(0x20);
00977     sendDataByte(0x04);
00978     sendDataByte(startMinute);
00979     sendDataByte(startSecond);
00980     sendDataByte(stopMinute);
00981     sendDataByte(stopSecond);
00982     sendChecksum();
00983 }
00984
00985 void DFR0534::stopRepeatPart()
00986 {
00987     if (m_ptrStream == NULL) return; // Should not happen
00988     sendStartingCode();
00989     sendDataByte(0x21);
00990     sendDataByte(0x00);
00991     sendChecksum();
00992 }
00993
00994 void DFR0534::fastBackwardDuration(word seconds)
00995 {
00996     if (m_ptrStream == NULL) return; // Should not happen
00997     sendStartingCode();
00998     sendDataByte(0x22);
00999     sendDataByte(0x02);
01000     sendDataByte((seconds » 8) & 0xff);
01001     sendDataByte(seconds & 0xff);
01002     sendChecksum();
01003 }
01004
01005 void DFR0534::fastForwardDuration(word seconds)
01006 {
01007     if (m_ptrStream == NULL) return; // Should not happen
01008     sendStartingCode();
01009     sendDataByte(0x23);
01010     sendDataByte(0x02);
01011     sendDataByte((seconds » 8) & 0xff);
01012     sendDataByte(seconds & 0xff);
01013     sendChecksum();
01014 }
01015
01016 bool DFR0534::getDuration(byte &hour, byte &minute, byte &second)
01017 {
01018     #define COMMAND 0x24

```

```

01062 #define RECEIVEFAILED false
01063 #define RECEIVEBYTETIMEOUTMS 100
01064 #define RECEIVEGLOBALTIMEOUTMS 500
01065 #define RECEIVEHEADERLENGTH 2 // startingcode+command
01066
01067 if (m_ptrStream == NULL) return false; // Should not happen
01068 sendStartingCode();
01069 sendDataByte(COMMAND);
01070 sendDataByte(0x00);
01071 sendChecksum();
01072
01073 // Receive
01074 int i=0;
01075 byte data, firstByte = 0, sum, length=0xff;
01076 word result = 0;
01077 unsigned long receiveStartMS = millis();
01078 do {
01079     byte dataReady = 0;
01080     unsigned long lastMS = millis();
01081     // Wait for response or timeout
01082     while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01083
01084     if (dataReady == 0) return RECEIVEFAILED; // Timeout
01085     data = m_ptrStream->read();
01086
01087     if (i==0) { // Begin of transmission
01088         firstByte=data;
01089         sum = 0;
01090     }
01091     if ((i == 1) && (data != COMMAND)) {
01092         // Invalid signal => reset receive
01093         i=0;
01094         firstByte = 0;
01095     }
01096     if (i == RECEIVEHEADERLENGTH) {
01097         length = data; // Length of receiving data
01098         if (length != 3) {
01099             // Invalid length => reset receive
01100             i=0;
01101             firstByte = 0;
01102         }
01103     }
01104     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01105         switch (i-RECEIVEHEADERLENGTH-1) {
01106             case 0:
01107                 hour=data;
01108                 break;
01109             case 1:
01110                 minute=data;
01111                 break;
01112             case 2:
01113                 second=data;
01114                 break;
01115         }
01116     }
01117     if (firstByte == STARTINGCODE) {
01118         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01119         i++;
01120     }
01121     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01122 } while (i<length+RECEIVEHEADERLENGTH+2);
01123
01124 return (data == sum); // Does checksum matches?
01125 }
01126
01130 void DFR0534::startSendingRuntime()
01131 {
01132     if (m_ptrStream == NULL) return; // Should not happen
01133     sendStartingCode();
01134     sendDataByte(0x25);
01135     sendDataByte(0x00);
01136     sendChecksum();
01137 }
01138
01152 bool DFR0534::getRuntime(byte &hour, byte &minute, byte &second)
01153 {
01154     #define COMMAND 0x25
01155     #define RECEIVEFAILED false
01156     #define RECEIVEBYTETIMEOUTMS 100
01157     #define RECEIVEGLOBALTIMEOUTMS 500
01158     #define RECEIVEHEADERLENGTH 2 // startingcode+command
01159
01160     if (m_ptrStream == NULL) return false; // Should not happen
01161
01162     // Receive
01163     int i=0;

```

```

01164   byte data, firstByte = 0, sum, length=0xff;
01165   word result = 0;
01166   unsigned long receiveStartMS = millis();
01167   do {
01168       byte dataReady = 0;
01169       unsigned long lastMS = millis();
01170       // Wait for response or timeout
01171       while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01172
01173       if (dataReady == 0) return RECEIVEFAILED; // Timeout
01174       data = m_ptrStream->read();
01175
01176       if (i==0) { // Begin of transmission
01177           firstByte=data;
01178           sum = 0;
01179       }
01180       if ((i == 1) && (data != COMMAND)) {
01181           // Invalid signal => reset receive
01182           i=0;
01183           firstByte = 0;
01184       }
01185       if (i == RECEIVEHEADERLENGTH) {
01186           length = data; // Length of receiving data
01187           if (length != 3) {
01188               // Invalid length => reset receive
01189               i=0;
01190               firstByte = 0;
01191           }
01192       }
01193       if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01194           switch (i-RECEIVEHEADERLENGTH-1) {
01195               case 0:
01196                   hour=data;
01197                   break;
01198               case 1:
01199                   minute=data;
01200                   break;
01201               case 2:
01202                   second=data;
01203                   break;
01204           }
01205       }
01206       if (firstByte == STARTINGCODE) {
01207           if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01208           i++;
01209       }
01210       if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01211   } while (i<length+RECEIVEHEADERLENGTH+2);
01212
01213   return (data == sum); // Does checksum matches?
01214 }
01215
01219 void DFR0534::stopSendingRuntime()
01220 {
01221     if (m_ptrStream == NULL) return; // Should not happen
01222     sendStartingCode();
01223     sendDataByte(0x26);
01224     sendDataByte(0x00);
01225     sendChecksum();
01226 }

```

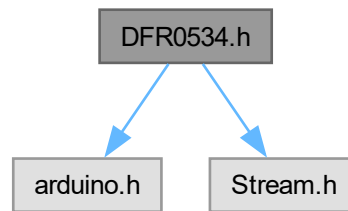
## 5.6 DFR0534.h File Reference

```

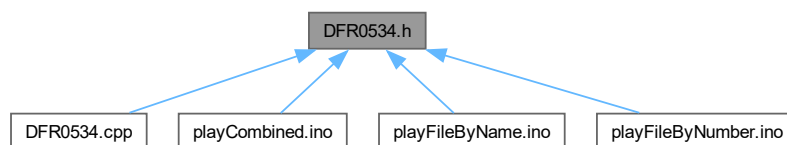
#include <arduino.h>
#include <Stream.h>

```

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 "1.0.1"`

### 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

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

1.0.1

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

## 5.6.2 Macro Definition Documentation

### 5.6.2.1 DFR0534\_VERSION

```
#define DFR0534_VERSION "1.0.1"
```

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 "1.0.1"
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     EQUUNKNOWN
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 getFirstFileNumberInCurrentDirectory();
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, [9](#)  
CHANNELMP3  
    DFR0534, [9](#)  
CHANNELMP3AUX  
    DFR0534, [9](#)  
CHANNELUNKNOWN  
    DFR0534, [9](#)  
  
decreaseVolume  
    DFR0534, [11](#)  
DFR0534, [1](#), [7](#)  
    CHANNELAUX, [9](#)  
    CHANNELMP3, [9](#)  
    CHANNELMP3AUX, [9](#)  
    CHANNELUNKNOWN, [9](#)  
    decreaseVolume, [11](#)  
    DFR0534, [11](#)  
    DFR0534CHANNELS, [9](#)  
    DFR0534DRIVE, [9](#)  
    DFR0534EQ, [10](#)  
    DFR0534LOOPMODE, [10](#)  
    DFR0534STATUS, [11](#)  
    DIRECTORYLOOP, [10](#)  
    DRIVEFLASH, [10](#)  
    DRIVENO, [10](#)  
    DRIVESD, [10](#)  
    DRIVEUNKNOWN, [10](#)  
    DRIVEUSB, [10](#)  
    fastBackwardDuration, [12](#)  
    fastForwardDuration, [12](#)  
    getDrive, [12](#)  
    getDrivesStates, [13](#)  
    getDuration, [14](#)  
    getFileName, [16](#)  
    getFileName, [17](#)  
    getFirstFileNameInCurrentDirectory, [18](#)  
    getRuntime, [19](#)  
    getStatus, [20](#)  
    getTotalFiles, [21](#)  
    getTotalFilesInCurrentDirectory, [22](#)  
    increaseVolume, [23](#)  
    insertFileByNumber, [23](#)  
    LOOPBACKALL, [10](#)  
    NORMAL, [10](#)  
    pause, [24](#)  
    PAUSED, [11](#)  
    play, [24](#)  
    playCombined, [24](#)  
    playFileByName, [25](#)  
    playFileByNumber, [26](#)  
    PLAYING, [11](#)  
    playLastInDirectory, [26](#)  
    PLAYMODEUNKNOWN, [10](#)  
    playNext, [26](#)  
    playNextDirectory, [27](#)  
    playPrevious, [27](#)  
    PLAYRANDOM, [10](#)  
    prepareFileByNumber, [27](#)  
    RANDOMINDIRECTORY, [10](#)  
    repeatPart, [28](#)  
    SEQUENTIAL, [10](#)  
    SEQUENTIALINDIRECTORY, [10](#)  
    setChannel, [28](#)  
    setDirectory, [29](#)  
    setDrive, [29](#)  
    setEqualizer, [29](#)  
    setLoopMode, [30](#)  
    setRepeatLoops, [30](#)  
    setVolume, [31](#)  
    SINGLEAUDIOLOOP, [10](#)  
    SINGLEAUDIOSTOP, [10](#)  
    startSendingRuntime, [31](#)  
    STATUSUNKNOWN, [11](#)  
    stop, [31](#)  
    stopCombined, [32](#)  
    stopInsertedFile, [32](#)  
    STOPPED, [11](#)  
    stopRepeatPart, [32](#)  
    stopSendingRuntime, [32](#)  
DFR0534.cpp, [38](#), [39](#)  
DFR0534.h, [50](#), [52](#)  
    DFR0534\_VERSION, [52](#)  
DFR0534\_VERSION  
    DFR0534.h, [52](#)  
DFR0534CHANNELS  
    DFR0534, [9](#)  
DFR0534DRIVE  
    DFR0534, [9](#)  
DFR0534EQ  
    DFR0534, [10](#)  
DFR0534LOOPMODE  
    DFR0534, [10](#)  
DFR0534STATUS  
    DFR0534, [11](#)  
DIRECTORYLOOP  
    DFR0534, [10](#)  
DRIVEFLASH  
    DFR0534, [10](#)

- DRIVENO
  - DFR0534, [10](#)
- DRIVESD
  - DFR0534, [10](#)
- DRIVEUNKNOWN
  - DFR0534, [10](#)
- DRIVEUSB
  - DFR0534, [10](#)
- fastBackwardDuration
  - DFR0534, [12](#)
- fastForwardDuration
  - DFR0534, [12](#)
- getDrive
  - DFR0534, [12](#)
- getDrivesStates
  - DFR0534, [13](#)
- getDuration
  - DFR0534, [14](#)
- getFileName
  - DFR0534, [16](#)
- getFileNumber
  - DFR0534, [17](#)
- getFirstFileNumberInCurrentDirectory
  - DFR0534, [18](#)
- getRuntime
  - DFR0534, [19](#)
- getStatus
  - DFR0534, [20](#)
- getTotalFiles
  - DFR0534, [21](#)
- getTotalFilesInCurrentDirectory
  - DFR0534, [22](#)
- increaseVolume
  - DFR0534, [23](#)
- insertFileByNumber
  - DFR0534, [23](#)
- LOOPBACKALL
  - DFR0534, [10](#)
- NORMAL
  - DFR0534, [10](#)
- pause
  - DFR0534, [24](#)
- PAUSED
  - DFR0534, [11](#)
- play
  - DFR0534, [24](#)
- playCombined
  - DFR0534, [24](#)
- playCombined.ino, [35](#)
- playFileByName
  - DFR0534, [25](#)
- playFileByName.ino, [36](#)
- playFileByNumber
  - DFR0534, [26](#)
- playFileByNumber.ino, [37](#)
- PLAYING
  - DFR0534, [11](#)
- playLastInDirectory
  - DFR0534, [26](#)
- PLAYMODEUNKNOWN
  - DFR0534, [10](#)
- playNext
  - DFR0534, [26](#)
- playNextDirectory
  - DFR0534, [27](#)
- playPrevious
  - DFR0534, [27](#)
- PLAYRANDOM
  - DFR0534, [10](#)
- prepareFileByNumber
  - DFR0534, [27](#)
- RANDOMINDIRECTORY
  - DFR0534, [10](#)
- repeatPart
  - DFR0534, [28](#)
- SEQUENTIAL
  - DFR0534, [10](#)
- SEQUENTIALINDIRECTORY
  - DFR0534, [10](#)
- setChannel
  - DFR0534, [28](#)
- setDirectory
  - DFR0534, [29](#)
- setDrive
  - DFR0534, [29](#)
- setEqualizer
  - DFR0534, [29](#)
- setLoopMode
  - DFR0534, [30](#)
- setRepeatLoops
  - DFR0534, [30](#)
- setVolume
  - DFR0534, [31](#)
- SINGLEAUDIOLOOP
  - DFR0534, [10](#)
- SINGLEAUDIOSTOP
  - DFR0534, [10](#)
- startSendingRuntime
  - DFR0534, [31](#)
- STATUSUNKNOWN
  - DFR0534, [11](#)
- stop
  - DFR0534, [31](#)
- stopCombined
  - DFR0534, [32](#)
- stopInsertedFile
  - DFR0534, [32](#)
- STOPPED
  - DFR0534, [11](#)
- stopRepeatPart
  - DFR0534, [32](#)

stopSendingRuntime  
DFR0534, [32](#)