

DFR0534

1.0.2

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()	32
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 DFR0534 pinout

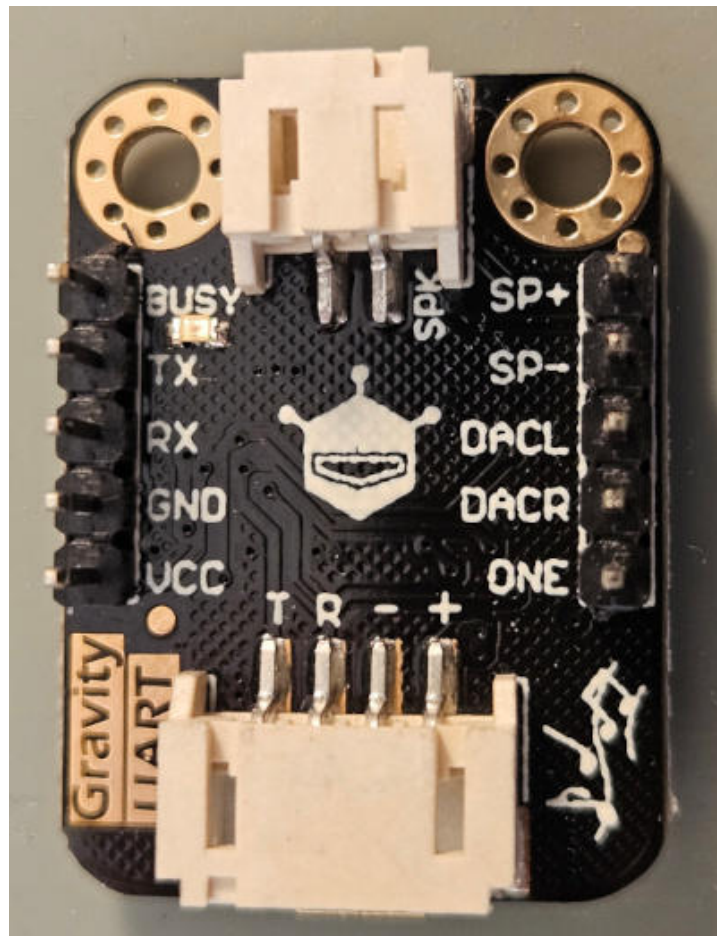


Figure 1.1 DFR0534

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

```
00748 {
00749     if (m_ptrStream == NULL) return; // Should not happen
00750     sendStartingCode();
00751     sendDataByte(0x15);
00752     sendDataByte(0x00);
00753     sendChecksum();
00754 }
```

#### 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 1024 of file [DFR0534.cpp](#).

```
01025 {
01026     if (m_ptrStream == NULL) return; // Should not happen
01027     sendStartingCode();
01028     sendDataByte(0x22);
01029     sendDataByte(0x02);
01030     sendDataByte((seconds >> 8) & 0xff);
01031     sendDataByte(seconds & 0xff);
01032     sendChecksum();
01033 }
```

#### 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 1041 of file [DFR0534.cpp](#).

```
01042 {
01043     if (m_ptrStream == NULL) return; // Should not happen
01044     sendStartingCode();
01045     sendDataByte(0x23);
01046     sendDataByte(0x02);
01047     sendDataByte((seconds >> 8) & 0xff);
01048     sendDataByte(seconds & 0xff);
01049     sendChecksum();
01050 }
```

#### 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 344 of file `DFR0534.cpp`.

```

00345 {
00346     #define COMMAND 0x0A
00347     #define RECEIVEBYTETIMEOUTMS 100
00348     #define RECEIVEGLOBALTIMEOUTMS 500
00349     #define RECEIVEFAILED DRIVEUNKNOWN
00350     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00351
00352     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00353     sendStartingCode();
00354     sendDataByte(COMMAND);
00355     sendDataByte(0x00);
00356     sendChecksum();
00357
00358     // Receive
00359     int i=0;
00360     byte data, firstByte = 0, sum, length=0xff, result = 0;
00361     unsigned long receiveStartMS = millis();
00362     do {
00363         byte dataReady = 0;
00364         unsigned long lastMS = millis();
00365         // Wait for response or timeout
00366         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00367
00368         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00369         data = m_ptrStream->read();
00370
00371         if (i==0) { // Begin of transmission
00372             firstByte=data;
00373             sum = 0;
00374         }
00375         if ((i == 1) && (data != COMMAND)) {
00376             // Invalid signal => reset receive
00377             i=0;
00378             firstByte = 0;
00379         }
00380         if (i == RECEIVEHEADERLENGTH) {
00381             length = data; // Length of receiving data
00382             if (length != 1) {
00383                 // Invalid length => reset receive
00384                 i=0;
00385                 firstByte = 0;
00386             }
00387         }
00388         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00389             result = data;
00390         }
00391         if (firstByte == STARTINGCODE) {
00392             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00393             i++;
00394         }
00395         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00396     } while (i<length+RECEIVEHEADERLENGTH+2);
00397
00398     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00399     return result;
00400 }

```

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

```

00278 {
00279     #define COMMAND 0x09
00280     #define RECEIVEBYTETIMEOUTMS 100
00281     #define RECEIVEGLOBALTIMEOUTMS 500
00282     #define RECEIVEFAILED DRIVEUNKNOWN
00283     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00284
00285     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00286     sendStartingCode();
00287     sendDataByte(COMMAND);
00288     sendDataByte(0x00);
00289     sendChecksum();
00290
00291     // Receive
00292     int i=0;
00293     byte data, firstByte = 0, sum, length=0xff, result = 0;
00294     unsigned long receiveStartMS = millis();
00295     do {
00296         byte dataReady = 0;
00297         unsigned long lastMS = millis();
00298         // Wait for response or timeout
00299         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00300
00301         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00302         data = m_ptrStream->read();
00303
00304         if (i==0) { // Begin of transmission
00305             firstByte=data;
00306             sum = 0;
00307         }
00308         if ((i == 1) && (data != COMMAND)) {
00309             // Invalid signal => reset receive
00310             i=0;
00311             firstByte = 0;
00312         }
00313         if (i == RECEIVEHEADERLENGTH) {
00314             length = data; // Length of receiving data
00315             if (length != 1) {
00316                 // Invalid length => reset receive
00317                 i=0;
00318                 firstByte = 0;
00319             }
00320         }
00321         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00322             result = data;
00323         }
00324         if (firstByte == STARTINGCODE) {
00325             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00326             i++;
00327         }
00328         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00329     } while (i<length+RECEIVEHEADERLENGTH+2);
00330
00331     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00332     return result;
00333 }
```

## 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 1064 of file DFR0534.cpp.

```
01065 {
01066     #define COMMAND 0x24
01067     #define RECEIVEFAILED false
01068     #define RECEIVEBYTETIMEOUTMS 100
01069     #define RECEIVEGLOBALTIMEOUTMS 500
01070     #define RECEIVEHEADERLENGTH 2 // startingcode+command
01071
01072     if (m_ptrStream == NULL) return false; // Should not happen
01073     sendStartingCode();
01074     sendDataByte(COMMAND);
01075     sendDataByte(0x00);
01076     sendChecksum();
01077
01078     // Receive
01079     int i=0;
01080     byte data, firstByte = 0, sum, length=0xff;
01081     word result = 0;
01082     unsigned long receiveStartMS = millis();
01083     do {
01084         byte dataReady = 0;
01085         unsigned long lastMS = millis();
01086         // Wait for response or timeout
01087         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01088
01089         if (dataReady == 0) return RECEIVEFAILED; // Timeout
01090         data = m_ptrStream->read();
01091
01092         if (i==0) { // Begin of transmission
01093             firstByte=data;
01094             sum = 0;
01095         }
01096         if ((i == 1) && (data != COMMAND)) {
01097             // Invalid signal => reset receive
01098             i=0;
01099             firstByte = 0;
01100         }
01101         if (i == RECEIVEHEADERLENGTH) {
01102             length = data; // Length of receiving data
01103             if (length != 3) {
01104                 // Invalid length => reset receive
01105                 i=0;
01106                 firstByte = 0;
01107             }
01108         }
01109         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01110             switch (i-RECEIVEHEADERLENGTH-1) {
01111                 case 0:
```

```

01112         hour=data;
01113         break;
01114     case 1:
01115         minute=data;
01116         break;
01117     case 2:
01118         second=data;
01119         break;
01120     }
01121 }
01122 if (firstByte == STARTINGCODE) {
01123     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01124     i++;
01125 }
01126 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01127 } while (i<length+RECEIVEHEADERLENGTH+2);
01128
01129 return (data == sum); // Does checksum matches?
01130 }

```

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

```

00913 {
00914     #define COMMAND 0x1E
00915     #define RECEIVEBYTETIMEOUTMS 100
00916     #define RECEIVEGLOBALTIMEOUTMS 500
00917     #define RECEIVEFAILED false
00918     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00919
00920     if (m_ptrStream == NULL) return false; // Should not happen
00921     if (name == NULL) return false;
00922     name[0] = '\0';
00923
00924     sendStartingCode();
00925     sendDataByte(COMMAND);
00926     sendDataByte(0x00);
00927     sendCheckSum();
00928
00929     // Receive
00930     int i=0;
00931     byte data, firstByte = 0, sum, length=0xff;
00932     unsigned long receiveStartMS = millis();
00933     do {
00934         byte dataReady = 0;
00935         unsigned long lastMS = millis();
00936         // Wait for response or timeout
00937         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00938
00939         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00940         data = m_ptrStream->read();
00941         if (i==0) { // Begin of transmission
00942             firstByte=data;
00943             sum = 0;
00944         }
00945         if ((i == 1) && (data != COMMAND)) {
00946             // Invalid signal => reset receive
00947             i=0;
00948             firstByte = 0;
00949         }
00950         if (i == RECEIVEHEADERLENGTH) length = data; // Length of receiving string

```

```

00951     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00952         if ((i-RECEIVEHEADERLENGTH) < 12) { // I expect no longer file names than 8+3 chars plus '\0'
00953             name[i-RECEIVEHEADERLENGTH-1] = data;
00954             name[i-RECEIVEHEADERLENGTH] = '\0';
00955         }
00956     }
00957     if (firstByte == STARTINGCODE) {
00958         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00959         i++;
00960     }
00961     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00962     } while (i<length+RECEIVEHEADERLENGTH+2);
00963     return (data == sum); // Does checksum matches?
00964 }

```

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

```

00427 {
00428     #define COMMAND 0x0D
00429     #define RECEIVEFAILED 0
00430     #define RECEIVEBYTETIMEOUTMS 100
00431     #define RECEIVEGLOBALTIMEOUTMS 500
00432     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00433
00434     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00435     sendStartingCode();
00436     sendDataByte(COMMAND);
00437     sendDataByte(0x00);
00438     sendChecksum();
00439
00440     // Receive
00441     int i=0;
00442     byte data, firstByte = 0, sum, length=0xff;
00443     word result = 0;
00444     unsigned long receiveStartMS = millis();
00445     do {
00446         byte dataReady = 0;
00447         unsigned long lastMS = millis();
00448         // Wait for response or timeout
00449         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00450
00451         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00452         data = m_ptrStream->read();
00453
00454         if (i==0) { // Begin of transmission
00455             firstByte=data;
00456             sum = 0;
00457         }
00458         if ((i == 1) && (data != COMMAND)) {
00459             // Invalid signal => reset receive
00460             i=0;
00461             firstByte = 0;
00462         }
00463         if (i == RECEIVEHEADERLENGTH) {
00464             length = data; // Length of receiving data
00465             if (length != 2) {

```

```

00466         // Invalid length => reset receive
00467         i=0;
00468         firstByte = 0;
00469     }
00470 }
00471 if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00472     switch (i-RECEIVEHEADERLENGTH-1) {
00473         case 0:
00474             result=data<<8;
00475             break;
00476         case 1:
00477             result+=data;
00478             break;
00479     }
00480 }
00481 if (firstByte == STARTINGCODE) {
00482     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00483     i++;
00484 }
00485 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00486 } while (i<length+RECEIVEHEADERLENGTH+2);
00487
00488 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00489 return result;
00490 }

```

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

```

00595 {
00596     #define COMMAND 0x11
00597     #define RECEIVEFAILED -1
00598     #define RECEIVEBYTETIMEOUTMS 100
00599     #define RECEIVEGLOBALTIMEOUTMS 500
00600     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00601
00602     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00603     sendStartingCode();
00604     sendDataByte(COMMAND);
00605     sendDataByte(0x00);
00606     sendChecksum();
00607
00608     // Receive
00609     int i=0;
00610     byte data, firstByte = 0, sum, length=0xff;
00611     word result = 0;
00612     unsigned long receiveStartMS = millis();
00613     do {
00614         byte dataReady = 0;
00615         unsigned long lastMS = millis();
00616         // Wait for response or timeout
00617         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00618
00619         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00620         data = m_ptrStream->read();
00621
00622         if (i==0) { // Begin of transmission
00623             firstByte=data;
00624             sum = 0;
00625         }

```

```

00626     if ((i == 1) && (data != COMMAND)) {
00627         // Invalid signal => reset receive
00628         i=0;
00629         firstByte = 0;
00630     }
00631     if (i == RECEIVEHEADERLENGTH) {
00632         length = data; // Length of receiving data
00633         if (length != 2) {
00634             // Invalid length => reset receive
00635             i=0;
00636             firstByte = 0;
00637         }
00638     }
00639     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00640         switch (i-RECEIVEHEADERLENGTH-1) {
00641             case 0:
00642                 result=data<<8;
00643                 break;
00644             case 1:
00645                 result+=data;
00646                 break;
00647         }
00648     }
00649     if (firstByte == STARTINGCODE) {
00650         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00651         i++;
00652     }
00653     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00654 } while (i<length+RECEIVEHEADERLENGTH+2);
00655
00656 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00657 return result;
00658 }

```

#### 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 1157 of file [DFR0534.cpp](#).

```

1158 {
1159     #define COMMAND 0x25
1160     #define RECEIVEFAILED false
1161     #define RECEIVEBYTETIMEOUTMS 100
1162     #define RECEIVEGLOBALTIMEOUTMS 500
1163     #define RECEIVEHEADERLENGTH 2 // startingcode+command
1164
1165     if (m_ptrStream == NULL) return false; // Should not happen
1166
1167     // Receive
1168     int i=0;

```

```

01169     byte data, firstByte = 0, sum, length=0xff;
01170     word result = 0;
01171     unsigned long receiveStartMS = millis();
01172     do {
01173         byte dataReady = 0;
01174         unsigned long lastMS = millis();
01175         // Wait for response or timeout
01176         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01177
01178         if (dataReady == 0) return RECEIVEFAILED; // Timeout
01179         data = m_ptrStream->read();
01180
01181         if (i==0) { // Begin of transmission
01182             firstByte=data;
01183             sum = 0;
01184         }
01185         if ((i == 1) && (data != COMMAND)) {
01186             // Invalid signal => reset receive
01187             i=0;
01188             firstByte = 0;
01189         }
01190         if (i == RECEIVEHEADERLENGTH) {
01191             length = data; // Length of receiving data
01192             if (length != 3) {
01193                 // Invalid length => reset receive
01194                 i=0;
01195                 firstByte = 0;
01196             }
01197         }
01198         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01199             switch (i-RECEIVEHEADERLENGTH-1) {
01200                 case 0:
01201                     hour=data;
01202                     break;
01203                 case 1:
01204                     minute=data;
01205                     break;
01206                 case 2:
01207                     second=data;
01208                     break;
01209             }
01210         }
01211         if (firstByte == STARTINGCODE) {
01212             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01213             i++;
01214         }
01215         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01216     } while (i<length+RECEIVEHEADERLENGTH+2);
01217     return (data == sum); // Does checksum matches?
01218 }
01219 }

```

#### 4.1.4.11 getStatus()

byte DFR0534::getStatus ( )

Get module status.

##### Return values

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

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

```

00054 {
00055     #define COMMAND 0x01
00056     #define RECEIVEBYTETIMEOUTMS 100
00057     #define RECEIVEGLOBALTIMEOUTMS 500
00058     #define RECEIVEFAILED STATUSUNKNOWN
00059     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00060

```



```

00061  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00062  sendStartingCode();
00063  sendDataByte(COMMAND);
00064  sendDataByte(0x00);
00065  sendChecksum();
00066
00067  // Receive
00068  int i=0;
00069  byte data, firstByte = 0, sum, length=0xff, result = 0;
00070  unsigned long receiveStartMS = millis();
00071  do {
00072      byte dataReady = 0;
00073      unsigned long lastMS = millis();
00074      // Wait for response or timeout
00075      while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00076
00077      if (dataReady == 0) return RECEIVEFAILED; // Timeout
00078      data = m_ptrStream->read();
00079
00080      if (i==0) { // Begin of transmission
00081          firstByte=data;
00082          sum = 0;
00083      }
00084      if ((i == 1) && (data != COMMAND)) {
00085          // Invalid signal => reset receive
00086          i=0;
00087          firstByte = 0;
00088      }
00089      if (i == RECEIVEHEADERLENGTH) {
00090          length = data; // Length of receiving data
00091          if (length != 1) {
00092              // Invalid length => reset receive
00093              i=0;
00094              firstByte = 0;
00095          }
00096      }
00097      if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00098          result = data;
00099      }
00100      if (firstByte == STARTINGCODE) {
00101          if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00102          i++;
00103      }
00104      if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00105  } while (i<length+RECEIVEHEADERLENGTH+2);
00106
00107  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00108  return result;
00109 }

```

#### 4.1.4.12 getTotalFiles()

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

Get total number of supported audio files on current drive.

##### Returns

Number of files

##### Return values

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

Definition at line 498 of file [DFR0534.cpp](#).

```

00499 {
00500     #define COMMAND 0x0C
00501     #define RECEIVEFAILED -1
00502     #define RECEIVEBYTETIMEOUTMS 100
00503     #define RECEIVEGLOBALTIMEOUTMS 500
00504     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00505

```

```

00506     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00507     sendStartingCode();
00508     sendDataByte(COMMAND);
00509     sendDataByte(0x00);
00510     sendChecksum();
00511
00512     // Receive
00513     int i=0;
00514     byte data, firstByte = 0, sum, length=0xff;
00515     word result = 0;
00516     unsigned long receiveStartMS = millis();
00517     do {
00518         byte dataReady = 0;
00519         unsigned long lastMS = millis();
00520         // Wait for response or timeout
00521         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00522
00523         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00524         data = m_ptrStream->read();
00525
00526         if (i==0) { // Begin of transmission
00527             firstByte=data;
00528             sum = 0;
00529         }
00530         if ((i == 1) && (data != COMMAND)) {
00531             // Invalid signal => reset receive
00532             i=0;
00533             firstByte = 0;
00534         }
00535         if (i == RECEIVEHEADERLENGTH) {
00536             length = data; // Length of receiving data
00537             if (length != 2) {
00538                 // Invalid length => reset receive
00539                 i=0;
00540                 firstByte = 0;
00541             }
00542         }
00543         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00544             switch (i-RECEIVEHEADERLENGTH-1) {
00545                 case 0:
00546                     result=data<<8;
00547                     break;
00548                 case 1:
00549                     result+=data;
00550                     break;
00551             }
00552         }
00553         if (firstByte == STARTINGCODE) {
00554             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00555             i++;
00556         }
00557         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00558     } while (i<length+RECEIVEHEADERLENGTH+2);
00559
00560     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00561     return result;
00562 }

```

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

```

00667 {
00668     #define COMMAND 0x12
00669     #define RECEIVEFAILED -1
00670     #define RECEIVEBYTETIMEOUTMS 100
00671     #define RECEIVEGLOBALTIMEOUTMS 500
00672     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00673
00674     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00675     sendStartingCode();
00676     sendDataByte(COMMAND);
00677     sendDataByte(0x00);
00678     sendChecksum();
00679
00680     // Receive
00681     int i=0;
00682     byte data, firstByte = 0, sum, length=0xff;
00683     word result = 0;
00684     unsigned long receiveStartMS = millis();
00685     do {
00686         byte dataReady = 0;
00687         unsigned long lastMS = millis();
00688         // Wait for response or timeout
00689         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00690
00691         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00692         data = m_ptrStream->read();
00693
00694         if (i==0) { // Begin of transmission
00695             firstByte=data;
00696             sum = 0;
00697         }
00698         if ((i == 1) && (data != COMMAND)) {
00699             // Invalid signal => reset receive
00700             i=0;
00701             firstByte = 0;
00702         }
00703         if (i == RECEIVEHEADERLENGTH) {
00704             length = data; // Length of receiving data
00705             if (length != 2) {
00706                 // Invalid length => reset receive
00707                 i=0;
00708                 firstByte = 0;
00709             }
00710         }
00711         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00712             switch (i-RECEIVEHEADERLENGTH-1) {
00713                 case 0:
00714                     result=data<<8;
00715                     break;
00716                 case 1:
00717                     result+=data;
00718                     break;
00719             }
00720         }
00721         if (firstByte == STARTINGCODE) {
00722             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00723             i++;
00724         }
00725         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00726     } while (i<length+RECEIVEHEADERLENGTH+2);
00727
00728     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00729     return result;
00730 }

```

#### 4.1.4.14 increaseVolume()

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

Increase volume by one step.

Definition at line 735 of file [DFR0534.cpp](#).

```

00736 {
00737     if (m_ptrStream == NULL) return; // Should not happen
00738     sendStartingCode();
00739     sendDataByte(0x14);
00740     sendDataByte(0x00);
00741     sendChecksum();
00742 }

```

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

```
00765 {
00766     if (m_ptrStream == NULL) return; // Should not happen
00767     if (drive >= DRIVEUNKNOWN) return;
00768     sendStartingCode();
00769     sendDataByte(0x16);
00770     sendDataByte(0x03);
00771     sendDataByte(drive);
00772     sendDataByte((track >> 8) & 0xff);
00773     sendDataByte(track & 0xff);
00774     sendChecksum();
00775 }
```

#### 4.1.4.16 pause()

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

Pause the current file.

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

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

#### 4.1.4.17 play()

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

Play the current selected file.

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

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

#### 4.1.4.18 playCombined()

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

Combined/concatenated play of files.

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

##### Parameters

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

Definition at line 857 of file [DFR0534.cpp](#).

```
00858 {
00859     if (m_ptrStream == NULL) return; // Should not happen
00860     if (list == NULL) return;
00861     if ((strlen(list) % 2) != 0) return;
00862
00863     sendStartingCode();
00864     sendDataByte(0x1B);
00865     sendDataByte(strlen(list));
00866     for (int i=0;i<strlen(list);i++) {
00867         sendDataByte(list[i]);
00868     }
00869     sendChecksum();
00870 }
```

#### 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
- maximal 8 characters
- Every file and folder whose name length is shorter then 8 chars must be filled up to the 8 chars length by space chars
- must end with WAV or MP3
- Only WAV and MP3 files are supported
- Wildcards \* (=multiple arbitrary characters) and ? (=one single arbitrary character) are allowed and can be used to reduce the filling space chars

Valid examples:

- "/01 WAV" for file 01.wav

- "/99-AFR~1MP3" for a file /99-Africa.mp3
- "/SUN\*MP3" for first file matching /sun\*.mp3, for example '/sun.mp3' (in this order for example: sun0.mp3 sun.mp3 sun1.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 251 of file [DFR0534.cpp](#).

```

00252 {
00253     if (m_ptrStream == NULL) return; // Should not happen
00254     if (path == NULL) return;
00255     if (drive >= DRIVEUNKNOWN) return;
00256     sendStartingCode();
00257     sendDataByte(0x08);
00258     sendDataByte(strlen(path)+1);
00259     sendDataByte(drive);
00260     for (int i=0;i<strlen(path);i++) {
00261         sendDataByte(path[i]);
00262     }
00263     sendChecksum();
00264 }
```

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

```
00568 {
00569     if (m_ptrStream == NULL) return; // Should not happen
00570     sendStartingCode();
00571     sendDataByte(0x0E);
00572     sendDataByte(0x00);
00573     sendChecksum();
00574 }
```

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

```
00580 {
00581     if (m_ptrStream == NULL) return; // Should not happen
00582     sendStartingCode();
00583     sendDataByte(0x0F);
00584     sendDataByte(0x00);
00585     sendChecksum();
00586 }
```

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

```
00972 {
00973     if (m_ptrStream == NULL) return; // Should not happen
00974     sendStartingCode();
00975     sendDataByte(0x1F);
00976     sendDataByte(0x02);
00977     sendDataByte((track » 8) & 0xff);
00978     sendDataByte(track & 0xff);
00979     sendChecksum();
00980 }
```

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

```
00993 {
00994     if (m_ptrStream == NULL) return; // Should not happen
00995     sendStartingCode();
00996     sendDataByte(0x20);
00997     sendDataByte(0x04);
00998     sendDataByte(startMinute);
00999     sendDataByte(startSecond);
01000     sendDataByte(stopMinute);
01001     sendDataByte(stopSecond);
01002     sendChecksum();
01003 }
```

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

```
00893 {
00894     if (m_ptrStream == NULL) return; // Should not happen
00895     if (channel >= CHANNELUNKNOWN) return;
00896     sendStartingCode();
00897     sendDataByte(0x1D);
00898     sendDataByte(0x01);
00899     sendDataByte(channel);
00900     sendChecksum();
00901 }
```

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

```
00798 {
00799     if (m_ptrStream == NULL) return; // Should not happen
00800     if (path == NULL) return;
00801     if (drive >= DRIVEUNKNOWN) return;
00802     sendStartingCode();
00803     sendDataByte(0x17);
00804     sendDataByte(strlen(path)+1);
00805     sendDataByte(drive);
00806     for (int i=0; i<strlen(path); i++) {
00807         sendDataByte(path[i]);
00808     }
00809     sendChecksum();
00810 }
```

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

```
00408 {
```

```

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

```

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

```

00818 {
00819     if (m_ptrStream == NULL) return; // Should not happen
00820     if (mode >= PLAYMODEUNKNOWN) return;
00821     sendStartingCode();
00822     sendDataByte(0x18);
00823     sendDataByte(0x01);
00824     sendDataByte(mode);
00825     sendChecksum();
00826 }

```

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

```
00836 {
00837     if (m_ptrStream == NULL) return; // Should not happen
00838     sendStartingCode();
00839     sendDataByte(0x19);
00840     sendDataByte(0x02);
00841     sendDataByte((loops >> 8) & 0xff);
00842     sendDataByte(loops & 0xff);
00843     sendChecksum();
00844 }
```

#### 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 1135 of file [DFR0534.cpp](#).

```
01136 {
01137     if (m_ptrStream == NULL) return; // Should not happen
01138     sendStartingCode();
01139     sendDataByte(0x25);
01140     sendDataByte(0x00);
01141     sendChecksum();
01142 }
```

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

```
00876 {
00877     if (m_ptrStream == NULL) return; // Should not happen
00878     sendStartingCode();
00879     sendDataByte(0x1C);
00880     sendDataByte(0x00);
00881     sendChecksum();
00882 }
```

#### 4.1.4.37 stopInsertedFile()

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

Stop inserted file.

Continue original file

Definition at line 782 of file [DFR0534.cpp](#).

```
00783 {
00784     if (m_ptrStream == NULL) return; // Should not happen
00785     sendStartingCode();
00786     sendDataByte(0x10);
00787     sendDataByte(0x00);
00788     sendChecksum();
00789 }
```

#### 4.1.4.38 stopRepeatPart()

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

Stop repeating part of the current file.

Definition at line 1008 of file [DFR0534.cpp](#).

```
01009 {
01010     if (m_ptrStream == NULL) return; // Should not happen
01011     sendStartingCode();
01012     sendDataByte(0x21);
01013     sendDataByte(0x00);
01014     sendChecksum();
01015 }
```

#### 4.1.4.39 stopSendingRuntime()

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

Stop sending runtime.

Definition at line 1224 of file [DFR0534.cpp](#).

```
01225 {  
01226     if (m_ptrStream == NULL) return; // Should not happen  
01227     sendStartingCode();  
01228     sendDataByte(0x26);  
01229     sendDataByte(0x00);  
01230     sendChecksum();  
01231 }
```

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
00023      * full path of the audio file to be played in format which looks like
00024      * a special unix 8+3 format:
00025      * - Without the dot for the file extension
00026      * - All characters in upper case
00027      * - maximal 8 characters
00028      * - Every file and folder whose name length is shorter then 8 chars
00029      * - must be filled up to the 8 chars length by space chars
00030      * - must end with WAV or MP3
00031      * - Only WAV and MP3 files are supported
00032      * - Wildcards * (=multiple arbitrary characters) and ? (one single arbitrary character)
00033      *   are allowed and can be used to reduce the filling space chars
00034      *
00035      * Valid examples:
00036      * - "/01      WAV" for file '/01.wav'
00037      * - "/99-AFR-1MP3" for a file '/99-Africa.mp3'
00038      * - "/SUN*MP3" for first file matching /sun*.mp3, for example '/sun.mp3' (in this order for
00039      *   example: sun0.mp3 sun.mp3 sun1.mp3)
00040      * - "/99-AFR*MP3" for first file matching '/99-Afr*.mp3'
00041      * - "/10*" for first* audio file matching /10*.*
00042      * - "/10      /20      WAV" for the file /10/20.wav
00043      *
00044      * You can get example files from
00045      * https://github.com/codingABI/DFR0534/tree/main/assets/exampleContent
00046      */
00047     // Play the file "test.wav"
00048     g_audio.playFileByName("/TEST      WAV");
00049 }
00050
00051 void loop() {
00052     static unsigned long lastDisplayMS = millis()-500;
00053     char name[12];
00054
00055     // Show information about current track once per second
00056     if (millis()-lastDisplayMS > 1000) {
00057         Serial.print("number: ");
00058         word fileNumber = g_audio.getFileNumber();
00059         if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");
00060
00061         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;

```

```

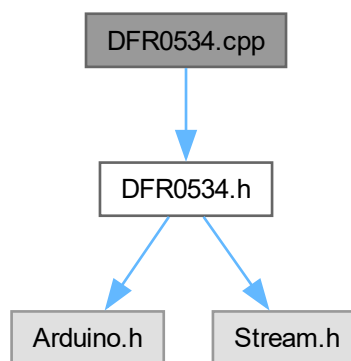
00064   char name[12];
00065
00066   // Show information about current track once per second
00067   if (millis()-lastDisplayMS > 1000) {
00068       Serial.print("number: ");
00069       word fileNumber = g_audio.getFileNumber();
00070       if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");
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

1.0.2

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
00251 void DFR0534::playFileByName(char *path, byte drive)
00252 {
00253     if (m_ptrStream == NULL) return; // Should not happen
00254     if (path == NULL) return;
00255     if (drive >= DRIVEUNKNOWN) return;
00256     sendStartingCode();
00257     sendDataByte(0x08);
00258     sendDataByte(strlen(path)+1);
00259     sendDataByte(drive);
00260     for (int i=0; i<strlen(path); i++) {
00261         sendDataByte(path[i]);
00262     }
00263     sendChecksum();
00264 }
00265
00277 byte DFR0534::getDrivesStates()
00278 {
00279     #define COMMAND 0x09
00280     #define RECEIVEBYTETIMEOUTMS 100
00281     #define RECEIVEGLOBALTIMEOUTMS 500
00282     #define RECEIVEFAILED DRIVEUNKNOWN
00283     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00284
00285     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00286     sendStartingCode();
00287     sendDataByte(COMMAND);
00288     sendDataByte(0x00);
00289     sendChecksum();
00290
00291     // Receive
00292     int i=0;
00293     byte data, firstByte = 0, sum, length=0xff, result = 0;
00294     unsigned long receiveStartMS = millis();
00295     do {
00296         byte dataReady = 0;
00297         unsigned long lastMS = millis();
00298         // Wait for response or timeout
00299         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00300
00301         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00302         data = m_ptrStream->read();
00303
00304         if (i==0) { // Begin of transmission
00305             firstByte=data;
00306             sum = 0;

```

```

00307     }
00308     if ((i == 1) && (data != COMMAND)) {
00309         // Invalid signal => reset receive
00310         i=0;
00311         firstByte = 0;
00312     }
00313     if (i == RECEIVEHEADERLENGTH) {
00314         length = data; // Length of receiving data
00315         if (length != 1) {
00316             // Invalid length => reset receive
00317             i=0;
00318             firstByte = 0;
00319         }
00320     }
00321     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00322         result = data;
00323     }
00324     if (firstByte == STARTINGCODE) {
00325         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00326         i++;
00327     }
00328     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00329 } while (i<length+RECEIVEHEADERLENGTH+2);
00330
00331 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00332 return result;
00333 }
00334
00344 byte DFR0534::getDrive()
00345 {
00346     #define COMMAND 0x0A
00347     #define RECEIVEBYTETIMEOUTMS 100
00348     #define RECEIVEGLOBALTIMEOUTMS 500
00349     #define RECEIVEFAILED DRIVEUNKNOWN
00350     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00351
00352     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00353     sendStartingCode();
00354     sendDataByte(COMMAND);
00355     sendDataByte(0x00);
00356     sendChecksum();
00357
00358     // Receive
00359     int i=0;
00360     byte data, firstByte = 0, sum, length=0xff, result = 0;
00361     unsigned long receiveStartMS = millis();
00362     do {
00363         byte dataReady = 0;
00364         unsigned long lastMS = millis();
00365         // Wait for response or timeout
00366         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00367
00368         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00369         data = m_ptrStream->read();
00370
00371         if (i==0) { // Begin of transmission
00372             firstByte=data;
00373             sum = 0;
00374         }
00375         if ((i == 1) && (data != COMMAND)) {
00376             // Invalid signal => reset receive
00377             i=0;
00378             firstByte = 0;
00379         }
00380         if (i == RECEIVEHEADERLENGTH) {
00381             length = data; // Length of receiving data
00382             if (length != 1) {
00383                 // Invalid length => reset receive
00384                 i=0;
00385                 firstByte = 0;
00386             }
00387         }
00388         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00389             result = data;
00390         }
00391         if (firstByte == STARTINGCODE) {
00392             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00393             i++;
00394         }
00395         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00396     } while (i<length+RECEIVEHEADERLENGTH+2);
00397
00398     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00399     return result;
00400 }
00401

```

```

00407 void DFR0534::setDrive(byte drive)
00408 {
00409     if (m_ptrStream == NULL) return; // Should not happen
00410     if (drive >= DRIVEUNKNOWN) return;
00411     sendStartingCode();
00412     sendDataByte(0x0B);
00413     sendDataByte(0x01);
00414     sendDataByte(drive);
00415     sendChecksum();
00416 }
00417
00426 word DFR0534::getFileNumber()
00427 {
00428     #define COMMAND 0x0D
00429     #define RECEIVEFAILED 0
00430     #define RECEIVEBYTETIMEOUTMS 100
00431     #define RECEIVEGLOBALTIMEOUTMS 500
00432     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00433
00434     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00435     sendStartingCode();
00436     sendDataByte(COMMAND);
00437     sendDataByte(0x00);
00438     sendChecksum();
00439
00440     // Receive
00441     int i=0;
00442     byte data, firstByte = 0, sum, length=0xff;
00443     word result = 0;
00444     unsigned long receiveStartMS = millis();
00445     do {
00446         byte dataReady = 0;
00447         unsigned long lastMS = millis();
00448         // Wait for response or timeout
00449         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00450
00451         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00452         data = m_ptrStream->read();
00453
00454         if (i==0) { // Begin of transmission
00455             firstByte=data;
00456             sum = 0;
00457         }
00458         if ((i == 1) && (data != COMMAND)) {
00459             // Invalid signal => reset receive
00460             i=0;
00461             firstByte = 0;
00462         }
00463         if (i == RECEIVEHEADERLENGTH) {
00464             length = data; // Length of receiving data
00465             if (length != 2) {
00466                 // Invalid length => reset receive
00467                 i=0;
00468                 firstByte = 0;
00469             }
00470         }
00471         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00472             switch (i-RECEIVEHEADERLENGTH-1) {
00473                 case 0:
00474                     result=data<<8;
00475                     break;
00476                 case 1:
00477                     result+=data;
00478                     break;
00479             }
00480         }
00481         if (firstByte == STARTINGCODE) {
00482             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00483             i++;
00484         }
00485         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00486     } while (i<length+RECEIVEHEADERLENGTH+2);
00487
00488     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00489     return result;
00490 }
00491
00498 int DFR0534::getTotalFiles()
00499 {
00500     #define COMMAND 0x0C
00501     #define RECEIVEFAILED -1
00502     #define RECEIVEBYTETIMEOUTMS 100
00503     #define RECEIVEGLOBALTIMEOUTMS 500
00504     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00505
00506     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen

```

```

00507     sendStartingCode();
00508     sendDataByte(COMMAND);
00509     sendDataByte(0x00);
00510     sendChecksum();
00511
00512     // Receive
00513     int i=0;
00514     byte data, firstByte = 0, sum, length=0xff;
00515     word result = 0;
00516     unsigned long receiveStartMS = millis();
00517     do {
00518         byte dataReady = 0;
00519         unsigned long lastMS = millis();
00520         // Wait for response or timeout
00521         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00522
00523         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00524         data = m_ptrStream->read();
00525
00526         if (i==0) { // Begin of transmission
00527             firstByte=data;
00528             sum = 0;
00529         }
00530         if ((i == 1) && (data != COMMAND)) {
00531             // Invalid signal => reset receive
00532             i=0;
00533             firstByte = 0;
00534         }
00535         if (i == RECEIVEHEADERLENGTH) {
00536             length = data; // Length of receiving data
00537             if (length != 2) {
00538                 // Invalid length => reset receive
00539                 i=0;
00540                 firstByte = 0;
00541             }
00542         }
00543         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00544             switch (i-RECEIVEHEADERLENGTH-1) {
00545                 case 0:
00546                     result=data<<8;
00547                     break;
00548                 case 1:
00549                     result+=data;
00550                     break;
00551             }
00552         }
00553         if (firstByte == STARTINGCODE) {
00554             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00555             i++;
00556         }
00557         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00558     } while (i<length+RECEIVEHEADERLENGTH+2);
00559
00560     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00561     return result;
00562 }
00563
00564 void DFR0534::playLastInDirectory()
00565 {
00566     if (m_ptrStream == NULL) return; // Should not happen
00567     sendStartingCode();
00568     sendDataByte(0x0E);
00569     sendDataByte(0x00);
00570     sendChecksum();
00571 }
00572
00573 void DFR0534::playNextDirectory()
00574 {
00575     if (m_ptrStream == NULL) return; // Should not happen
00576     sendStartingCode();
00577     sendDataByte(0x0F);
00578     sendDataByte(0x00);
00579     sendChecksum();
00580 }
00581
00582 int DFR0534::getFirstFileNumberInCurrentDirectory()
00583 {
00584     #define COMMAND 0x11
00585     #define RECEIVEFAILED -1
00586     #define RECEIVEBYTETIMEOUTMS 100
00587     #define RECEIVEGLOBALTIMEOUTMS 500
00588     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00589
00590     if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
00591     sendStartingCode();
00592     sendDataByte(COMMAND);

```



```

00605     sendDataByte(0x00);
00606     sendChecksum();
00607
00608     // Receive
00609     int i=0;
00610     byte data, firstByte = 0, sum, length=0xff;
00611     word result = 0;
00612     unsigned long receiveStartMS = millis();
00613     do {
00614         byte dataReady = 0;
00615         unsigned long lastMS = millis();
00616         // Wait for response or timeout
00617         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00618
00619         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00620         data = m_ptrStream->read();
00621
00622         if (i==0) { // Begin of transmission
00623             firstByte=data;
00624             sum = 0;
00625         }
00626         if ((i == 1) && (data != COMMAND)) {
00627             // Invalid signal => reset receive
00628             i=0;
00629             firstByte = 0;
00630         }
00631         if (i == RECEIVEHEADERLENGTH) {
00632             length = data; // Length of receiving data
00633             if (length != 2) {
00634                 // Invalid length => reset receive
00635                 i=0;
00636                 firstByte = 0;
00637             }
00638         }
00639         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00640             switch (i-RECEIVEHEADERLENGTH-1) {
00641                 case 0:
00642                     result=data<<8;
00643                     break;
00644                 case 1:
00645                     result+=data;
00646                     break;
00647             }
00648         }
00649         if (firstByte == STARTINGCODE) {
00650             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00651             i++;
00652         }
00653         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00654     } while (i<length+RECEIVEHEADERLENGTH+2);
00655
00656     if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00657     return result;
00658 }
00659
00666 int DFR0534::getTotalFilesInCurrentDirectory()
00667 {
00668     #define COMMAND 0x12
00669     #define RECEIVEFAILED -1
00670     #define RECEIVEBYTETIMEOUTMS 100
00671     #define RECEIVEGLOBALTIMEOUTMS 500
00672     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00673
00674     if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
00675     sendStartingCode();
00676     sendDataByte(COMMAND);
00677     sendDataByte(0x00);
00678     sendChecksum();
00679
00680     // Receive
00681     int i=0;
00682     byte data, firstByte = 0, sum, length=0xff;
00683     word result = 0;
00684     unsigned long receiveStartMS = millis();
00685     do {
00686         byte dataReady = 0;
00687         unsigned long lastMS = millis();
00688         // Wait for response or timeout
00689         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
00690
00691         if (dataReady == 0) return RECEIVEFAILED; // Timeout
00692         data = m_ptrStream->read();
00693
00694         if (i==0) { // Begin of transmission
00695             firstByte=data;

```

```

00696     sum = 0;
00697 }
00698 if ((i == 1) && (data != COMMAND)) {
00699     // Invalid signal => reset receive
00700     i=0;
00701     firstByte = 0;
00702 }
00703 if (i == RECEIVEHEADERLENGTH) {
00704     length = data; // Length of receiving data
00705     if (length != 2) {
00706         // Invalid length => reset receive
00707         i=0;
00708         firstByte = 0;
00709     }
00710 }
00711 if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00712     switch (i-RECEIVEHEADERLENGTH-1) {
00713         case 0:
00714             result=data<<8;
00715             break;
00716         case 1:
00717             result+=data;
00718             break;
00719     }
00720 }
00721 if (firstByte == STARTINGCODE) {
00722     if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00723     i++;
00724 }
00725 if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00726 } while (i<length+RECEIVEHEADERLENGTH+2);
00727
00728 if (data != sum) return RECEIVEFAILED; // Does checksum matches?
00729 return result;
00730 }
00731
00735 void DFR0534::increaseVolume()
00736 {
00737     if (m_ptrStream == NULL) return; // Should not happen
00738     sendStartingCode();
00739     sendDataByte(0x14);
00740     sendDataByte(0x00);
00741     sendChecksum();
00742 }
00743
00747 void DFR0534::decreaseVolume()
00748 {
00749     if (m_ptrStream == NULL) return; // Should not happen
00750     sendStartingCode();
00751     sendDataByte(0x15);
00752     sendDataByte(0x00);
00753     sendChecksum();
00754 }
00755
00764 void DFR0534::insertFileByNumber(word track, byte drive)
00765 {
00766     if (m_ptrStream == NULL) return; // Should not happen
00767     if (drive >= DRIVEUNKNOWN) return;
00768     sendStartingCode();
00769     sendDataByte(0x16);
00770     sendDataByte(0x03);
00771     sendDataByte(drive);
00772     sendDataByte((track >> 8) & 0xff);
00773     sendDataByte(track & 0xff);
00774     sendChecksum();
00775 }
00776
00782 void DFR0534::stopInsertedFile()
00783 {
00784     if (m_ptrStream == NULL) return; // Should not happen
00785     sendStartingCode();
00786     sendDataByte(0x10);
00787     sendDataByte(0x00);
00788     sendChecksum();
00789 }
00790
00797 void DFR0534::setDirectory(char *path, byte drive)
00798 {
00799     if (m_ptrStream == NULL) return; // Should not happen
00800     if (path == NULL) return;
00801     if (drive >= DRIVEUNKNOWN) return;
00802     sendStartingCode();
00803     sendDataByte(0x17);
00804     sendDataByte(strlen(path)+1);
00805     sendDataByte(drive);
00806     for (int i=0;i<strlen(path);i++) {
00807         sendDataByte(path[i]);

```

```

00808     }
00809     sendChecksum();
00810 }
00811
00817 void DFR0534::setLoopMode(byte mode)
00818 {
00819     if (m_ptrStream == NULL) return; // Should not happen
00820     if (mode >= PLAYMODEUNKNOWN) return;
00821     sendStartingCode();
00822     sendDataByte(0x18);
00823     sendDataByte(0x01);
00824     sendDataByte(mode);
00825     sendChecksum();
00826 }
00827
00835 void DFR0534::setRepeatLoops(word loops)
00836 {
00837     if (m_ptrStream == NULL) return; // Should not happen
00838     sendStartingCode();
00839     sendDataByte(0x19);
00840     sendDataByte(0x02);
00841     sendDataByte((loops >> 8) & 0xff);
00842     sendDataByte(loops & 0xff);
00843     sendChecksum();
00844 }
00845
00857 void DFR0534::playCombined(char* list)
00858 {
00859     if (m_ptrStream == NULL) return; // Should not happen
00860     if (list == NULL) return;
00861     if ((strlen(list) % 2) != 0) return;
00862     sendStartingCode();
00863     sendDataByte(0x1B);
00864     sendDataByte(strlen(list));
00865     for (int i=0; i<strlen(list); i++) {
00866         sendDataByte(list[i]);
00867     }
00868     sendChecksum();
00869 }
00870
00871
00875 void DFR0534::stopCombined()
00876 {
00877     if (m_ptrStream == NULL) return; // Should not happen
00878     sendStartingCode();
00879     sendDataByte(0x1C);
00880     sendDataByte(0x00);
00881     sendChecksum();
00882 }
00883
00892 void DFR0534::setChannel(byte channel)
00893 {
00894     if (m_ptrStream == NULL) return; // Should not happen
00895     if (channel >= CHANNELUNKNOWN) return;
00896     sendStartingCode();
00897     sendDataByte(0x1D);
00898     sendDataByte(0x01);
00899     sendDataByte(channel);
00900     sendChecksum();
00901 }
00902
00912 bool DFR0534::getFileName(char *name)
00913 {
00914     #define COMMAND 0x1E
00915     #define RECEIVEBYTETIMEOUTMS 100
00916     #define RECEIVEGLOBALTIMEOUTMS 500
00917     #define RECEIVEFAILED false
00918     #define RECEIVEHEADERLENGTH 2 // startingcode+command
00919
00920     if (m_ptrStream == NULL) return false; // Should not happen
00921     if (name == NULL) return false;
00922     name[0] = '\0';
00923
00924     sendStartingCode();
00925     sendDataByte(COMMAND);
00926     sendDataByte(0x00);
00927     sendChecksum();
00928
00929     // Receive
00930     int i=0;
00931     byte data, firstByte = 0, sum, length=0xff;
00932     unsigned long receiveStartMS = millis();
00933     do {
00934         byte dataReady = 0;
00935         unsigned long lastMS = millis();
00936         // Wait for response or timeout
00937         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =

```

```

    m_ptrStream->available();
00938
00939     if (dataReady == 0) return RECEIVEFAILED; // Timeout
00940     data = m_ptrStream->read();
00941     if (i==0) { // Begin of transmission
00942         firstByte=data;
00943         sum = 0;
00944     }
00945     if ((i == 1) && (data != COMMAND)) {
00946         // Invalid signal => reset receive
00947         i=0;
00948         firstByte = 0;
00949     }
00950     if (i == RECEIVEHEADERLENGTH) length = data; // Length of receiving string
00951     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
00952         if ((i-RECEIVEHEADERLENGTH) < 12) { // I expect no longer file names than 8+3 chars plus '\0'
00953             name[i-RECEIVEHEADERLENGTH-1] = data;
00954             name[i-RECEIVEHEADERLENGTH] = '\0';
00955         }
00956     }
00957     if (firstByte == STARTINGCODE) {
00958         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
00959         i++;
00960     }
00961     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
00962     } while (i<length+RECEIVEHEADERLENGTH+2);
00963     return (data == sum); // Does checksum matches?
00964 }
00965
00971 void DFR0534::prepareFileByNumber(word track)
00972 {
00973     if (m_ptrStream == NULL) return; // Should not happen
00974     sendStartingCode();
00975     sendDataByte(0x1F);
00976     sendDataByte(0x02);
00977     sendDataByte((track » 8) & 0xff);
00978     sendDataByte(track & 0xff);
00979     sendChecksum();
00980 }
00981
00992 void DFR0534::repeatPart(byte startMinute, byte startSecond, byte stopMinute, byte stopSecond )
00993 {
00994     if (m_ptrStream == NULL) return; // Should not happen
00995     sendStartingCode();
00996     sendDataByte(0x20);
00997     sendDataByte(0x04);
00998     sendDataByte(startMinute);
00999     sendDataByte(startSecond);
01000     sendDataByte(stopMinute);
01001     sendDataByte(stopSecond);
01002     sendChecksum();
01003 }
01004
01008 void DFR0534::stopRepeatPart()
01009 {
01010     if (m_ptrStream == NULL) return; // Should not happen
01011     sendStartingCode();
01012     sendDataByte(0x21);
01013     sendDataByte(0x00);
01014     sendChecksum();
01015 }
01016
01024 void DFR0534::fastBackwardDuration(word seconds)
01025 {
01026     if (m_ptrStream == NULL) return; // Should not happen
01027     sendStartingCode();
01028     sendDataByte(0x22);
01029     sendDataByte(0x02);
01030     sendDataByte((seconds » 8) & 0xff);
01031     sendDataByte(seconds & 0xff);
01032     sendChecksum();
01033 }
01034
01041 void DFR0534::fastForwardDuration(word seconds)
01042 {
01043     if (m_ptrStream == NULL) return; // Should not happen
01044     sendStartingCode();
01045     sendDataByte(0x23);
01046     sendDataByte(0x02);
01047     sendDataByte((seconds » 8) & 0xff);
01048     sendDataByte(seconds & 0xff);
01049     sendChecksum();
01050 }
01051
01064 bool DFR0534::getDuration(byte &hour, byte &minute, byte &second)
01065 {
01066     #define COMMAND 0x24

```

```

01067 #define RECEIVEFAILED false
01068 #define RECEIVEBYTETIMEOUTMS 100
01069 #define RECEIVEGLOBALTIMEOUTMS 500
01070 #define RECEIVEHEADERLENGTH 2 // startingcode+command
01071
01072 if (m_ptrStream == NULL) return false; // Should not happen
01073 sendStartingCode();
01074 sendDataByte(COMMAND);
01075 sendDataByte(0x00);
01076 sendChecksum();
01077
01078 // Receive
01079 int i=0;
01080 byte data, firstByte = 0, sum, length=0xff;
01081 word result = 0;
01082 unsigned long receiveStartMS = millis();
01083 do {
01084     byte dataReady = 0;
01085     unsigned long lastMS = millis();
01086     // Wait for response or timeout
01087     while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01088
01089     if (dataReady == 0) return RECEIVEFAILED; // Timeout
01090     data = m_ptrStream->read();
01091
01092     if (i==0) { // Begin of transmission
01093         firstByte=data;
01094         sum = 0;
01095     }
01096     if ((i == 1) && (data != COMMAND)) {
01097         // Invalid signal => reset receive
01098         i=0;
01099         firstByte = 0;
01100     }
01101     if (i == RECEIVEHEADERLENGTH) {
01102         length = data; // Length of receiving data
01103         if (length != 3) {
01104             // Invalid length => reset receive
01105             i=0;
01106             firstByte = 0;
01107         }
01108     }
01109     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01110         switch (i-RECEIVEHEADERLENGTH-1) {
01111             case 0:
01112                 hour=data;
01113                 break;
01114             case 1:
01115                 minute=data;
01116                 break;
01117             case 2:
01118                 second=data;
01119                 break;
01120         }
01121     }
01122     if (firstByte == STARTINGCODE) {
01123         if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01124         i++;
01125     }
01126     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01127 } while (i<length+RECEIVEHEADERLENGTH+2);
01128
01129 return (data == sum); // Does checksum matches?
01130 }
01131
01135 void DFR0534::startSendingRuntime()
01136 {
01137     if (m_ptrStream == NULL) return; // Should not happen
01138     sendStartingCode();
01139     sendDataByte(0x25);
01140     sendDataByte(0x00);
01141     sendChecksum();
01142 }
01143
01157 bool DFR0534::getRuntime(byte &hour, byte &minute, byte &second)
01158 {
01159     #define COMMAND 0x25
01160     #define RECEIVEFAILED false
01161     #define RECEIVEBYTETIMEOUTMS 100
01162     #define RECEIVEGLOBALTIMEOUTMS 500
01163     #define RECEIVEHEADERLENGTH 2 // startingcode+command
01164
01165     if (m_ptrStream == NULL) return false; // Should not happen
01166
01167     // Receive
01168     int i=0;

```

```

01169     byte data, firstByte = 0, sum, length=0xff;
01170     word result = 0;
01171     unsigned long receiveStartMS = millis();
01172     do {
01173         byte dataReady = 0;
01174         unsigned long lastMS = millis();
01175         // Wait for response or timeout
01176         while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();
01177
01178         if (dataReady == 0) return RECEIVEFAILED; // Timeout
01179         data = m_ptrStream->read();
01180
01181         if (i==0) { // Begin of transmission
01182             firstByte=data;
01183             sum = 0;
01184         }
01185         if ((i == 1) && (data != COMMAND)) {
01186             // Invalid signal => reset receive
01187             i=0;
01188             firstByte = 0;
01189         }
01190         if (i == RECEIVEHEADERLENGTH) {
01191             length = data; // Length of receiving data
01192             if (length != 3) {
01193                 // Invalid length => reset receive
01194                 i=0;
01195                 firstByte = 0;
01196             }
01197         }
01198         if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01199             switch (i-RECEIVEHEADERLENGTH-1) {
01200                 case 0:
01201                     hour=data;
01202                     break;
01203                 case 1:
01204                     minute=data;
01205                     break;
01206                 case 2:
01207                     second=data;
01208                     break;
01209             }
01210         }
01211         if (firstByte == STARTINGCODE) {
01212             if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01213             i++;
01214         }
01215         if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01216     } while (i<length+RECEIVEHEADERLENGTH+2);
01217
01218     return (data == sum); // Does checksum matches?
01219 }
01220
01224 void DFR0534::stopSendingRuntime()
01225 {
01226     if (m_ptrStream == NULL) return; // Should not happen
01227     sendStartingCode();
01228     sendDataByte(0x26);
01229     sendDataByte(0x00);
01230     sendChecksum();
01231 }

```

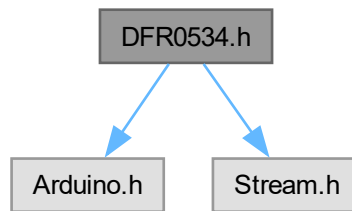
## 5.6 DFR0534.h File Reference

```

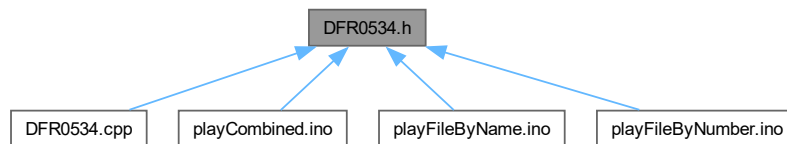
#include <Arduino.h>
#include <Stream.h>

```

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 "1.0.2"`

### 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

1.0.2

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

## 5.6.2 Macro Definition Documentation

### 5.6.2.1 DFR0534\_VERSION

```
#define DFR0534_VERSION "1.0.2"
```

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 "1.0.2"
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, [27](#)  
    playNextDirectory, [27](#)  
    playPrevious, [27](#)  
    PLAYRANDOM, [10](#)  
    prepareFileByNumber, [27](#)  
    RANDOMINDIRECTORY, [10](#)  
    repeatPart, [28](#)  
    SEQUENTIAL, [10](#)  
    SEQUENTIALINDIRECTORY, [10](#)  
    setChannel, [28](#)  
    setDirectory, [29](#)  
    setDrive, [29](#)  
    setEqualizer, [30](#)  
    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](#)  
 getFileNameNumber  
     DFR0534, [17](#)  
 getFirstFileNameInCurrentDirectory  
     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](#)  
 playFileName  
     DFR0534, [25](#)  
 playFileName.ino, [36](#)  
 playFileByNumber  
     DFR0534, [26](#)  
  
 playFileByNumber.ino, [37](#)  
 PLAYING  
     DFR0534, [11](#)  
 playLastInDirectory  
     DFR0534, [26](#)  
 PLAYMODEUNKNOWN  
     DFR0534, [10](#)  
 playNext  
     DFR0534, [27](#)  
 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, [30](#)  
 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](#)