DFR0534

1.0.4

Chapter 1

DFR0534

An Arduino Uno/Nano/ATmega328p, ESP32 library for a `DFR0534` audio module. The library works with SoftwareSerial/HardwareSerial and is very similar to `https://github.com/sleemanj/JQ8400_↩ Serial`, but is no fork.

Examples, how to use the library

- examples/playFileByName/playFileByName.ino

- examples/playFileByNumber/playFileByNumber.ino

- examples/playCombined/playCombined.ino

1.1 SoftwareSerial for Arduino Uno/Nano/ATmega328p

To create a DFR0534 object pass the SoftwareSerial object as a parameter to the DFR0534 constructor, for example
```
#include <SoftwareSerial.h>
#include <DFR0534.h>

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

1.2 HardwareSerial for ESP32

To create a DFR0534 object pass the HardwareSerial object as a parameter to the DFR0534 constructor, for example
```
#include <DFR0534.h>

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

DFR0534 g_audio(g_serial);

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

1.3 License and copyright

This library is licensed under the terms of

BSD 2-Clause License

Copyright (c) 2024, codingABI All rights reserved.

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

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

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

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPE-CIAL, 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 SOFT-WARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.4 Appendix

1.4.1 DFR0534 pinout
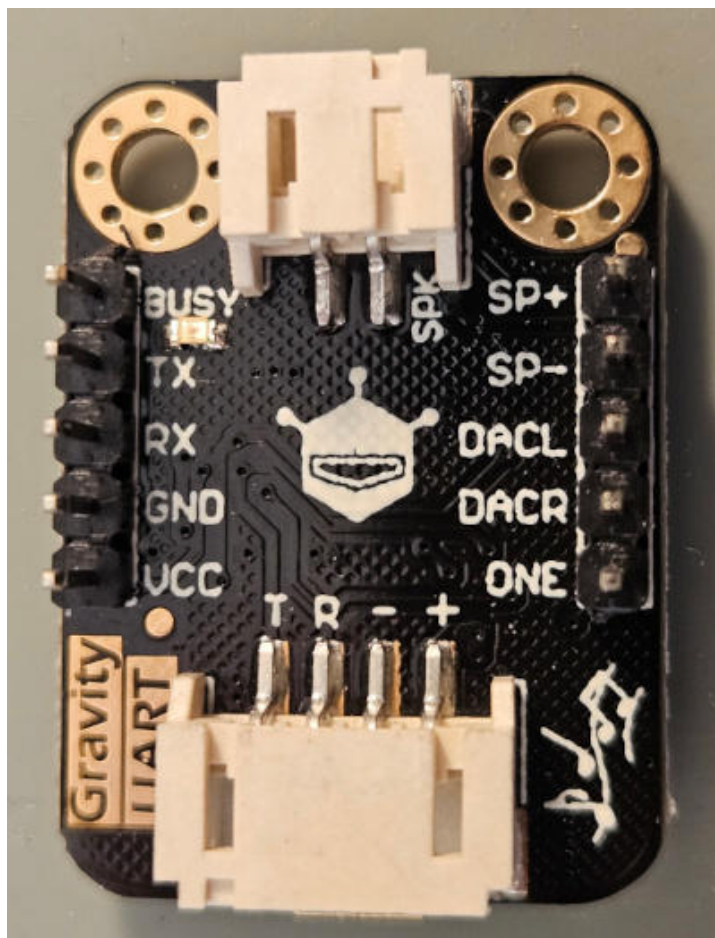
Figure 1.1 DFR0534 frontside
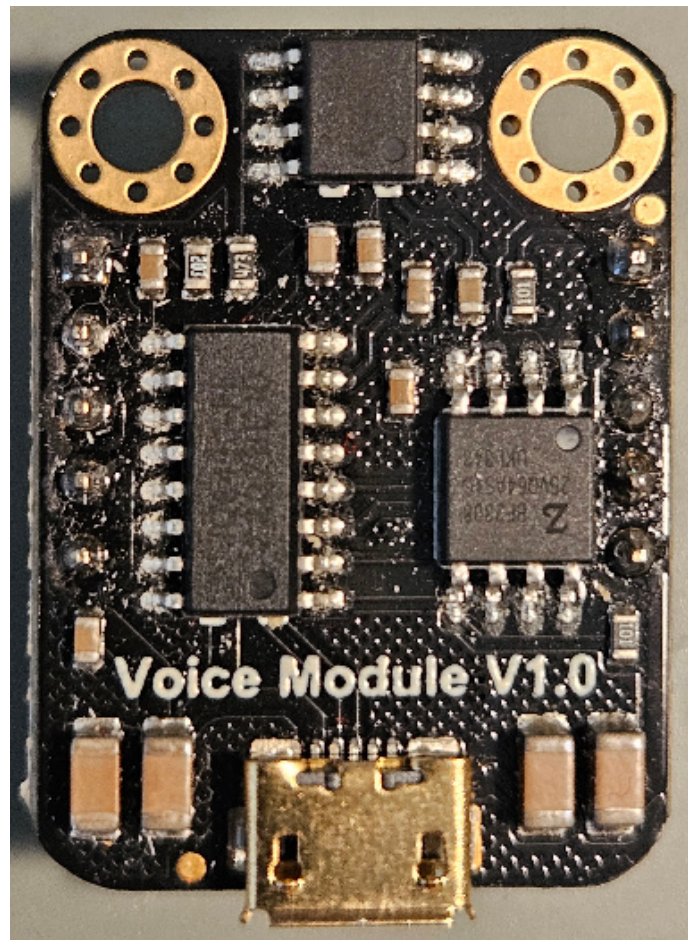
Figure 1.2 DFR0534 backside

Minimal schematic to use this library

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

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

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 in seconds.
- 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	*stream*	Serial connection object, like SoftwareSerial or HardwareSerial

Definition at line 87 of file DFR0534.h.

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

4.1.4 Member Function Documentation

4.1.4.1 decreaseVolume()

void DFR0534::decreaseVolume ()

Decrease volume by one step.

Definition at line 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 in seconds.

Parameters

in	*seconds*	Seconds to go backward

Definition at line 1023 of file DFR0534.cpp.

```
01024 {
01025   if (m_ptrStream == NULL) return; // Should not happen
01026   sendStartingCode();
01027   sendDataByte(0x22);
01028   sendDataByte(0x02);
01029   sendDataByte((seconds » 8) & 0xff);
01030   sendDataByte(seconds & 0xff);
01031   sendCheckSum();
01032 }
```

4.1.4.3 fastForwardDuration()

void DFR0534::fastForwardDuration (
            word *seconds*)

Fast forward in seconds.

Parameters

in	*seconds*	Seconds to go forward

Definition at line 1040 of file DFR0534.cpp.

```
01041 {
01042   if (m_ptrStream == NULL) return; // Should not happen
01043   sendStartingCode();
01044   sendDataByte(0x23);
01045   sendDataByte(0x02);
01046   sendDataByte((seconds » 8) & 0xff);
01047   sendDataByte(seconds & 0xff);
01048   sendCheckSum();
01049 }
```

4.1.4.4 getDrive()

byte DFR0534::getDrive ()

Get current drive.

Return values

DFR0534::DRIVEUSB	USB drive
DFR0534::DRIVESD	SD card
DFR0534::DRIVEFLASH	Flash memory chip
DFR0534::DRIVENO	No drive found
DFR0534::DRIVEUNKNOWN	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

DFR0534::DRIVEUNKNOWN	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	*hour*	Hours
out	*minute*	Minutes
out	*second*	Seconds

Return values

true	Request was successful
false	Request failed

Definition at line 1063 of file DFR0534.cpp.

```
01064 {
01065    #define COMMAND 0x24
01066    #define RECEIVEFAILED false
01067    #define RECEIVEBYTETIMEOUTMS 100
01068    #define RECEIVEGLOBALTIMEOUTMS 500
01069    #define RECEIVEHEADERLENGTH 2 // startingcode+command
01070
01071    if (m_ptrStream == NULL) return false; // Should not happen
01072    sendStartingCode();
01073    sendDataByte(COMMAND);
01074    sendDataByte(0x00);
01075    sendCheckSum();
01076
01077    // Receive
01078    int i=0;
01079    byte data, firstByte = 0, sum, length=0xff;
01080    word result = 0;
01081    unsigned long receiveStartMS = millis();
01082    do {
01083      byte dataReady = 0;
01084      unsigned long lastMS = millis();
01085      // Wait for response or timeout
01086      while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
    m_ptrStream->available();
01087
01088      if (dataReady == 0) return RECEIVEFAILED; // Timeout
01089      data = m_ptrStream->read();
01090
01091      if (i==0) { // Begin of transmission
01092        firstByte=data;
01093        sum = 0;
01094      }
01095      if ((i == 1) && (data != COMMAND)) {
01096        // Invalid signal => reset receive
01097        i=0;
01098        firstByte = 0;
01099      }
01100      if (i == RECEIVEHEADERLENGTH) {
01101        length = data; // Length of receiving data
01102        if (length != 3) {
01103          // Invalid length => reset receive
01104          i=0;
01105          firstByte = 0;
01106        }
01107      }
01108      if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01109        switch (i-RECEIVEHEADERLENGTH-1) {
01110          case 0:
```

```
01111            hour=data;
01112            break;
01113          case 1:
01114            minute=data;
01115            break;
01116          case 2:
01117            second=data;
01118            break;
01119        }
01120      }
01121      if (firstByte == STARTINGCODE) {
01122        if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01123        i++;
01124      }
01125      if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01126    } while (i<length+RECEIVEHEADERLENGTH+2);
01127
01128    return (data == sum); // Does checksum matches?
01129 }
```

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	*name*	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	*hour*	Hours
out	*minute*	Minutes
out	*second*	Seconds

Return values

true	Request was successful
false	Request failed

Definition at line 1156 of file DFR0534.cpp.

```
01157 {
01158   #define COMMAND 0x25
01159   #define RECEIVEFAILED false
01160   #define RECEIVEBYTETIMEOUTMS 100
01161   #define RECEIVEGLOBALTIMEOUTMS 500
01162   #define RECEIVEHEADERLENGTH 2 // startingcode+command
01163
01164   if (m_ptrStream == NULL) return false; // Should not happen
01165
01166   // Receive
01167   int i=0;
01168   byte data, firstByte = 0, sum, length=0xff;
01169   word result = 0;
01170   unsigned long receiveStartMS = millis();
01171   do {
```

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

4.1.4.11 getStatus()

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

Get module status.

Return values

DFR0534::STOPPED	Audio module is idle
DFR0534::PLAYING	Audio module is playing a file
DFR0534::PAUSED	Audio module is paused
DFR0534::STATUSUNKNOWN	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	*track*	File number of the audio file
in	*drive*	Drive, where file is stored: Drive DFR0534::DRIVEUSB, DFR0534::DRIVESD or DFR0534::DRIVEFLASH (=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	*list*	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 a dot between file name and file extension

- All characters in upper case

- Maximal 8 characters for file name

- Every file and folder whose name length is shorter then 8 chars must be filled up to the 8 chars length by space chars

- Must end with WAV or MP3

- Only WAV and MP3 files are supported

- Wildcards ∗ (=multiple arbitrary characters) and ? (=one single arbitrary character) are allowed and can be used to reduce the filling space chars

Valid examples (first means first in "file copy order"):

- "/01 WAV" for a file '/01.wav'

- "/99-AFR∼1MP3" for a file '/99-Africa.mp3'

- "/SUN∗MP3" for first file matching '/sun∗.mp3', for example '/sun.mp3'

- "/99-AFR∗MP3" for first file matching '/99-Afr∗.mp3'

- "/10∗" for first audio file matching '/10∗.∗'

- "/10 /20 WAV" for the file '/10/20.wav'

Parameters

in	*path*	Full path of the audio file
in	*drive*	Drive, where file is stored: Drive DFR0534::DRIVEUSB, DFR0534::DRIVESD or DFR0534::DRIVEFLASH (=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	*track*	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	*track*	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	*startMinute*	Minute for start position
in	*startSecond*	Second for start position
in	*stopMinute*	Minute for stop position
in	*stopSecond*	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	*channel*	Output channel: DFR0534::CHANNELMP3, DFR0534::CHANNELAUX or DFR0534::CHANNELMP3AUX

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	*path*	Directory
in	*drive*	Drive, where directory is stored: Drive DFR0534::DRIVEUSB, DFR0534::DRIVESD or DFR0534::DRIVEFLASH (=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	*drive*	Drive DFR0534::DRIVEUSB, DFR0534::DRIVESD or DFR0534::DRIVEFLASH

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: DFR0534::NORMAL, DFR0534::POP, DFR0534::ROCK, DFR0534::JAZZ or DFR0534::CLASSIC

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: DFR0534::LOOPBACKALL, DFR0534::SINGLEAUDIOLOOP, DFR0534::SINGLEAUDIOSTOP, DFR0534::PLAYRANDOM, DFR0534::DIRECTORYLOOP, DFR0534::RANDOMINDIRECTORY, DFR0534::SEQUENTIALINDIRECTORY or DFR0534::SEQUENTIAL

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	*loops*	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	*volume*	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 1134 of file DFR0534.cpp.

```
01135 {
01136   if (m_ptrStream == NULL) return; // Should not happen
01137   sendStartingCode();
01138   sendDataByte(0x25);
01139   sendDataByte(0x00);
01140   sendCheckSum();
01141 }
```

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 1223 of file DFR0534.cpp.

```
01224 {
01225   if (m_ptrStream == NULL) return; // Should not happen
01226   sendStartingCode();
01227   sendDataByte(0x26);
01228   sendDataByte(0x00);
01229   sendCheckSum();
01230 }
```

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

- DFR0534.h
- DFR0534.cpp

Chapter 5

File Documentation

5.1 playCombined.ino

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

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

5.2 playFileByName.ino

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

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

5.3 playFileByNumber.ino

```
00001 /*
00002  * Example for using the DFR0534 for playing audio files by file number
00003  *
00004  * This example code was made for Arduino Uno/Nano/ATmega328p. For ESP32 you have the change the code
       to use HardwareSerial
00005  * instead of SoftwareSerial (see https://github.com/codingABI/DFR0534#hardwareserial-for-esp32)
00006  */
00007
00008 #include <SoftwareSerial.h>
00009 #include <DFR0534.h>
00010
00011 #define TX_PIN A0
00012 #define RX_PIN A1
00013 SoftwareSerial g_serial(RX_PIN, TX_PIN);
00014 DFR0534 g_audio(g_serial);
00015
00016 void setup() {
00017   // Serial for console output
00018   Serial.begin(9600);
00019   // Software serial for communication to DFR0534 module
00020   g_serial.begin(9600);
00021
00022   // Set volume
00023   g_audio.setVolume(18);
00024
00025   // Show some device infos
00026   Serial.print("Ready drives: ");
00027   byte drive = g_audio.getDrivesStates();
00028   if (((drive » DFR0534::DRIVEUSB) & 1) == 1) Serial.print("USB ");
00029   if (((drive » DFR0534::DRIVESD) & 1) == 1) Serial.print("SD ");
00030   if (((drive » DFR0534::DRIVEFLASH) & 1) == 1) Serial.print("FLASH ");
00031   Serial.println();
00032
00033   Serial.print("Current playing drive: ");
00034   switch(g_audio.getDrive()) {
00035     case DFR0534::DRIVEUSB:
00036       Serial.println("USB");
00037       break;
00038     case DFR0534::DRIVESD:
00039       Serial.println("SD");
00040       break;
00041     case DFR0534::DRIVEFLASH:
00042       Serial.println("FLASH");
00043       break;
00044     case DFR0534::DRIVENO:
00045       Serial.println("No drive");
00046       break;
00047     default:
00048       Serial.println("Unknown");
00049       break;
00050   }
00051
00052   Serial.print("Total files: ");
00053   Serial.println(g_audio.getTotalFiles());
00054   Serial.print("Total files in directory: ");
00055   Serial.println(g_audio.getTotalFilesInCurrentDirectory());
```

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

5.4 DFR0534.cpp File Reference

```
#include "DFR0534.h"
```
Include dependency graph for DFR0534.cpp:
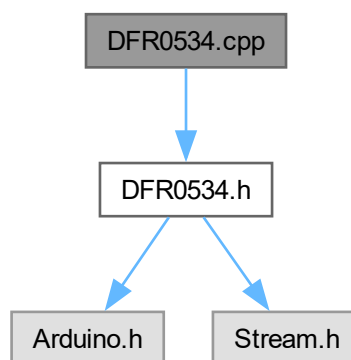
5.4.1 Detailed Description

Class: DFR0534

Description: Class for controlling a DFR0534 audio module (`https://wiki.dfrobot.com/Voice_`↩
`Module_SKU__DFR0534`) by SoftwareSerial or HardwareSerial

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

Notes for DFR0534 audio module:

- Consumes about 20mA when idle (Vcc = 5V)

- Creates a short "click" noise, when Vcc 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 to original audiofile is paused and will be resumed after the "inserted" audiofile

- Can play files in a playlist like mode called "combined" for files stored in a directory /ZH

- Can select the file to play by a file number∗ or file name∗∗ ∗File number is independent from file name. The first WAV or MP3 copied to the DFR0534 gets file number 1 and so on. To play a file by number use playFileByNumber() ∗∗File name is a little bit like a 8+3 file path and can be used with playFileByName(), but have special rules (see playFileByName() for details)

- Can send automatically the file runtime every second (when enabled)

- Has a NS8002 amplifier, JQ8400 Audio chip, W25Q64JVSIQ flash memory

- Has a Sleep mode 0x1B and this mode only works with one-wire protocol (`https://github.`↩
`com/arduino12/mp3_player_module_wire`) and does not work for me without additional electric modifications (e.g. disconnecting speakers) => Switching off DFR0534 with a FET is a better solution

Home: `https://github.com/codingABI/DFR0534`

Author

codingABI `https://github.com/codingABI/`

Copyright

2-Clause BSD License

Version

1.0.4

Definition in file DFR0534.cpp.

5.5 DFR0534.cpp

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

```
00144    sendCheckSum();
00145 }
00146
00154 void DFR0534::setVolume(byte volume)
00155 {
00156    if (m_ptrStream == NULL) return; // Should not happen
00157    if (volume > 30) volume = 30;
00158    sendStartingCode();
00159    sendDataByte(0x13);
00160    sendDataByte(0x01);
00161    sendDataByte(volume);
00162    sendCheckSum();
00163 }
00164
00168 void DFR0534::play()
00169 {
00170    if (m_ptrStream == NULL) return; // Should not happen
00171    sendStartingCode();
00172    sendDataByte(0x02);
00173    sendDataByte(0x00);
00174    sendCheckSum();
00175 }
00176
00180 void DFR0534::pause()
00181 {
00182    if (m_ptrStream == NULL) return; // Should not happen
00183    sendStartingCode();
00184    sendDataByte(0x03);
00185    sendDataByte(0x00);
00186    sendCheckSum();
00187 }
00188
00192 void DFR0534::stop()
00193 {
00194    if (m_ptrStream == NULL) return; // Should not happen
00195    sendStartingCode();
00196    sendDataByte(0x04);
00197    sendDataByte(0x00);
00198    sendCheckSum();
00199 }
00200
00204 void DFR0534::playPrevious()
00205 {
00206    if (m_ptrStream == NULL) return; // Should not happen
00207    sendStartingCode();
00208    sendDataByte(0x05);
00209    sendDataByte(0x00);
00210    sendCheckSum();
00211 }
00212
00216 void DFR0534::playNext()
00217 {
00218    if (m_ptrStream == NULL) return; // Should not happen
00219    sendStartingCode();
00220    sendDataByte(0x06);
00221    sendDataByte(0x00);
00222    sendCheckSum();
00223 }
00224
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
00567 void DFR0534::playLastInDirectory()
00568 {
00569    if (m_ptrStream == NULL) return; // Should not happen
00570    sendStartingCode();
00571    sendDataByte(0x0E);
00572    sendDataByte(0x00);
00573    sendCheckSum();
00574 }
00575
00579 void DFR0534::playNextDirectory()
00580 {
```

```
00581    if (m_ptrStream == NULL) return; // Should not happen
00582    sendStartingCode();
00583    sendDataByte(0x0F);
00584    sendDataByte(0x00);
00585    sendCheckSum();
00586 }
00587
00594 int DFR0534::getFirstFileNumberInCurrentDirectory()
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 }
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
00863    sendStartingCode();
00864    sendDataByte(0x1B);
00865    sendDataByte(strlen(list));
00866    for (int i=0;i<strlen(list);i++) {
00867      sendDataByte(list[i]);
00868    }
00869    sendCheckSum();
00870 }
00871
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
01023 void DFR0534::fastBackwardDuration(word seconds)
01024 {
01025     if (m_ptrStream == NULL) return; // Should not happen
01026     sendStartingCode();
01027     sendDataByte(0x22);
01028     sendDataByte(0x02);
01029     sendDataByte((seconds » 8) & 0xff);
```

```
01030    sendDataByte(seconds & 0xff);
01031    sendCheckSum();
01032  }
01033
01040  void DFR0534::fastForwardDuration(word seconds)
01041  {
01042    if (m_ptrStream == NULL) return; // Should not happen
01043    sendStartingCode();
01044    sendDataByte(0x23);
01045    sendDataByte(0x02);
01046    sendDataByte((seconds » 8) & 0xff);
01047    sendDataByte(seconds & 0xff);
01048    sendCheckSum();
01049  }
01050
01063  bool DFR0534::getDuration(byte &hour, byte &minute, byte &second)
01064  {
01065    #define COMMAND 0x24
01066    #define RECEIVEFAILED false
01067    #define RECEIVEBYTETIMEOUTMS 100
01068    #define RECEIVEGLOBALTIMEOUTMS 500
01069    #define RECEIVEHEADERLENGTH 2 // startingcode+command
01070
01071    if (m_ptrStream == NULL) return false; // Should not happen
01072    sendStartingCode();
01073    sendDataByte(COMMAND);
01074    sendDataByte(0x00);
01075    sendCheckSum();
01076
01077    // Receive
01078    int i=0;
01079    byte data, firstByte = 0, sum, length=0xff;
01080    word result = 0;
01081    unsigned long receiveStartMS = millis();
01082    do {
01083      byte dataReady = 0;
01084      unsigned long lastMS = millis();
01085      // Wait for response or timeout
01086      while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
    m_ptrStream->available();
01087
01088      if (dataReady == 0) return RECEIVEFAILED; // Timeout
01089      data = m_ptrStream->read();
01090
01091      if (i==0) { // Begin of transmission
01092        firstByte=data;
01093        sum = 0;
01094      }
01095      if ((i == 1) && (data != COMMAND)) {
01096        // Invalid signal => reset receive
01097        i=0;
01098        firstByte = 0;
01099      }
01100      if (i == RECEIVEHEADERLENGTH) {
01101        length = data; // Length of receiving data
01102        if (length != 3) {
01103          // Invalid length => reset receive
01104          i=0;
01105          firstByte = 0;
01106        }
01107      }
01108      if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01109        switch (i-RECEIVEHEADERLENGTH-1) {
01110          case 0:
01111            hour=data;
01112            break;
01113          case 1:
01114            minute=data;
01115            break;
01116          case 2:
01117            second=data;
01118            break;
01119        }
01120      }
01121      if (firstByte == STARTINGCODE) {
01122        if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01123        i++;
01124      }
01125      if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01126    } while (i<length+RECEIVEHEADERLENGTH+2);
01127
01128    return (data == sum); // Does checksum matches?
01129  }
01130
01134  void DFR0534::startSendingRuntime()
01135  {
01136    if (m_ptrStream == NULL) return; // Should not happen
```

```
01137   sendStartingCode();
01138   sendDataByte(0x25);
01139   sendDataByte(0x00);
01140   sendCheckSum();
01141 }
01142
01156 bool DFR0534::getRuntime(byte &hour, byte &minute, byte &second)
01157 {
01158   #define COMMAND 0x25
01159   #define RECEIVEFAILED false
01160   #define RECEIVEBYTETIMEOUTMS 100
01161   #define RECEIVEGLOBALTIMEOUTMS 500
01162   #define RECEIVEHEADERLENGTH 2 // startingcode+command
01163
01164   if (m_ptrStream == NULL) return false; // Should not happen
01165
01166   // Receive
01167   int i=0;
01168   byte data, firstByte = 0, sum, length=0xff;
01169   word result = 0;
01170   unsigned long receiveStartMS = millis();
01171   do {
01172     byte dataReady = 0;
01173     unsigned long lastMS = millis();
01174     // Wait for response or timeout
01175     while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
      m_ptrStream->available();
01176
01177     if (dataReady == 0) return RECEIVEFAILED; // Timeout
01178     data = m_ptrStream->read();
01179
01180     if (i==0) { // Begin of transmission
01181       firstByte=data;
01182       sum = 0;
01183     }
01184     if ((i == 1) && (data != COMMAND)) {
01185       // Invalid signal => reset receive
01186       i=0;
01187       firstByte = 0;
01188     }
01189     if (i == RECEIVEHEADERLENGTH) {
01190       length = data; // Length of receiving data
01191       if (length != 3) {
01192         // Invalid length => reset receive
01193         i=0;
01194         firstByte = 0;
01195       }
01196     }
01197     if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
01198       switch (i-RECEIVEHEADERLENGTH-1) {
01199         case 0:
01200           hour=data;
01201           break;
01202         case 1:
01203           minute=data;
01204           break;
01205         case 2:
01206           second=data;
01207           break;
01208       }
01209     }
01210     if (firstByte == STARTINGCODE) {
01211       if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
01212       i++;
01213     }
01214     if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
01215   } while (i<length+RECEIVEHEADERLENGTH+2);
01216
01217   return (data == sum); // Does checksum matches?
01218 }
01219
01223 void DFR0534::stopSendingRuntime()
01224 {
01225   if (m_ptrStream == NULL) return; // Should not happen
01226   sendStartingCode();
01227   sendDataByte(0x26);
01228   sendDataByte(0x00);
01229   sendCheckSum();
01230 }
```

5.6 DFR0534.h File Reference

```
#include <Arduino.h>
#include <Stream.h>
```
Include dependency graph for DFR0534.h:
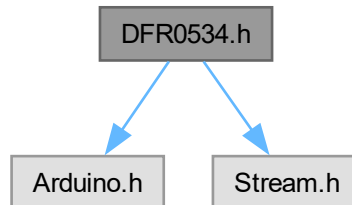
This graph shows which files directly or indirectly include this file:
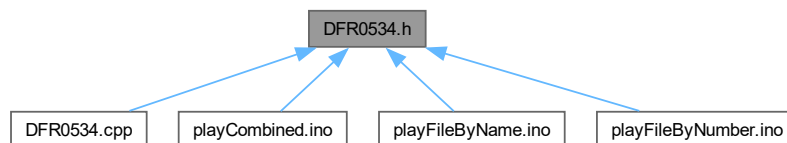
Classes

- class DFR0534

    Class for a DFR0534 audio module.

Macros

- #define DFR0534_VERSION "1.0.4"

5.6.1 Detailed Description

Class: DFR0534

Description: Class for controlling a DFR0534 audio module (`https://wiki.dfrobot.com/Voice_⤶` `Module_SKU__DFR0534`) by SoftwareSerial or HardwareSerial

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

Home: `https://github.com/codingABI/DFR0534`

Author

codingABI https://github.com/codingABI/

Copyright

2-Clause BSD License

Version

1.0.4

Definition in file DFR0534.h.

5.6.2 Macro Definition Documentation

5.6.2.1 DFR0534_VERSION

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

Library version

Definition at line 22 of file DFR0534.h.

5.7 DFR0534.h

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

```
00062        PLAYMODEUNKNOWN
00063      };
00065      enum DFR0534EQ
00066      {
00067        NORMAL,
00068        POP,
00069        ROCK,
00070        JAZZ ,
00071        CLASSIC,
00072        EQUNKNOWN
00073      };
00075      enum DFR0534STATUS
00076      {
00077        STOPPED,
00078        PLAYING,
00079        PAUSED,
00080        STATUSUNKNOWN
00081      };
00087      DFR0534(Stream &stream)
00088      {
00089        m_ptrStream = &stream;
00090      }
00091      void decreaseVolume();
00092      void fastBackwardDuration(word seconds);
00093      void fastForwardDuration(word seconds);
00094      byte getDrive();
00095      byte getDrivesStates();
00096      bool getDuration(byte &hour, byte &minute, byte &second);
00097      bool getFileName(char *name);
00098      word getFileNumber();
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