DFR0534

1.0.1

Generated by Doxygen 1.10.0

Chapter 1

DFR0534

An Arduino Uno/Nano library for a DFR0534 audio module. The library works with SoftwareSerial and is very similar to https://github.com/sleemanj/JQ8400_Serial, but is no fork.

To create a DFR0534 object pass the existing SoftwareSerial object as parameter to the DFR0534 constructor, for example

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

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

Examples how to use the library

- examples/playFileByName/playFileByName.ino
- examples/playFileByNumber/playFileByNumber.ino
- examples/playCombined/playCombined.ino

1.1 License and copyright

This library is licensed under the terms of

BSD 2-Clause License

Copyright (c) 2024, codingABI All rights reserved.

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

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

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

1.2 Appendix

1.2.1 DF0534 pinout

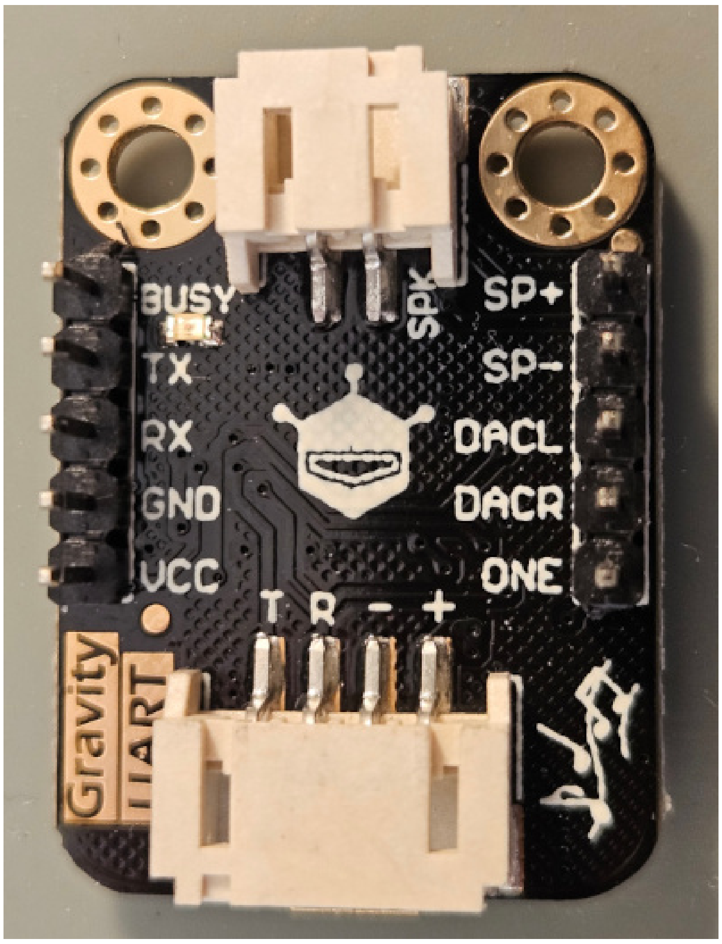


Figure 1.1 DFR0534

Minimal schematic to use this library

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

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

Chapter 2

Class Index

2.1 Class List

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

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.
- 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.

```
    {
      DRIVEUSB,
      DRIVESD,
      DRIVEFLASH,
      DRIVEUNKNOWN,
      DRIVENO = 0xff
    };
```

4.1.2.3 DFR0534EQ

`enum DFR0534::DFR0534EQ`

EQ modes

Enumerator

NORMAL	(=default after device startup)

Definition at line 65 of file DFR0534.h.

```
    {
      NORMAL,
      POP,
      ROCK,
      JAZZ ,
      CLASSIC,
      EQUNKNOWN
    };
```

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.

```
    {
      LOOPBACKALL,
      SINGLEAUDIOLOOP,
      SINGLEAUDIOSTOP,
      PLAYRANDOM,
      DIRECTORYLOOP,
      RANDOMINDIRECTORY,
      SEQUENTIALINDIRECTORY,
      SEQUENTIAL,
      PLAYMODEUNKNOWN
    };
```

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.

```
    {
      STOPPED,
      PLAYING,
      PAUSED,
      STATUSUNKNOWN
    };
```

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

Definition at line 87 of file DFR0534.h.

```
    {
      m_ptrStream = &stream;
    }
```

4.1.4 Member Function Documentation

4.1.4.1 decreaseVolume()

`void DFR0534::decreaseVolume ( )`

Decrease volume by one step.

Definition at line 742 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x15);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.2 fastBackwardDuration()

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

Fast backward.

Fast backward in seconds

Parameters

in	*seconds*	Seconds to go backward

Definition at line 1019 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x22);
  sendDataByte(0x02);
  sendDataByte((seconds » 8) & 0xff);
  sendDataByte(seconds & 0xff);
  sendCheckSum();
}
```

4.1.4.3 fastForwardDuration()

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

Fast forward in seconds.

Parameters

in	*seconds*	Seconds to go forward

Definition at line 1036 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x23);
  sendDataByte(0x02);
  sendDataByte((seconds » 8) & 0xff);
  sendDataByte(seconds & 0xff);
  sendCheckSum();
}
```

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 339 of file DFR0534.cpp.

```
{
  #define COMMAND 0x0A
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEFAILED DRIVEUNKNOWN
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff, result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 1) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      result = data;
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}
```

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 272 of file DFR0534.cpp.

```
{
  #define COMMAND 0x09
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEFAILED DRIVEUNKNOWN
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff, result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 1) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      result = data;
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}
```

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 1059 of file DFR0534.cpp.

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

return (data == sum); // Does checksum matches?
}
```

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 907 of file DFR0534.cpp.

```
{
  #define COMMAND 0x1E
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEFAILED false
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return false; // Should not happen
  if (name == NULL) return false;
  name[0] = '\0';

  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();
    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) length = data; // Length of receiving string
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      if ((i-RECEIVEHEADERLENGTH) < 12) { // I expect no longer file names than 8+3 chars plus '\0'
        name[i-RECEIVEHEADERLENGTH-1] = data;
        name[i-RECEIVEHEADERLENGTH] = '\0';
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);
  return (data == sum); // Does checksum matches?
}
```

4.1.4.8 getFileNumber()

```
word DFR0534::getFileNumber ( )
```

Get file number of current file.

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

Returns

File number

Return values

0	Error (for example request timeout)

Definition at line 421 of file DFR0534.cpp.

```
{
  #define COMMAND 0x0D
  #define RECEIVEFAILED 0
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
 m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 2) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          result=data«8;
          break;
        case 1:
          result+=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}
```

4.1.4.9 getFirstFileNumberInCurrentDirectory()

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

Get number of first file in current directory.

Returns

File number

Return values

-1	Error (for example request timeout)

Definition at line 589 of file DFR0534.cpp.

```
{
  #define COMMAND 0x11
  #define RECEIVEFAILED -1
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
 m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 2) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          result=data«8;
          break;
        case 1:
          result+=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}
```

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 1152 of file DFR0534.cpp.

```
{
  #define COMMAND 0x25
  #define RECEIVEFAILED false
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return false; // Should not happen

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
 m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 3) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          hour=data;
          break;
        case 1:
          minute=data;
          break;
        case 2:
          second=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  return (data == sum); // Does checksum matches?
}
```

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
```

```
  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);;
  sendDataByte(0x00);;
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff, result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 1) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      result = data;
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}
```

4.1.4.12 getTotalFiles()

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

Get total number of supported audio files on current drive.

Returns

Number of files

Return values

-1	Error (for example request timeout)

Definition at line 493 of file DFR0534.cpp.

```
{
  #define COMMAND 0x0C
  #define RECEIVEFAILED -1
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 2) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          result=data«8;
          break;
        case 1:
          result+=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}
```

4.1.4.13 getTotalFilesInCurrentDirectory()

`int DFR0534::getTotalFilesInCurrentDirectory ( )`

Count all audio files for the current directory.

Returns

File count

Return values

-1	Error (for example request timeout)

Definition at line 661 of file DFR0534.cpp.

```
{
  #define COMMAND 0x12
  #define RECEIVEFAILED -1
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 2) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          result=data«8;
          break;
        case 1:
          result+=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}
```

4.1.4.14 increaseVolume()

`void DFR0534::increaseVolume ( )`

Increase volume by one step.

Definition at line 730 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x14);
  sendDataByte(0x00);
  sendCheckSum();
}
```

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 759 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (drive >= DRIVEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x16);
  sendDataByte(0x03);
  sendDataByte(drive);
  sendDataByte((track » 8) & 0xff);
  sendDataByte(track & 0xff);
  sendCheckSum();
}
```

4.1.4.16 pause()

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

Pause the current file.

Definition at line 180 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x03);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.17 play()

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

Play the current selected file.

Definition at line 168 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x02);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.18 playCombined()

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

Combined/concatenated play of files.

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

Parameters

in	*list*	Concatenated list of all files to play

Definition at line 852 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (list == NULL) return;
  if ((strlen(list) % 2) != 0) return;

  sendStartingCode();
  sendDataByte(0x1B);
  sendDataByte(strlen(list));
  for (int i=0;i<strlen(list);i++) {
    sendDataByte(list[i]);
  }
  sendCheckSum();
}
```

4.1.4.19 playFileByName()

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

Play audio file by file name/path.

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

- Without the dot for the file extension
- All characters in upper case
- Every file and folder whose length is shorter then 8 chars must be filled up to the 8 chars length by spaces.
- Only WAV and MP3 files are supported Wildcards ∗ (=multiple arbitrary characters) and ? (=one single arbitrary character) are allowed and can be used to reduce filling spaces.

Valid examples:

- "/01 WAV" for file 01.wav
- "/99-AFR∼1MP3" for a file /99-Africa.mp3
- "/99-AFR∗MP3" for first file matching /99-Afr∗.mp3
- "/10∗" for first audio file matching /10∗.∗
- "/10 /20 WAV" for the file /10/20.wav

Parameters

in	*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 246 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (path == NULL) return;
  if (drive >= DRIVEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x08);
  sendDataByte(strlen(path)+1);
  sendDataByte(drive);
  for (int i=0;i<strlen(path);i++) {
    sendDataByte(path[i]);
  }
  sendCheckSum();
}
```

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.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (track <=0) return;
  sendStartingCode();
  sendDataByte(0x07);
  sendDataByte(0x02);
  sendDataByte((track » 8) & 0xff);
  sendDataByte(track & 0xff);
  sendCheckSum();
}
```

4.1.4.21 playLastInDirectory()

```
void DFR0534::playLastInDirectory ( )
```

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

Definition at line 562 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x0E);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.22 playNext()

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

Play next file (in "file copy order")

Definition at line 216 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x06);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.23 playNextDirectory()

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

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

Definition at line 574 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x0F);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.24 playPrevious()

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

Play previous file (in "file copy order")

Definition at line 204 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x05);
  sendDataByte(0x00);
  sendCheckSum();
}
```

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 966 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x1F);
  sendDataByte(0x02);
  sendDataByte((track » 8) & 0xff);
  sendDataByte(track & 0xff);
  sendCheckSum();
}
```

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 987 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x20);
  sendDataByte(0x04);
  sendDataByte(startMinute);
  sendDataByte(startSecond);
  sendDataByte(stopMinute);
  sendDataByte(stopSecond);
  sendCheckSum();
}
```

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 887 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (channel >= CHANNELUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x1D);
  sendDataByte(0x01);
  sendDataByte(channel);
  sendCheckSum();
}
```

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 792 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (path == NULL) return;
  if (drive >= DRIVEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x17);
  sendDataByte(strlen(path)+1);
  sendDataByte(drive);
  for (int i=0;i<strlen(path);i++) {
    sendDataByte(path[i]);
  }
  sendCheckSum();
}
```

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 402 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (drive >= DRIVEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x0B);
  sendDataByte(0x01);
  sendDataByte(drive);
  sendCheckSum();
}
```

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.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (mode >= EQUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x1A);
  sendDataByte(0x01);
  sendDataByte(mode);
  sendCheckSum();
}
```

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 812 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (mode >= PLAYMODEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x18);
  sendDataByte(0x01);
  sendDataByte(mode);
  sendCheckSum();
}
```

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 830 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x19);
  sendDataByte(0x02);
  sendDataByte((loops » 8) & 0xff);
  sendDataByte(loops & 0xff);
  sendCheckSum();
}
```

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.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (volume > 30) volume = 30;
  sendStartingCode();
  sendDataByte(0x13);
  sendDataByte(0x01);
  sendDataByte(volume);
  sendCheckSum();
}
```

4.1.4.34 startSendingRuntime()

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

Start sending elapsed runtime every 1 second.

Definition at line 1130 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x25);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.35 stop()

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

Stop the current file.

Definition at line 192 of file DFR0534.cpp.
```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x04);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.36 stopCombined()

`void DFR0534::stopCombined ( )`

Stop combined play (playlist)

Definition at line 870 of file DFR0534.cpp.
```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x1C);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.37 stopInsertedFile()

`void DFR0534::stopInsertedFile ( )`

Stop inserted file.

Continue original file

Definition at line 777 of file DFR0534.cpp.
```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x10);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.38 stopRepeatPart()

`void DFR0534::stopRepeatPart ( )`

Stop repeating part of the current file.

Definition at line 1003 of file DFR0534.cpp.
```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x21);
  sendDataByte(0x00);
  sendCheckSum();
}
```

4.1.4.39 stopSendingRuntime()

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

Stop sending runtime.

Definition at line 1219 of file DFR0534.cpp.

```
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x26);
  sendDataByte(0x00);
  sendCheckSum();
}
```

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

- DFR0534.h
- DFR0534.cpp

Chapter 5

File Documentation

5.1 playCombined.ino

```
/*
 * Example for using the DFR0534 for playing combined audio files like a playlist
 */

#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);

void setup() {
  // Serial for console output
  Serial.begin(9600);
  // Software serial for communication to DFR0534 module
  g_serial.begin(9600);

  // Set volume
  g_audio.setVolume(18);

  /* The parameter string for the playCombined function is just
   * a concatenation of all files in the desired order without
   * path and without extension.
   * All files have to be in the folder /ZH and the each
   * file has to have a length (without extension) of two chars.
   *
   * You can get example files from
https://github.com/codingABI/DFR0534/tree/main/assets/exampleContent
   */

  /* Plays files the custom order, like a playlist and stops after the last file:
   * /ZH/05.wav
   * /ZH/04.wav
   * /ZH/03.wav
   * /ZH/02.wav
   * /ZH/01.wav
   * /ZH/0A.wav
   */
  g_audio.playCombined("05040302010A");
}

void loop() {
  static unsigned long lastDisplayMS = millis();
  char name[12];

  // Show information about current track every 500ms
  if (millis()-lastDisplayMS > 500) {
    Serial.print("number: ");
    word fileNumber = g_audio.getFileNumber();
    if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");

    Serial.print(" name: ");
    if (g_audio.getFileName(name)) Serial.print(name);

    Serial.print(" status: ");
    switch (g_audio.getStatus()) {
      case DFR0534::STOPPED:
        Serial.println("Stopped");
        break;
      case DFR0534::PAUSED:
        Serial.println("Paused");
        break;
      case DFR0534::PLAYING:
        Serial.println("Playing");
        break;
      case DFR0534::STATUSUNKNOWN:
        Serial.println("Unknown");
        break;
    }
    lastDisplayMS = millis();
  }
}
```

5.2 playFileByName.ino

```
/*
 * Example for using the DFR0534 for playing audio files by file name
 */

#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);

void setup() {
  // Serial for console output
  Serial.begin(9600);
  // Software serial for communication to DFR0534 module
  g_serial.begin(9600);

  // Set volume
  g_audio.setVolume(18);

  /* The file name/path for the function playFileByName() is the full path of the audio file to be played
   * in format which looks like a special unix 8+3 format:
   * - Without the dot for the file extension
   * - All characters in upper case
   * - Every file and folder whose length is shorter then 8 chars must be filled up to the 8 chars length by spaces.
   * - Only WAV and MP3 files are supported
   * Wildcards * (=multiple arbitrary characters) and ? (=one single arbitrary character) are allowed and can be used to reduce filling spaces.
   *
   * Valid examples:
   * - "/01      WAV" for file 01.wav
   * - "/99-AFR~1MP3" for a file /99-Africa.mp3
   * - "/99-AFR*MP3" for first file matching /99-Afr*.mp3
   * - "/10*" for first audio file matching /10*.*
   * - "/10      /20      WAV" for the file /10/20.wav
   *
   * You can get example files from https://github.com/codingABI/DFR0534/tree/main/assets/exampleContent
   *
   * Valid examples:
   * "/01      WAV" for file 01.wav
   * "/99-AFR~1MP3" for a file /99-Africa.mp3
   * "/99-AFR*MP3" for first file matching /99-Afr*.mp3
   * "/10*" for first audio file matching /10*.*
   * "/10      /20      WAV" for the file /10/20.wav
   */

  // Play the file "test.wav"
  g_audio.playFileByName("/TEST    WAV");
}

void loop() {
  static unsigned long lastDisplayMS = millis()-500;
  char name[12];

  // Show information about current track once per second
  if (millis()-lastDisplayMS > 1000) {
    Serial.print("number: ");
    word fileNumber = g_audio.getFileNumber();
    if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");

    Serial.print(" name: ");
    if (g_audio.getFileName(name)) Serial.print(name);

    Serial.print(" status: ");
    switch (g_audio.getStatus()) {
      case DFR0534::STOPPED:
        Serial.println("Stopped");
        break;
      case DFR0534::PAUSED:
        Serial.println("Paused");
        break;
      case DFR0534::PLAYING:
        Serial.println("Playing");
        break;
      case DFR0534::STATUSUNKNOWN:
        Serial.println("Unknown");
        break;
    }
    lastDisplayMS = millis();
  }
}
```

5.3 playFileByNumber.ino

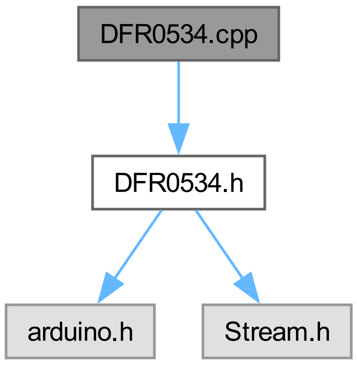
```
/*
 * Example for using the DFR0534 for playing audio files by file number
 */

#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);

void setup() {
  // Serial for console output
  Serial.begin(9600);
  // Software serial for communication to DFR0534 module
  g_serial.begin(9600);

  // Set volume
  g_audio.setVolume(18);

  // Show some device infos
  Serial.print("Ready drives: ");
  byte drive = g_audio.getDrivesStates();
  if (((drive » DFR0534::DRIVEUSB) & 1) == 1) Serial.print("USB ");
  if (((drive » DFR0534::DRIVESD) & 1) == 1) Serial.print("SD ");
  if (((drive » DFR0534::DRIVEFLASH) & 1) == 1) Serial.print("FLASH ");
  Serial.println();

  Serial.print("Current playing drive: ");
  switch(g_audio.getDrive()) {
    case DFR0534::DRIVEUSB:
      Serial.println("USB");
      break;
    case DFR0534::DRIVESD:
      Serial.println("SD");
      break;
    case DFR0534::DRIVEFLASH:
      Serial.println("FLASH");
      break;
    case DFR0534::DRIVENO:
      Serial.println("No drive");
      break;
    default:
      Serial.println("Unknown");
      break;
  }

  Serial.print("Total files: ");
  Serial.println(g_audio.getTotalFiles());
  Serial.print("Total files in directory: ");
  Serial.println(g_audio.getTotalFilesInCurrentDirectory());

  Serial.print("First file: ");
  Serial.println(g_audio.getFirstFileNumberInCurrentDirectory());

  // Play the first audio file copied to the DFR0534
  // (Second file copied to the DFR0534 would be number 2...)
  g_audio.playFileByNumber(1);
}
```

```

void loop() {
  static unsigned long lastDisplayMS = millis()-500;
  char name[12];

  // Show information about current track once per second
  if (millis()-lastDisplayMS > 1000) {
    Serial.print("number: ");
    word fileNumber = g_audio.getFileNumber();
    if (fileNumber > 0) Serial.print(fileNumber); else Serial.print("--");

    Serial.print(" name: ");
    if (g_audio.getFileName(name)) Serial.print(name);

    Serial.print(" status: ");
    switch (g_audio.getStatus()) {
      case DFR0534::STOPPED:
        Serial.println("Stopped");
        break;
      case DFR0534::PAUSED:
        Serial.println("Paused");
        break;
      case DFR0534::PLAYING:
        Serial.println("Playing");
        break;
      case DFR0534::STATUSUNKNOWN:
        Serial.println("Unknown");
        break;
    }
    lastDisplayMS = millis();
  }
}
```

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

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

1.0.1

Definition in file DFR0534.cpp.

5.5 DFR0534.cpp

Go to the documentation of this file.

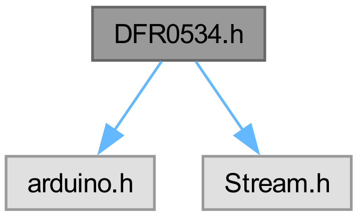
```
#include "DFR0534.h"

byte DFR0534::getStatus()
{
  #define COMMAND 0x01
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEFAILED STATUSUNKNOWN
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);;
  sendDataByte(0x00);;
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff, result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 1) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      result = data;
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}

void DFR0534::setEqualizer(byte mode)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (mode >= EQUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x1A);
  sendDataByte(0x01);
  sendDataByte(mode);
  sendCheckSum();
}

void DFR0534::playFileByNumber(word track)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (track <=0) return;
  sendStartingCode();
  sendDataByte(0x07);
  sendDataByte(0x02);
  sendDataByte((track » 8) & 0xff);
  sendDataByte(track & 0xff);
  sendCheckSum();
}

void DFR0534::setVolume(byte volume)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (volume > 30) volume = 30;
  sendStartingCode();
  sendDataByte(0x13);
  sendDataByte(0x01);
  sendDataByte(volume);
  sendCheckSum();
}

void DFR0534::play()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x02);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::pause()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x03);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::stop()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x04);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::playPrevious()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x05);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::playNext()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x06);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::playFileByName(char *path, byte drive)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (path == NULL) return;
  if (drive >= DRIVEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x08);
  sendDataByte(strlen(path)+1);
  sendDataByte(drive);
  for (int i=0;i<strlen(path);i++) {
    sendDataByte(path[i]);
  }
  sendCheckSum();
}

byte DFR0534::getDrivesStates()
{
  #define COMMAND 0x09
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEFAILED DRIVEUNKNOWN
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff, result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 1) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      result = data;
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}

byte DFR0534::getDrive()
{
  #define COMMAND 0x0A
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEFAILED DRIVEUNKNOWN
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff, result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 1) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      result = data;
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}

```

```
void DFR0534::setDrive(byte drive)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (drive >= DRIVEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x0B);
  sendDataByte(0x01);
  sendDataByte(drive);
  sendCheckSum();
}

word DFR0534::getFileNumber()
{
  #define COMMAND 0x0D
  #define RECEIVEFAILED 0
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 2) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          result=data«8;
          break;
        case 1:
          result+=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}

int DFR0534::getTotalFiles()
{
  #define COMMAND 0x0C
  #define RECEIVEFAILED -1
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 2) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          result=data«8;
          break;
        case 1:
          result+=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}

void DFR0534::playLastInDirectory()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x0E);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::playNextDirectory()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x0F);
  sendDataByte(0x00);
  sendCheckSum();
}

int DFR0534::getFirstFileNumberInCurrentDirectory()
{
  #define COMMAND 0x11
  #define RECEIVEFAILED -1
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 2) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          result=data«8;
          break;
        case 1:
          result+=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}

int DFR0534::getTotalFilesInCurrentDirectory()
{
  #define COMMAND 0x12
  #define RECEIVEFAILED -1
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) RECEIVEFAILED; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 2) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          result=data«8;
          break;
        case 1:
          result+=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  if (data != sum) return RECEIVEFAILED; // Does checksum matches?
  return result;
}

void DFR0534::increaseVolume()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x14);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::decreaseVolume()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x15);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::insertFileByNumber(word track, byte drive)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (drive >= DRIVEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x16);
  sendDataByte(0x03);
  sendDataByte(drive);
  sendDataByte((track » 8) & 0xff);
  sendDataByte(track & 0xff);
  sendCheckSum();
}

void DFR0534::stopInsertedFile()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x10);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::setDirectory(char *path, byte drive)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (path == NULL) return;
  if (drive >= DRIVEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x17);
  sendDataByte(strlen(path)+1);
  sendDataByte(drive);
  for (int i=0;i<strlen(path);i++) {
    sendDataByte(path[i]);
  }
  sendCheckSum();
}

void DFR0534::setLoopMode(byte mode)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (mode >= PLAYMODEUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x18);
  sendDataByte(0x01);
  sendDataByte(mode);
  sendCheckSum();
}

void DFR0534::setRepeatLoops(word loops)
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x19);
  sendDataByte(0x02);
  sendDataByte((loops » 8) & 0xff);
  sendDataByte(loops & 0xff);
  sendCheckSum();
}

void DFR0534::playCombined(char* list)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (list == NULL) return;
  if ((strlen(list) % 2) != 0) return;

  sendStartingCode();
  sendDataByte(0x1B);
  sendDataByte(strlen(list));
  for (int i=0;i<strlen(list);i++) {
    sendDataByte(list[i]);
  }
  sendCheckSum();
}

void DFR0534::stopCombined()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x1C);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::setChannel(byte channel)
{
  if (m_ptrStream == NULL) return; // Should not happen
  if (channel >= CHANNELUNKNOWN) return;
  sendStartingCode();
  sendDataByte(0x1D);
  sendDataByte(0x01);
  sendDataByte(channel);
  sendCheckSum();
}

bool DFR0534::getFileName(char *name)
{
  #define COMMAND 0x1E
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEFAILED false
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return false; // Should not happen
  if (name == NULL) return false;
  name[0] = '\0';

  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady =
m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();
    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) length = data; // Length of receiving string
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      if ((i-RECEIVEHEADERLENGTH) < 12) { // I expect no longer file names than 8+3 chars plus '\0'
        name[i-RECEIVEHEADERLENGTH-1] = data;
        name[i-RECEIVEHEADERLENGTH] = '\0';
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);
  return (data == sum); // Does checksum matches?
}

void DFR0534::prepareFileByNumber(word track)
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x1F);
  sendDataByte(0x02);
  sendDataByte((track » 8) & 0xff);
  sendDataByte(track & 0xff);
  sendCheckSum();
}

void DFR0534::repeatPart(byte startMinute, byte startSecond, byte stopMinute, byte stopSecond )
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x20);
  sendDataByte(0x04);
  sendDataByte(startMinute);
  sendDataByte(startSecond);
  sendDataByte(stopMinute);
  sendDataByte(stopSecond);
  sendCheckSum();
}

void DFR0534::stopRepeatPart()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x21);
  sendDataByte(0x00);
  sendCheckSum();
}

void DFR0534::fastBackwardDuration(word seconds)
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x22);
  sendDataByte(0x02);
  sendDataByte((seconds » 8) & 0xff);
  sendDataByte(seconds & 0xff);
  sendCheckSum();
}

void DFR0534::fastForwardDuration(word seconds)
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x23);
  sendDataByte(0x02);
  sendDataByte((seconds » 8) & 0xff);
  sendDataByte(seconds & 0xff);
  sendCheckSum();
}

bool DFR0534::getDuration(byte &hour, byte &minute, byte &second)
{
  #define COMMAND 0x24
  #define RECEIVEFAILED false
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return false; // Should not happen
  sendStartingCode();
  sendDataByte(COMMAND);
  sendDataByte(0x00);
  sendCheckSum();

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 3) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          hour=data;
          break;
        case 1:
          minute=data;
          break;
        case 2:
          second=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  return (data == sum); // Does checksum matches?
}

void DFR0534::startSendingRuntime()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x25);
  sendDataByte(0x00);
  sendCheckSum();
}

bool DFR0534::getRuntime(byte &hour, byte &minute, byte &second)
{
  #define COMMAND 0x25
  #define RECEIVEFAILED false
  #define RECEIVEBYTETIMEOUTMS 100
  #define RECEIVEGLOBALTIMEOUTMS 500
  #define RECEIVEHEADERLENGTH 2 // startingcode+command

  if (m_ptrStream == NULL) return false; // Should not happen

  // Receive
  int i=0;
  byte data, firstByte = 0, sum, length=0xff;
  word result = 0;
  unsigned long receiveStartMS = millis();
  do {
    byte dataReady = 0;
    unsigned long lastMS = millis();
    // Wait for response or timeout
    while ((millis()-lastMS < RECEIVEBYTETIMEOUTMS) && (dataReady==0)) dataReady = m_ptrStream->available();

    if (dataReady == 0) return RECEIVEFAILED; // Timeout
    data = m_ptrStream->read();

    if (i==0) { // Begin of transmission
      firstByte=data;
      sum = 0;
    }
    if ((i == 1) && (data != COMMAND)) {
      // Invalid signal => reset receive
      i=0;
      firstByte = 0;
    }
    if (i == RECEIVEHEADERLENGTH) {
      length = data; // Length of receiving data
      if (length != 3) {
        // Invalid length => reset receive
        i=0;
        firstByte = 0;
      }
    }
    if ((i > RECEIVEHEADERLENGTH) && (i-RECEIVEHEADERLENGTH-1<length)) {
      switch (i-RECEIVEHEADERLENGTH-1) {
        case 0:
          hour=data;
          break;
        case 1:
          minute=data;
          break;
        case 2:
          second=data;
          break;
      }
    }
    if (firstByte == STARTINGCODE) {
      if (i-RECEIVEHEADERLENGTH<=length) sum+=data; // Update checksum
      i++;
    }
    if (millis()-receiveStartMS > RECEIVEGLOBALTIMEOUTMS) return RECEIVEFAILED; // Timeout
  } while (i<length+RECEIVEHEADERLENGTH+2);

  return (data == sum); // Does checksum matches?
}

void DFR0534::stopSendingRuntime()
{
  if (m_ptrStream == NULL) return; // Should not happen
  sendStartingCode();
  sendDataByte(0x26);
  sendDataByte(0x00);
  sendCheckSum();
}
```

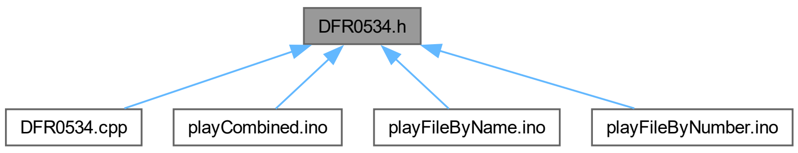
5.6 DFR0534.h File Reference

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

Include dependency graph for DFR0534.h:



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



Classes

- class DFR0534

  Class for a DFR0534 audio module.

Macros

- #define DFR0534_VERSION "1.0.1"

5.6.1 Detailed Description

Class: DFR0534

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

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

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

Author

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

Copyright

2-Clause BSD License

Version

1.0.1

Definition in file DFR0534.h.

5.6.2 Macro Definition Documentation

5.6.2.1 DFR0534_VERSION

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

Library version

Definition at line 22 of file DFR0534.h.

5.7 DFR0534.h

Go to the documentation of this file.

```
00001
00019 #pragma once
00020
00022 #define DFR0534_VERSION "1.0.1"
00023
00024 #include <arduino.h>
00025 #include <Stream.h>
00026
00027 #define STARTINGCODE 0xAA
00028
00032 class DFR0534 {
00033   public:
00035     enum DFR0534CHANNELS
00036     {
00037       CHANNELMP3,
00038       CHANNELAUX,
00039       CHANNELMP3AUX,
00040       CHANNELUNKNOWN
00041     };
00043     enum DFR0534DRIVE
00044     {
00045       DRIVEUSB,
00046       DRIVESD,
00047       DRIVEFLASH,
00048       DRIVEUNKNOWN,
00049       DRIVENO = 0xff
00050     };
00052     enum DFR0534LOOPMODE
00053     {
00054       LOOPBACKALL,
00055       SINGLEAUDIOLOOP,
00056       SINGLEAUDIOSTOP,
00057       PLAYRANDOM,
00058       DIRECTORYLOOP,
00059       RANDOMINDIRECTORY,
00060       SEQUENTIALINDIRECTORY,
00061       SEQUENTIAL,
00062       PLAYMODEUNKNOWN
00063     };
00065     enum DFR0534EQ
00066     {
00067       NORMAL,
00068       POP,
00069       ROCK,
      JAZZ ,
      CLASSIC,
      EQUNKNOWN
    };
    enum DFR0534STATUS
    {
      STOPPED,
      PLAYING,
      PAUSED,
      STATUSUNKNOWN
    };
    DFR0534(Stream &stream)
    {
      m_ptrStream = &stream;
    }
    void decreaseVolume();
    void fastBackwardDuration(word seconds);
    void fastForwardDuration(word seconds);
    byte getDrive();
    byte getDrivesStates();
    bool getDuration(byte &hour, byte &minute, byte &second);
    bool getFileName(char *name);
    word getFileNumber();
    int getFirstFileNumberInCurrentDirectory();
    bool getRuntime(byte &hour, byte &minute, byte &second);
    byte getStatus();
    int getTotalFiles();
    int getTotalFilesInCurrentDirectory();
    void increaseVolume();
    void insertFileByNumber(word track, byte drive=DRIVEFLASH);
    void pause();
    void play();
    void playCombined(char* list);
    void playFileByName(char *path, byte drive=DRIVEFLASH);
    void playFileByNumber(word track);
    void playLastInDirectory();
    void playNext();
    void playNextDirectory();
    void playPrevious();
    void prepareFileByNumber(word track);
    void repeatPart(byte startMinute, byte startSecond, byte stopMinute, byte stopSecond );
    void setChannel(byte channel);
    void setDirectory(char *path, byte drive=DRIVEFLASH);
    void setDrive(byte drive);
    void setEqualizer(byte mode);
    void setLoopMode(byte mode);
    void setRepeatLoops(word loops);
    void setVolume(byte volume);
    void stop();
    void stopInsertedFile();
    void startSendingRuntime();
    void stopCombined();
    void stopRepeatPart();
    void stopSendingRuntime();
  private:
    void sendStartingCode() {
      m_checksum=STARTINGCODE;
      m_ptrStream->write((byte)STARTINGCODE);
    }
    void sendDataByte(byte data) {
      m_checksum +=data;
      m_ptrStream->write((byte)data);
    }
    void sendCheckSum() {
      m_ptrStream->write((byte)m_checksum);
    }
    byte m_checksum;
    Stream *m_ptrStream = NULL;
};
```

Index