Arduino MIDI Library

Version 3.1.1

Generated by Doxygen 1.7.4

Fri May 20 2011 19:49:48

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 MIDI_Class Class Reference

The main class for MIDI handling.

See member descriptions to know how to use it, or check out the examples supplied with the library.

`#include <MIDI.h>`

Public Member Functions

- MIDI_Class ()

  Default constructor for MIDI_Class.
- ~MIDI_Class ()

  Default destructor for MIDI_Class.
- void begin (const byte inChannel=1)

  Call the begin method in the setup() function of the Arduino.
- void sendNoteOn (byte NoteNumber, byte Velocity, byte Channel)

  Send a Note On message.
- void sendNoteOff (byte NoteNumber, byte Velocity, byte Channel)

  Send a Note Off message (a real Note Off, not a Note On with null velocity)
- void sendProgramChange (byte ProgramNumber, byte Channel)

  Send a Program Change message.
- void sendControlChange (byte ControlNumber, byte ControlValue, byte Channel)

  Send a Control Change message.
- void sendPitchBend (int PitchValue, byte Channel)

  Send a Pitch Bend message using a signed integer value.
- void sendPitchBend (unsigned int PitchValue, byte Channel)

  Send a Pitch Bend message using an unsigned integer value.

- void sendPitchBend (double PitchValue, byte Channel)

  Send a Pitch Bend message using a floating point value.
- void sendPolyPressure (byte NoteNumber, byte Pressure, byte Channel)

  Send a Polyphonic AfterTouch message (applies to only one specified note)
- void sendAfterTouch (byte Pressure, byte Channel)

  Send a MonoPhonic AfterTouch message (applies to all notes)
- void sendSysEx (byte length, byte *array, bool ArrayContainsBoundaries=false)

  Generate and send a System Exclusive frame.
- void sendTimeCodeQuarterFrame (byte TypeNibble, byte ValuesNibble)

  Send a MIDI Time Code Quarter Frame.
- void sendTimeCodeQuarterFrame (byte data)

  Send a MIDI Time Code Quarter Frame.
- void sendSongPosition (unsigned int Beats)

  Send a Song Position Pointer message.
- void sendSongSelect (byte SongNumber)

  Send a Song Select message.
- void sendTuneRequest ()

  Send a Tune Request message.
- void sendRealTime (kMIDIType Type)

  Send a Real Time (one byte) message.
- void send (kMIDIType type, byte param1, byte param2, byte channel)

  Generate and send a MIDI message from the values given.
- bool read ()

  Read a MIDI message from the serial port using the main input channel (see setInputChannel() for reference).
- bool read (const byte Channel)

  Reading/thru-ing method, the same as read() with a given input channel to read on.
- kMIDIType getType ()

  Get the last received message's type.
- byte getChannel ()

  Get the channel of the message stored in the structure.
- byte getData1 ()

  Get the first data byte of the last received message.
- byte getData2 ()

  Get the second data byte of the last received message.
- byte * getSysExArray ()

  Get the System Exclusive byte array.
- bool check ()

  Check if a valid message is stored in the structure.
- byte getInputChannel ()
- void setInputChannel (const byte Channel)

  Set the value for the input MIDI channel.
- void setHandleNoteOff (void(*fptr)(byte channel, byte note, byte velocity))

- void setHandleNoteOn (void(*fptr)(byte channel, byte note, byte velocity))
- void setHandleAfterTouchPoly (void(*fptr)(byte channel, byte note, byte pressure))
- void setHandleControlChange (void(*fptr)(byte channel, byte number, byte value))
- void setHandleProgramChange (void(*fptr)(byte channel, byte number))
- void setHandleAfterTouchChannel (void(*fptr)(byte channel, byte pressure))
- void setHandlePitchBend (void(*fptr)(byte channel, int bend))
- void setHandleSystemExclusive (void(*fptr)(byte *array, byte size))
- void setHandleTimeCodeQuarterFrame (void(*fptr)(byte data))
- void setHandleSongPosition (void(*fptr)(unsigned int beats))
- void setHandleSongSelect (void(*fptr)(byte songnumber))
- void setHandleTuneRequest (void(*fptr)(void))
- void setHandleClock (void(*fptr)(void))
- void setHandleStart (void(*fptr)(void))
- void setHandleContinue (void(*fptr)(void))
- void setHandleStop (void(*fptr)(void))
- void setHandleActiveSensing (void(*fptr)(void))
- void setHandleSystemReset (void(*fptr)(void))
- void disconnectCallbackFromType (kMIDIType Type)

  Detach an external function from the given type.
- kThruFilterMode getFilterMode ()
- bool getThruState ()
- void turnThruOn (kThruFilterMode inThruFilterMode=Full)

  Setter method: turn message mirroring on.
- void turnThruOff ()

  Setter method: turn message mirroring off.
- void setThruFilterMode (const kThruFilterMode inThruFilterMode)

  Set the filter for thru mirroring.

Static Public Member Functions

- static const kMIDIType getTypeFromStatusByte (const byte inStatus)

  Extract an enumerated MIDI type from a status byte.

3.1.1 Detailed Description

The main class for MIDI handling.

See member descriptions to know how to use it, or check out the examples supplied with the library.

Definition at line 120 of file MIDI.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 MIDI_Class::MIDI_Class ( )

Default constructor for MIDI_Class.

Definition at line 22 of file MIDI.cpp.

```
                        {
#if USE_CALLBACKS
    // Initialise callbacks to NULL pointer
    mNoteOffCallback = NULL;
    mNoteOnCallback = NULL;
    mAfterTouchPolyCallback = NULL;
    mControlChangeCallback = NULL;
    mProgramChangeCallback = NULL;
    mAfterTouchChannelCallback = NULL;
    mPitchBendCallback = NULL;
    mSystemExclusiveCallback = NULL;
    mTimeCodeQuarterFrameCallback = NULL;
    mSongPositionCallback = NULL;
    mSongSelectCallback = NULL;
    mTuneRequestCallback = NULL;
    mClockCallback = NULL;
    mStartCallback = NULL;
    mContinueCallback = NULL;
    mStopCallback = NULL;
    mActiveSensingCallback = NULL;
    mSystemResetCallback = NULL;
#endif
}
```

3.1.2.2 MIDI_Class::~MIDI_Class ( )

Default destructor for MIDI_Class.

This is not really useful for the Arduino, as it is never called...

Definition at line 49 of file MIDI.cpp.

```
{ }
```

3.1.3 Member Function Documentation

3.1.3.1 void MIDI_Class::begin (const byte *inChannel* = 1)

Call the begin method in the setup() function of the Arduino.

All parameters are set to their default values:

- Input channel set to 1 if no value is specified
- Full thru mirroring

Definition at line 58 of file MIDI.cpp.

```
                                            {

    // Initialise the Serial port
    USE_SERIAL_PORT.begin(MIDI_BAUDRATE);


#if COMPILE_MIDI_OUT

#if USE_RUNNING_STATUS
    mRunningStatus_TX = InvalidType;
#endif // USE_RUNNING_STATUS

#endif // COMPILE_MIDI_OUT


#if COMPILE_MIDI_IN

    mInputChannel = inChannel;
    mRunningStatus_RX = InvalidType;
    mPendingMessageIndex = 0;
    mPendingMessageExpectedLenght = 0;

    mMessage.valid = false;
    mMessage.type = InvalidType;
    mMessage.channel = 0;
    mMessage.data1 = 0;
    mMessage.data2 = 0;

#endif // COMPILE_MIDI_IN


#if (COMPILE_MIDI_IN && COMPILE_MIDI_OUT && COMPILE_MIDI_THRU) // Thru

    mThruFilterMode = Full;
    mThruActivated = true;

#endif // Thru

}
```

3.1.3.2 bool MIDI_Class::check ( )

Check if a valid message is stored in the structure.

Definition at line 696 of file MIDI.cpp.

```
{ return mMessage.valid; }
```

3.1.3.3 void MIDI_Class::disconnectCallbackFromType (kMIDIType *Type*)

Detach an external function from the given type.

Use this method to cancel the effects of setHandle********.

Parameters

Type	The type of message to unbind. When a message of this type is received, no function will be called.

Definition at line 733 of file MIDI.cpp.

```
                                                    {

    switch (Type) {
        case NoteOff:               mNoteOffCallback = NULL;               break
      ;
        case NoteOn:                mNoteOnCallback = NULL;                break
      ;
        case AfterTouchPoly:        mAfterTouchPolyCallback = NULL;        break
      ;
        case ControlChange:         mControlChangeCallback = NULL;         break
      ;
        case ProgramChange:         mProgramChangeCallback = NULL;         break
      ;
        case AfterTouchChannel:     mAfterTouchChannelCallback = NULL;     break
      ;
        case PitchBend:             mPitchBendCallback = NULL;             break
      ;
        case SystemExclusive:       mSystemExclusiveCallback = NULL;       break
      ;
        case TimeCodeQuarterFrame:  mTimeCodeQuarterFrameCallback = NULL;  break
      ;
        case SongPosition:          mSongPositionCallback = NULL;          break
      ;
        case SongSelect:            mSongSelectCallback = NULL;            break
      ;
        case TuneRequest:           mTuneRequestCallback = NULL;           break
      ;
        case Clock:                 mClockCallback = NULL;                 break
      ;
        case Start:                 mStartCallback = NULL;                 break
      ;
        case Continue:              mContinueCallback = NULL;              break
      ;
        case Stop:                  mStopCallback = NULL;                  break
      ;
        case ActiveSensing:         mActiveSensingCallback = NULL;         break
      ;
        case SystemReset:           mSystemResetCallback = NULL;           break
      ;
        default:
            break;
    }

}
```

3.1.3.4 byte MIDI_Class::getChannel ( )

Get the channel of the message stored in the structure.

Channel range is 1 to 16. For non-channel messages, this will return 0.

Definition at line 678 of file MIDI.cpp.

```
{ return mMessage.channel; }
```

3.1.3.5 byte MIDI_Class::getData1 ( )

Get the first data byte of the last received message.

If the message is SysEx, the length of the array is stocked there.

Definition at line 684 of file MIDI.cpp.

```
{ return mMessage.data1; }
```

3.1.3.6 byte MIDI_Class::getData2 ( )

Get the second data byte of the last received message.

Definition at line 687 of file MIDI.cpp.

```
{ return mMessage.data2; }
```

3.1.3.7 kThruFilterMode MIDI_Class::getFilterMode ( ) [inline]

Definition at line 285 of file MIDI.h.

```
{ return mThruFilterMode; }
```

3.1.3.8 byte MIDI_Class::getInputChannel ( ) [inline]

Definition at line 188 of file MIDI.h.

```
{ return mInputChannel; }
```

3.1.3.9 byte * MIDI_Class::getSysExArray ( )

Get the System Exclusive byte array.

Array length is stocked in Data1.

Definition at line 693 of file MIDI.cpp.

```
{ return mMessage.sysex_array; }
```

3.1.3.10 bool MIDI_Class::getThruState ( ) [inline]

Definition at line 286 of file MIDI.h.

```
{ return mThruActivated; }
```

3.1.3.11 kMIDIType MIDI_Class::getType ( )

Get the last received message's type.

Returns an enumerated type.

See also

kMIDIType

Definition at line 672 of file MIDI.cpp.

```
{ return mMessage.type; }
```

3.1.3.12 static const kMIDIType MIDI_Class::getTypeFromStatusByte (const byte *inStatus*) [inline, static]

Extract an enumerated MIDI type from a status byte.

This is a utility static method, used internally, made public so you can handle kMIDITypes more easily.

Definition at line 197 of file MIDI.h.

```
                                                                        {
        if ((inStatus < 0x80)
            || (inStatus == 0xF4)
            || (inStatus == 0xF5)
            || (inStatus == 0xF9)
            || (inStatus == 0xFD)) return InvalidType; // data bytes and undefine
      d.
        if (inStatus < 0xF0) return (kMIDIType)(inStatus & 0xF0);   // Channel me
      ssage, remove channel nibble.
        else return (kMIDIType)inStatus;
    }
```

3.1.3.13 bool MIDI_Class::read (const byte *Channel*)

Reading/thru-ing method, the same as read() with a given input channel to read on.

Definition at line 350 of file MIDI.cpp.

```
                                        {

    if (inChannel >= MIDI_CHANNEL_OFF) return false; // MIDI Input disabled.


    if (parse(inChannel)) {
        if (input_filter(inChannel)) {

#if (COMPILE_MIDI_OUT && COMPILE_MIDI_THRU)
            thru_filter(inChannel);
#endif

#if USE_CALLBACKS
            launchCallback();
#endif

            return true;
        }
    }

    return false;
}
```

3.1.3.14 bool MIDI_Class::read ( )

Read a MIDI message from the serial port using the main input channel (see setInputChannel() for reference).

Returned value: true if any valid message has been stored in the structure, false if not. A valid message is a message that matches the input channel.

If the Thru is enabled and the messages matches the filter, it is sent back on the MIDI output.

Definition at line 345 of file MIDI.cpp.

```
                    {
    return read(mInputChannel);
}
```

3.1.3.15 void MIDI_Class::send (kMIDIType *type*, byte *data1*, byte *data2*, byte *channel*)

Generate and send a MIDI message from the values given.

Parameters

type	The message type (see type defines for reference)
data1	The first data byte.
data2	The second data byte (if the message contains only 1 data byte, set this one to 0).
channel	The output channel on which the message will be sent (values from 1 to 16). Note: you cannot send to OMNI.

This is an internal method, use it only if you need to send raw data from your code, at your own risks.

Definition at line 114 of file MIDI.cpp.

```
                                                                                {

    // Then test if channel is valid
    if (channel >= MIDI_CHANNEL_OFF || channel == MIDI_CHANNEL_OMNI || type <
      NoteOff) {

#if USE_RUNNING_STATUS
        mRunningStatus_TX = InvalidType;
#endif

        return; // Don't send anything
    }

    if (type <= PitchBend) {
        // Channel messages

        // Protection: remove MSBs on data
        data1 &= 0x7F;
        data2 &= 0x7F;

        byte statusbyte = genstatus(type,channel);

#if USE_RUNNING_STATUS
        // Check Running Status
        if (mRunningStatus_TX != statusbyte) {
            // New message, memorise and send header
            mRunningStatus_TX = statusbyte;
            USE_SERIAL_PORT.write(mRunningStatus_TX);
        }
#else
        // Don't care about running status, send the Control byte.
        USE_SERIAL_PORT.write(statusbyte);
#endif

        // Then send data
        USE_SERIAL_PORT.write(data1);
        if (type != ProgramChange && type != AfterTouchChannel) {
            USE_SERIAL_PORT.write(data2);
        }
        return;
    }
    if (type >= TuneRequest && type <= SystemReset) {
        // System Real-time and 1 byte.
        sendRealTime(type);
    }

}
```

3.1.3.16 void MIDI_Class::sendAfterTouch (byte *Pressure,* byte *Channel*)

Send a MonoPhonic AfterTouch message (applies to all notes)

Parameters

Pressure	The amount of AfterTouch to apply to all notes.
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 199 of file MIDI.cpp.

```
{ send(AfterTouchChannel,Pressure,0,Channel); }
```

3.1.3.17 void MIDI_Class::sendControlChange (byte *ControlNumber,* byte *ControlValue,* byte *Channel*)

Send a Control Change message.

Parameters

ControlNumber	The controller number (0 to 127). See the detailed description here: http://www.somascape.org/midi/tech/spec.html#ctrlnums
ControlValue	The value for the specified controller (0 to 127).
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 186 of file MIDI.cpp.

```
{ send(ControlChange,ControlNumber,ControlValue,Channel); }
```

3.1.3.18 void MIDI_Class::sendNoteOff (byte *NoteNumber,* byte *Velocity,* byte *Channel*)

Send a Note Off message (a real Note Off, not a Note On with null velocity)

Parameters

NoteNumber	Pitch value in the MIDI format (0 to 127). Take a look at the values, names and frequencies of notes here: http://www.phys.unsw.edu.au/jw/notes.html
Velocity	Release velocity (0 to 127).
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 173 of file MIDI.cpp.

```
{ send(NoteOff,NoteNumber,Velocity,Channel); }
```

3.1.3.19 void MIDI_Class::sendNoteOn (byte *NoteNumber,* byte *Velocity,* byte *Channel*)

Send a Note On message.

Parameters

NoteNumber	Pitch value in the MIDI format (0 to 127). Take a look at the values, names and frequencies of notes here: http://www.phys.unsw.edu.au/jw/notes.html
Velocity	Note attack velocity (0 to 127). A NoteOn with 0 velocity is considered as a NoteOff.
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 166 of file MIDI.cpp.

```
{ send(NoteOn,NoteNumber,Velocity,Channel); }
```

3.1.3.20 void MIDI_Class::sendPitchBend (double *PitchValue*, byte *Channel*)

Send a Pitch Bend message using a floating point value.

Parameters

PitchValue	The amount of bend to send (in a floating point format), between -1.0f (maximum downwards bend) and +1.0f (max upwards bend), center value is 0.0f.
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 224 of file MIDI.cpp.

```
                                                                {

    unsigned int pitchval = (PitchValue+1.f)*8192;
    if (pitchval > 16383) pitchval = 16383;     // overflow protection
    sendPitchBend(pitchval,Channel);

}
```

3.1.3.21 void MIDI_Class::sendPitchBend (int *PitchValue*, byte *Channel*)

Send a Pitch Bend message using a signed integer value.

Parameters

PitchValue	The amount of bend to send (in a signed integer format), between -8192 (maximum downwards bend) and 8191 (max upwards bend), center value is 0.
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 205 of file MIDI.cpp.

```
                                                                {
```

```
    unsigned int bend = PitchValue + 8192;
    sendPitchBend(bend,Channel);

}
```

3.1.3.22 void MIDI_Class::sendPitchBend (unsigned int *PitchValue,* byte *Channel*)

Send a Pitch Bend message using an unsigned integer value.

Parameters

PitchValue	The amount of bend to send (in a signed integer format), between 0 (maximum downwards bend) and 16383 (max upwards bend), center value is 8192.
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 215 of file MIDI.cpp.

```
                                                                    {

    send(PitchBend,(PitchValue & 0x7F),(PitchValue >> 7) & 0x7F,Channel);

}
```

3.1.3.23 void MIDI_Class::sendPolyPressure (byte *NoteNumber,* byte *Pressure,* byte *Channel*)

Send a Polyphonic AfterTouch message (applies to only one specified note)

Parameters

NoteNumber	The note to apply AfterTouch to (0 to 127).
Pressure	The amount of AfterTouch to apply (0 to 127).
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 193 of file MIDI.cpp.

```
{ send(AfterTouchPoly,NoteNumber,Pressure,Channel); }
```

3.1.3.24 void MIDI_Class::sendProgramChange (byte *ProgramNumber,* byte *Channel*)

Send a Program Change message.

Parameters

Program-Number	The Program to select (0 to 127).
Channel	The channel on which the message will be sent (1 to 16).

Definition at line 179 of file MIDI.cpp.

```
{ send(ProgramChange,ProgramNumber,0,Channel); }
```

3.1.3.25 void MIDI_Class::sendRealTime (kMIDIType *Type*)

Send a Real Time (one byte) message.

Parameters

Type	The available Real Time types are: Start, Stop, Continue, Clock, ActiveSensing and SystemReset. You can also send a Tune Request with this method.

See also

kMIDIType

Definition at line 309 of file MIDI.cpp.

```
                                                {
    switch (Type) {
        case TuneRequest: // Not really real-time, but one byte anyway.
        case Clock:
        case Start:
        case Stop:
        case Continue:
        case ActiveSensing:
        case SystemReset:
            USE_SERIAL_PORT.write((byte)Type);
            break;
        default:
            // Invalid Real Time marker
            break;
    }

    // Do not cancel Running Status for real-time messages as they can be interle
      aved within any message.
    // Though, TuneRequest can be sent here, and as it is a System Common message
      , it must reset Running Status.
#if USE_RUNNING_STATUS
    if (Type == TuneRequest) mRunningStatus_TX = InvalidType;
#endif

}
```

3.1.3.26 void MIDI_Class::sendSongPosition (unsigned int *Beats*)

Send a Song Position Pointer message.

Parameters

Beats	The number of beats since the start of the song.

Definition at line 283 of file MIDI.cpp.

```
                                                    {

    USE_SERIAL_PORT.write((byte)SongPosition);
    USE_SERIAL_PORT.write(Beats & 0x7F);
    USE_SERIAL_PORT.write((Beats >> 7) & 0x7F);
#if USE_RUNNING_STATUS
    mRunningStatus_TX = InvalidType;
#endif
}
```

3.1.3.27 void MIDI_Class::sendSongSelect (byte *SongNumber*)

Send a Song Select message.

Definition at line 294 of file MIDI.cpp.

```
                                                {

    USE_SERIAL_PORT.write((byte)SongSelect);
    USE_SERIAL_PORT.write(SongNumber & 0x7F);
#if USE_RUNNING_STATUS
    mRunningStatus_TX = InvalidType;
#endif
}
```

3.1.3.28 void MIDI_Class::sendSysEx (byte *length*, byte * *array*, bool *ArrayContainsBoundaries* = `false`)

Generate and send a System Exclusive frame.

Parameters

length	The size of the array to send
array	The byte array containing the data to send
ArrayContainsBoundaries	When set to 'true', 0xF0 & 0xF7 bytes (start & stop SysEx) will NOT be sent (and therefore must be included in the array). default value is set to 'false' for compatibility with previous versions of the library.

Definition at line 238 of file MIDI.cpp.

```
       {
    if (!ArrayContainsBoundaries) USE_SERIAL_PORT.write(0xF0);
    for (byte i=0;i<length;i++) USE_SERIAL_PORT.write(array[i]);
    if (!ArrayContainsBoundaries) USE_SERIAL_PORT.write(0xF7);
#if USE_RUNNING_STATUS
    mRunningStatus_TX = InvalidType;
#endif
}
```

3.1.3.29 void MIDI_Class::sendTimeCodeQuarterFrame (byte *TypeNibble*, byte *ValuesNibble*)

Send a MIDI Time Code Quarter Frame.

See MIDI Specification for more information.

Parameters

TypeNibble	MTC type
ValuesNibble	MTC data

Definition at line 259 of file MIDI.cpp.

```
{
    byte data = ( ((TypeNibble & 0x07) << 4) | (ValuesNibble & 0x0F) );
    sendTimeCodeQuarterFrame(data);
}
```

3.1.3.30 void MIDI_Class::sendTimeCodeQuarterFrame (byte *data*)

Send a MIDI Time Code Quarter Frame.

See MIDI Specification for more information.

Parameters

data	if you want to encode directly the nibbles in your program, you can send the byte here.

Definition at line 271 of file MIDI.cpp.

```
{
    USE_SERIAL_PORT.write((byte)TimeCodeQuarterFrame);
    USE_SERIAL_PORT.write(data);
#if USE_RUNNING_STATUS
    mRunningStatus_TX = InvalidType;
#endif
}
```

3.1.3.31 void MIDI_Class::sendTuneRequest ( )

Send a Tune Request message.

When a MIDI unit receives this message, it should tune its oscillators (if equipped with any)

Definition at line 251 of file MIDI.cpp.

```
{ sendRealTime(TuneRequest); }
```

3.1.3.32 void MIDI_Class::setHandleActiveSensing (void(*)(void) *fptr*)

Definition at line 724 of file MIDI.cpp.

```
{ mActiveSensingCallback = fptr; }
```

3.1.3.33 void MIDI_Class::setHandleAfterTouchChannel (void(*)(byte channel, byte pressure) *fptr*)

Definition at line 713 of file MIDI.cpp.

```
{ mAfterTouchChannelCallback = fptr; }
```

3.1.3.34 void MIDI_Class::setHandleAfterTouchPoly (void(*)(byte channel, byte note, byte pressure) *fptr*)

Definition at line 710 of file MIDI.cpp.

```
{ mAfterTouchPolyCallback = fptr; }
```

3.1.3.35 void MIDI_Class::setHandleClock (void(*)(void) *fptr*)

Definition at line 720 of file MIDI.cpp.

```
{ mClockCallback = fptr; }
```

3.1.3.36 void MIDI_Class::setHandleContinue (void(*)(void) *fptr*)

Definition at line 722 of file MIDI.cpp.

```
{ mContinueCallback = fptr; }
```

3.1.3.37 void MIDI_Class::setHandleControlChange (void(*)(byte channel, byte number, byte value) *fptr*)

Definition at line 711 of file MIDI.cpp.

```
{ mControlChangeCallback = fptr; }
```

3.1.3.38 void MIDI_Class::setHandleNoteOff (void(*)(byte channel, byte note, byte velocity) *fptr*)

Definition at line 708 of file MIDI.cpp.

```
{ mNoteOffCallback = fptr; }
```

3.1.3.39 void MIDI_Class::setHandleNoteOn (void(*)(byte channel, byte note, byte velocity) *fptr*)

Definition at line 709 of file MIDI.cpp.

```
{ mNoteOnCallback = fptr; }
```

3.1.3.40 void MIDI_Class::setHandlePitchBend (void(*)(byte channel, int bend) *fptr*)

Definition at line 714 of file MIDI.cpp.

```
{ mPitchBendCallback = fptr; }
```

3.1.3.41 void MIDI_Class::setHandleProgramChange (void(*)(byte channel, byte number) *fptr*)

Definition at line 712 of file MIDI.cpp.

```
{ mProgramChangeCallback = fptr; }
```

3.1.3.42 void MIDI_Class::setHandleSongPosition (void(*)(unsigned int beats) *fptr*)

Definition at line 717 of file MIDI.cpp.

```
{ mSongPositionCallback = fptr; }
```

3.1.3.43 void MIDI_Class::setHandleSongSelect (void(*)(byte songnumber) *fptr*)

Definition at line 718 of file MIDI.cpp.

```
{ mSongSelectCallback = fptr; }
```

3.1.3.44 void MIDI_Class::setHandleStart (void(*)(void) *fptr*)

Definition at line 721 of file MIDI.cpp.

```
{ mStartCallback = fptr; }
```

3.1.3.45 void MIDI_Class::setHandleStop (void(*)(void) *fptr*)

Definition at line 723 of file MIDI.cpp.

```
{ mStopCallback = fptr; }
```

3.1.3.46 void MIDI_Class::setHandleSystemExclusive (void(*)(byte *array, byte size) *fptr*)

Definition at line 715 of file MIDI.cpp.

```
{ mSystemExclusiveCallback = fptr; }
```

3.1.3.47 void MIDI_Class::setHandleSystemReset (void(*)(void) *fptr*)

Definition at line 725 of file MIDI.cpp.

```
{ mSystemResetCallback = fptr; }
```

3.1.3.48 void MIDI_Class::setHandleTimeCodeQuarterFrame (void(*)(byte data) *fptr*)

Definition at line 716 of file MIDI.cpp.

```
{ mTimeCodeQuarterFrameCallback = fptr; }
```

3.1.3.49 void MIDI_Class::setHandleTuneRequest (void(*)(void) *fptr*)

Definition at line 719 of file MIDI.cpp.

```
{ mTuneRequestCallback = fptr; }
```

3.1.3.50 void MIDI_Class::setInputChannel (const byte *Channel*)

Set the value for the input MIDI channel.

Parameters

Channel	the channel value. Valid values are 1 to 16, MIDI_CHANNEL_OMNI if you want to listen to all channels, and MIDI_CHANNEL_OFF to disable MIDI input.

Definition at line 703 of file MIDI.cpp.

```
{ mInputChannel = Channel; }
```

3.1.3.51 void MIDI_Class::setThruFilterMode (const kThruFilterMode *inThruFilterMode*)

Set the filter for thru mirroring.

Parameters

inThruFilter-Mode	a filter mode

See also

kThruFilterMode

Definition at line 816 of file MIDI.cpp.

```
                                                                    {
    mThruFilterMode = inThruFilterMode;
    if (mThruFilterMode != Off) mThruActivated = true;
    else mThruActivated = false;
}
```

3.1.3.52 void MIDI_Class::turnThruOff ( )

Setter method: turn message mirroring off.

Definition at line 829 of file MIDI.cpp.

```
                            {
    mThruActivated = false;
    mThruFilterMode = Off;
}
```

3.1.3.53 void MIDI_Class::turnThruOn (kThruFilterMode *inThruFilterMode* = `Full`)

Setter method: turn message mirroring on.

Definition at line 824 of file MIDI.cpp.

```
                                                                {
    mThruActivated = true;
    mThruFilterMode = inThruFilterMode;
}
```

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

- /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/MIDI.h
- /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/MIDI.cpp

3.2 midimsg Struct Reference

`#include <MIDI.h>`

Public Attributes

- byte channel
- kMIDIType type
- byte data1
- byte data2
- byte sysex_array [MIDI_SYSEX_ARRAY_SIZE]
- bool valid

3.2.1 Detailed Description

The midimsg structure contains decoded data of a MIDI message read from the serial port with read() or thru().

Definition at line 98 of file MIDI.h.

3.2.2 Member Data Documentation

3.2.2.1 byte midimsg::channel

The MIDI channel on which the message was recieved.

Value goes from 1 to 16.

Definition at line 100 of file MIDI.h.

3.2.2.2 byte midimsg::data1

The first data byte.

Value goes from 0 to 127.

If the message is SysEx, this byte contains the array length.

Definition at line 104 of file MIDI.h.

3.2.2.3 byte midimsg::data2

The second data byte. If the message is only 2 bytes long, this one is null.

Value goes from 0 to 127.

Definition at line 106 of file MIDI.h.

3.2.2.4 byte midimsg::sysex_array[MIDI_SYSEX_ARRAY_SIZE]

System Exclusive dedicated byte array.

Array length is stocked in data1.

Definition at line 108 of file MIDI.h.

3.2.2.5 kMIDIType midimsg::type

The type of the message (see the define section for types reference)

Definition at line 102 of file MIDI.h.

3.2.2.6 bool midimsg::valid

This boolean indicates if the message is valid or not. There is no channel consideration here, validity means the message respects the MIDI norm.

Definition at line 110 of file MIDI.h.

The documentation for this struct was generated from the following file:

- /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/MIDI.h

Chapter 4

File Documentation

4.1 /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/MI File Reference

MIDI Library for the Arduino.

```
#include "MIDI.h"
#include <stdlib.h>
#include "WConstants.h"
#include "HardwareSerial.h"
```

Variables

- MIDI_Class MIDI

  Main instance (the class comes pre-instantiated).

4.1.1 Detailed Description

MIDI Library for the Arduino. Project MIDI Library

Version

3.1

Author

Francois Best

Date

24/02/11 license GPL Forty Seven Effects - 2011

Definition in file MIDI.cpp.

4.1.2 Variable Documentation

4.1.2.1 MIDI_Class MIDI

Main instance (the class comes pre-instantiated).

Definition at line 18 of file MIDI.cpp.

4.2 /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/A File Reference

MIDI Library for the Arduino Version 3.1.

```
#include <inttypes.h>
```

Classes

- struct midimsg
- class MIDI_Class

  The main class for MIDI handling.
  See member descriptions to know how to use it, or check out the examples supplied with the library.

Defines

- #define COMPILE_MIDI_IN 1
- #define COMPILE_MIDI_OUT 1
- #define COMPILE_MIDI_THRU 1
- #define USE_SERIAL_PORT Serial
- #define USE_RUNNING_STATUS 1
- #define USE_CALLBACKS 1
- #define MIDI_BAUDRATE 31250
- #define MIDI_CHANNEL_OMNI 0
- #define MIDI_CHANNEL_OFF 17
- #define MIDI_SYSEX_ARRAY_SIZE 255

Typedefs

- typedef uint8_t byte
- typedef uint16_t word

Enumerations

- enum kMIDIType {

  NoteOff = 0x80, NoteOn = 0x90, AfterTouchPoly = 0xA0, ControlChange = 0xB0,

  ProgramChange = 0xC0, AfterTouchChannel = 0xD0, PitchBend = 0xE0, SystemExclusive = 0xF0,

  TimeCodeQuarterFrame = 0xF1, SongPosition = 0xF2, SongSelect = 0xF3, TuneRequest = 0xF6,

  Clock = 0xF8, Start = 0xFA, Continue = 0xFB, Stop = 0xFC,

  ActiveSensing = 0xFE, SystemReset = 0xFF, InvalidType = 0x00 }
- enum kThruFilterMode { Off = 0, Full = 1, SameChannel = 2, DifferentChannel = 3 }

Variables

- MIDI_Class MIDI

  Main instance (the class comes pre-instantiated).

4.2.1 Detailed Description

MIDI Library for the Arduino Version 3.1. Project MIDI Library

Author

Francois Best

Date

24/02/11 License GPL Forty Seven Effects - 2011

Definition in file MIDI.h.

4.2.2 Define Documentation

4.2.2.1 #define COMPILE_MIDI_IN 1

Definition at line 31 of file MIDI.h.

4.2.2.2 #define COMPILE_MIDI_OUT 1

Definition at line 32 of file MIDI.h.

4.2.2.3 #define COMPILE_MIDI_THRU 1

Definition at line 33 of file MIDI.h.

4.2.2.4 #define MIDI_BAUDRATE 31250

Definition at line 54 of file MIDI.h.

4.2.2.5 #define MIDI_CHANNEL_OFF 17

Definition at line 57 of file MIDI.h.

4.2.2.6 #define MIDI_CHANNEL_OMNI 0

Definition at line 56 of file MIDI.h.

4.2.2.7 #define MIDI_SYSEX_ARRAY_SIZE 255

Definition at line 59 of file MIDI.h.

4.2.2.8 #define USE_CALLBACKS 1

Definition at line 46 of file MIDI.h.

4.2.2.9 #define USE_RUNNING_STATUS 1

Definition at line 41 of file MIDI.h.

4.2.2.10 #define USE_SERIAL_PORT Serial

Definition at line 37 of file MIDI.h.

4.2.3 Typedef Documentation

4.2.3.1 typedef uint8_t byte

Type definition for practical use (because "unsigned char" is a bit long to write..)
Definition at line 62 of file MIDI.h.

4.2.3.2 typedef uint16_t word

Definition at line 63 of file MIDI.h.

4.2.4 Enumeration Type Documentation

4.2.4.1 enum kMIDIType

Enumeration of MIDI types

Enumerator:

NoteOff Note Off.

NoteOn Note On.

AfterTouchPoly Polyphonic AfterTouch.

ControlChange Control Change / Channel Mode.

ProgramChange Program Change.

AfterTouchChannel Channel (monophonic) AfterTouch.

PitchBend Pitch Bend.

SystemExclusive System Exclusive.

TimeCodeQuarterFrame System Common - MIDI Time Code Quarter Frame.

SongPosition System Common - Song Position Pointer.

SongSelect System Common - Song Select.

TuneRequest System Common - Tune Request.

Clock System Real Time - Timing Clock.

Start System Real Time - Start.

Continue System Real Time - Continue.

Stop System Real Time - Stop.

ActiveSensing System Real Time - Active Sensing.

SystemReset System Real Time - System Reset.

InvalidType For notifying errors.

Definition at line 66 of file MIDI.h.

```
            {
    NoteOff               = 0x80,
    NoteOn                = 0x90,
    AfterTouchPoly        = 0xA0,
    ControlChange         = 0xB0,
    ProgramChange         = 0xC0,
    AfterTouchChannel     = 0xD0,
    PitchBend             = 0xE0,
    SystemExclusive       = 0xF0,
    TimeCodeQuarterFrame  = 0xF1,
    SongPosition          = 0xF2,
    SongSelect            = 0xF3,
    TuneRequest           = 0xF6,
    Clock                 = 0xF8,
    Start                 = 0xFA,
    Continue              = 0xFB,
    Stop                  = 0xFC,
    ActiveSensing         = 0xFE,
    SystemReset           = 0xFF,
    InvalidType           = 0x00
};
```

4.2.4.2 enum kThruFilterMode

Enumeration of Thru filter modes

Enumerator:

Off Thru disabled (nothing passes through).

Full Fully enabled Thru (every incoming message is sent back).

SameChannel Only the messages on the Input Channel will be sent back.

DifferentChannel All the messages but the ones on the Input Channel will be sent back.

Definition at line 89 of file MIDI.h.

```
                    {
    Off                     = 0,
    Full                    = 1,
    SameChannel             = 2,
    DifferentChannel        = 3
};
```

4.2.5 Variable Documentation

4.2.5.1 MIDI_Class MIDI

Main instance (the class comes pre-instantiated).

Definition at line 18 of file MIDI.cpp.