

Arduino MIDI Library  
Version 3.0

Generated by Doxygen 1.5.8

Sun Mar 6 15:47:07 2011



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	MIDI_Class Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Constructor & Destructor Documentation . . . . .	6
3.1.2.1	MIDI_Class . . . . .	6
3.1.2.2	~MIDI_Class . . . . .	6
3.1.3	Member Function Documentation . . . . .	6
3.1.3.1	begin . . . . .	6
3.1.3.2	check . . . . .	7
3.1.3.3	getChannel . . . . .	7
3.1.3.4	getData1 . . . . .	7
3.1.3.5	getData2 . . . . .	8
3.1.3.6	getFilterMode . . . . .	8
3.1.3.7	getInputChannel . . . . .	8
3.1.3.8	getSysExArray . . . . .	8
3.1.3.9	getThruState . . . . .	8
3.1.3.10	getType . . . . .	8
3.1.3.11	read . . . . .	9
3.1.3.12	read . . . . .	9
3.1.3.13	sendAfterTouch . . . . .	9
3.1.3.14	sendControlChange . . . . .	9
3.1.3.15	sendNoteOff . . . . .	9
3.1.3.16	sendNoteOn . . . . .	10

3.1.3.17	sendPitchBend . . . . .	10
3.1.3.18	sendPitchBend . . . . .	10
3.1.3.19	sendPolyPressure . . . . .	10
3.1.3.20	sendProgramChange . . . . .	11
3.1.3.21	sendRealTime . . . . .	11
3.1.3.22	sendSongPosition . . . . .	11
3.1.3.23	sendSongSelect . . . . .	12
3.1.3.24	sendSysEx . . . . .	12
3.1.3.25	sendTimeCodeQuarterFrame . . . . .	12
3.1.3.26	sendTimeCodeQuarterFrame . . . . .	13
3.1.3.27	sendTuneRequest . . . . .	13
3.1.3.28	setInputChannel . . . . .	13
3.1.3.29	setThruFilterMode . . . . .	13
3.1.3.30	setThruFilterMode . . . . .	14
3.1.3.31	turnThruOff . . . . .	14
3.1.3.32	turnThruOn . . . . .	14
3.2	midimsg Struct Reference . . . . .	15
3.2.1	Detailed Description . . . . .	15
3.2.2	Member Data Documentation . . . . .	15
3.2.2.1	channel . . . . .	15
3.2.2.2	data1 . . . . .	15
3.2.2.3	data2 . . . . .	15
3.2.2.4	sysex_array . . . . .	15
3.2.2.5	type . . . . .	16
3.2.2.6	valid . . . . .	16
<b>4</b>	<b>File Documentation</b>	<b>17</b>
4.1	/Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/Compatibility_v2.5.h File Reference . . . . .	17
4.1.1	Detailed Description . . . . .	17
4.1.2	Define Documentation . . . . .	18
4.1.2.1	ATCanal . . . . .	18
4.1.2.2	ATPoly . . . . .	18
4.1.2.3	CC . . . . .	18
4.1.2.4	MIDI_FILTER_ANTICANAL . . . . .	18
4.1.2.5	MIDI_FILTER_CANAL . . . . .	18
4.1.2.6	MIDI_FILTER_FULL . . . . .	18

4.1.2.7	MIDI_FILTER_OFF	18
4.1.2.8	MIDI_rate	18
4.1.2.9	PC	18
4.1.2.10	SysEx	19
4.2	/Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/MIDI.cpp File Reference	20
4.2.1	Detailed Description	20
4.2.2	Variable Documentation	20
4.2.2.1	MIDI	20
4.3	/Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/MIDI.h File Reference	21
4.3.1	Detailed Description	22
4.3.2	Define Documentation	22
4.3.2.1	COMPATIBILITY_V25	22
4.3.2.2	COMPFLAG_MIDI_IN	22
4.3.2.3	COMPFLAG_MIDI_OUT	22
4.3.2.4	MIDI_BAUDRATE	22
4.3.2.5	MIDI_CHANNEL_OFF	22
4.3.2.6	MIDI_CHANNEL_OMNI	22
4.3.2.7	MIDI_SYSEX_ARRAY_SIZE	22
4.3.2.8	USE_RUNNING_STATUS	22
4.3.2.9	USE_SERIAL_PORT	23
4.3.3	Typedef Documentation	23
4.3.3.1	byte	23
4.3.4	Enumeration Type Documentation	23
4.3.4.1	kMIDIType	23
4.3.4.2	kThruFilterMode	24
4.3.5	Variable Documentation	24
4.3.5.1	MIDI	24



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">MIDI_Class</a>	5
<a href="#">midimsg</a>	15





# Chapter 2

# File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

/Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/ <a href="#">Compatibility_v2.5.h</a> (Compatibility file for MIDI Library v2.5 Version 3.0 ) . . . . .	17
/Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/ <a href="#">MIDI.cpp</a> (MIDI Library for the Arduino ) . . . . .	20
/Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/ <a href="#">MIDI.h</a> (MIDI Library for the Arduino Version 3.0 ) . . . . .	21



# Chapter 3

## Class Documentation

### 3.1 MIDI\_Class Class Reference

```
#include <MIDI.h>
```

#### Public Member Functions

- [MIDI\\_Class](#) ()
- [~MIDI\\_Class](#) ()
- void [begin](#) (const [byte](#) inChannel=1)
- void [sendNoteOn](#) ([byte](#) NoteNumber, [byte](#) Velocity, [byte](#) Channel)
- void [sendNoteOff](#) ([byte](#) NoteNumber, [byte](#) Velocity, [byte](#) Channel)
- void [sendProgramChange](#) ([byte](#) ProgramNumber, [byte](#) Channel)
- void [sendControlChange](#) ([byte](#) ControlNumber, [byte](#) ControlValue, [byte](#) Channel)
- void [sendPitchBend](#) (unsigned int PitchValue, [byte](#) Channel)
- void [sendPitchBend](#) (double PitchValue, [byte](#) Channel)
- void [sendPolyPressure](#) ([byte](#) NoteNumber, [byte](#) Pressure, [byte](#) Channel)
- void [sendAfterTouch](#) ([byte](#) Pressure, [byte](#) Channel)
- void [sendSysEx](#) ([byte](#) length, [byte](#) \*array, bool ArrayContainsBoundaries=false)
- void [sendTimeCodeQuarterFrame](#) ([byte](#) TypeNibble, [byte](#) ValuesNibble)
- void [sendTimeCodeQuarterFrame](#) ([byte](#) data)
- void [sendSongPosition](#) (unsigned int Beats)
- void [sendSongSelect](#) ([byte](#) SongNumber)
- void [sendTuneRequest](#) ()
- void [sendRealTime](#) ([kMIDIType](#) Type)
- bool [read](#) ()
- bool [read](#) (const [byte](#) Channel)
- [kMIDIType](#) [getType](#) ()
- [byte](#) [getChannel](#) ()
- [byte](#) [getData1](#) ()
- [byte](#) [getData2](#) ()
- [byte](#) \* [getSysExArray](#) ()
- bool [check](#) ()
- [byte](#) [getInputChannel](#) ()
- void [setInputChannel](#) (const [byte](#) Channel)

- [kThruFilterMode](#) [getFilterMode](#) ()
- bool [getThruState](#) ()
- void [turnThruOn](#) ([kThruFilterMode](#) [inThruFilterMode=Full](#))
- void [turnThruOff](#) ()
- void [setThruFilterMode](#) (const [byte](#) [inThruFilterMode](#))
- void [setThruFilterMode](#) (const [kThruFilterMode](#) [inThruFilterMode](#))

### 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.

```
22 { }
```

#### 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 26 of file MIDI.cpp.

```
26 { }
```

### 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 34 of file MIDI.cpp.

```
34                                     {
35
36     // Initialise the Serial port
37     USE_SERIAL_PORT.begin(MIDI_BAUDRATE);
38
39
```

```
40 #if COMPFLAG_MIDI_OUT
41
42 #if USE_RUNNING_STATUS
43     mRunningStatus_TX = InvalidType;
44 #endif // USE_RUNNING_STATUS
45
46 #endif // COMPFLAG_MIDI_OUT
47
48
49 #if COMPFLAG_MIDI_IN
50
51     mInputChannel = inChannel;
52     mRunningStatus_RX = InvalidType;
53     mPendingMessageIndex = 0;
54     mPendingMessageExpectedLenght = 0;
55
56     mMessage.valid = false;
57     mMessage.type = InvalidType;
58     mMessage.channel = 0;
59     mMessage.data1 = 0;
60     mMessage.data2 = 0;
61
62 #endif // COMPFLAG_MIDI_IN
63
64
65 #if (COMPFLAG_MIDI_IN && COMPFLAG_MIDI_OUT) // Thru
66
67     mThruFilterMode = Full;
68
69 #endif // Thru
70
71 }
```

### 3.1.3.2 bool MIDI\_Class::check ()

Check if a valid message is stored in the structure.

Definition at line 571 of file MIDI.cpp.

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

### 3.1.3.3 byte MIDI\_Class::getChannel ()

Getter method: access to the channel of the message stored in the structure.

Definition at line 563 of file MIDI.cpp.

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

### 3.1.3.4 byte MIDI\_Class::getData1 ()

Getter method: access to the first data byte of the message stored in the structure.

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

Definition at line 565 of file MIDI.cpp.

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

### 3.1.3.5 byte MIDI\_Class::getData2 ()

Getter method: access to the second data byte of the message stored in the structure.

Definition at line 567 of file MIDI.cpp.

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

### 3.1.3.6 kThruFilterMode MIDI\_Class::getFilterMode () [inline]

Definition at line 226 of file MIDI.h.

```
226 { return mThruFilterMode; }
```

### 3.1.3.7 byte MIDI\_Class::getInputChannel () [inline]

Definition at line 186 of file MIDI.h.

```
186 { return mInputChannel; }
```

### 3.1.3.8 byte \* MIDI\_Class::getSysExArray ()

Getter method: access to the System Exclusive byte array. Array length is stocked in Data1.

Definition at line 569 of file MIDI.cpp.

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

### 3.1.3.9 bool MIDI\_Class::getThruState () [inline]

Definition at line 227 of file MIDI.h.

```
227 { return mThruActivated; }
```

### 3.1.3.10 kMIDIType MIDI\_Class::getType ()

Getter method: access to the message type stored in the structure.

Returns an enumerated type.

Definition at line 561 of file MIDI.cpp.

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

### 3.1.3.11 bool MIDI\_Class::read (const byte *inChannel*)

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

Definition at line 278 of file MIDI.cpp.

```
278                                     {
279
280         if (inChannel >= MIDI_CHANNEL_OFF) return false; // MIDI Input disabled.
281
282         if (parse(inChannel)) return filter(inChannel);
283         else return false;
284
285 }
```

### 3.1.3.12 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 273 of file MIDI.cpp.

```
273                                     {
274         return read(mInputChannel);
275 }
```

### 3.1.3.13 void MIDI\_Class::sendAfterTouch (byte *Pressure*, byte *Channel*)

Monophonic AfterTouch

Definition at line 165 of file MIDI.cpp.

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

### 3.1.3.14 void MIDI\_Class::sendControlChange (byte *ControlNumber*, byte *ControlValue*, byte *Channel*)

Send a Control Change message

Definition at line 161 of file MIDI.cpp.

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

### 3.1.3.15 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)

Definition at line 157 of file MIDI.cpp.

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

### 3.1.3.16 void MIDI\_Class::sendNoteOn (byte *NoteNumber*, byte *Velocity*, byte *Channel*)

Send a Note On message

Definition at line 155 of file MIDI.cpp.

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

### 3.1.3.17 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 (maximum downwards bend) and +1 (max upwards bend), center value is 0.

***Channel*** The channel to send the message on (1 to 16).

Definition at line 179 of file MIDI.cpp.

```
179                                     {
180
181     unsigned int pitchval = (PitchValue+1.f)*8192;
182     if (pitchval > 16383) pitchval = 16383;      // overflow protection
183     sendPitchBend(pitchval,Channel);
184
185 }
```

### 3.1.3.18 void MIDI\_Class::sendPitchBend (unsigned int *PitchValue*, byte *Channel*)

Send a Pitch Bend message using an integer value.

#### Parameters:

***PitchValue*** The amount of bend to send (in an integer format), between 0 (maximum downwards bend) and 16383 (max upwards bend), center value is 8192.

***Channel*** The channel to send the message on (1 to 16).

Definition at line 170 of file MIDI.cpp.

```
170                                     {
171
172     send(PitchBend, (PitchValue & 0x7F), (PitchValue >> 7) & 0x7F,Channel);
173
174 }
```

### 3.1.3.19 void MIDI\_Class::sendPolyPressure (byte *NoteNumber*, byte *Pressure*, byte *Channel*)

Polyphonic AfterTouch (carries the information of pressure of the given key/note)

Definition at line 163 of file MIDI.cpp.

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



### 3.1.3.20 void MIDI\_Class::sendProgramChange (byte *ProgramNumber*, byte *Channel*)

Send a Program Change message

Definition at line 159 of file MIDI.cpp.

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

### 3.1.3.21 void MIDI\_Class::sendRealTime (kMIDIType *Type*)

Send a Real Time (one byte) message.

You can also send a Tune Request with this method.

#### Parameters:

**Type** The available Real Time types are: Start, Stop, Continue, Clock, ActiveSensing and SystemReset.

Definition at line 245 of file MIDI.cpp.

```
245                                     {
246     switch (Type) {
247         case TuneRequest: // Not really real-time, but one byte anyway.
248         case Clock:
249         case Start:
250         case Stop:
251         case Continue:
252         case ActiveSensing:
253         case SystemReset:
254             USE_SERIAL_PORT.write((byte)Type);
255             break;
256         default:
257             // Invalid Real Time marker
258             break;
259     }
260 }
```

### 3.1.3.22 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 226 of file MIDI.cpp.

```
226                                     {
227
228     USE_SERIAL_PORT.write(SongPosition);
229     USE_SERIAL_PORT.write(Beats & 0x7F);
230     USE_SERIAL_PORT.write((Beats >> 7) & 0x7F);
231
232 }
```

### 3.1.3.23 void MIDI\_Class::sendSongSelect (byte *SongNumber*)

Send a Song Select message

Definition at line 235 of file MIDI.cpp.

```
235                                     {
236
237     USE_SERIAL_PORT.write(SongSelect);
238     USE_SERIAL_PORT.write(SongNumber & 0x7F);
239
240 }
```

### 3.1.3.24 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 193 of file MIDI.cpp.

```
193                                     {
194     if (!ArrayContainsBoundaries) USE_SERIAL_PORT.write(0xF0);
195     for (byte i=0;i<length;i++) USE_SERIAL_PORT.write(array[i]);
196     if (!ArrayContainsBoundaries) USE_SERIAL_PORT.write(0xF7);
197 }
```

### 3.1.3.25 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 216 of file MIDI.cpp.

```
216                                     {
217
218     USE_SERIAL_PORT.write(TimeCodeQuarterFrame);
219     USE_SERIAL_PORT.write(data);
220
221 }
```

**3.1.3.26 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 206 of file MIDI.cpp.

```

206                                                                 {
207
208         byte data = ( ((TypeNibble & 0x07) << 4) | (ValuesNibble & 0x0F) );
209         sendTimeCodeQuarterFrame(data);
210
211     }
```

**3.1.3.27 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 200 of file MIDI.cpp.

```

200 { sendRealTime(TuneRequest); }
```

**3.1.3.28 void MIDI\_Class::setInputChannel (const byte *inChannel*)**

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 578 of file MIDI.cpp.

```

578 { mInputChannel = inChannel; }
```

**3.1.3.29 void MIDI\_Class::setThruFilterMode (const kThruFilterMode *inThruFilterMode*)**

Set the filter for thru mirroring

**Parameters:**

*inThruFilterMode* a filter mode See kThruFilterMode for detailed description.

Definition at line 592 of file MIDI.cpp.

```

592                                                                 {
593         mThruFilterMode = inThruFilterMode;
594         if (mThruFilterMode != Off) mThruActivated = true;
595         else mThruActivated = false;
596     }
```

### 3.1.3.30 void MIDI\_Class::setThruFilterMode (const byte *inThruFilterMode*)

Set the filter for thru mirroring

#### Parameters:

***inThruFilterMode*** a filter mode See kThruFilterMode for detailed description.

This method uses a byte parameter and is for compatibility only, please use kThruFilterMode for future programs.

Definition at line 602 of file MIDI.cpp.

```
602                                     {
603     mThruFilterMode = (kThruFilterMode)inThruFilterMode;
604     if (mThruFilterMode != Off) mThruActivated = true;
605     else mThruActivated = false;
606 }
```

### 3.1.3.31 void MIDI\_Class::turnThruOff ()

Setter method: turn message mirroring off.

Definition at line 615 of file MIDI.cpp.

```
615                                     {
616     mThruActivated = false;
617     mThruFilterMode = Off;
618 }
```

### 3.1.3.32 void MIDI\_Class::turnThruOn (kThruFilterMode *inThruFilterMode* = Full)

Setter method: turn message mirroring on.

Definition at line 610 of file MIDI.cpp.

```
610                                     {
611     mThruActivated = true;
612     mThruFilterMode = inThruFilterMode;
613 }
```

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 100 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 102 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 106 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 108 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 110 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 104 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 112 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/MIDILibrary\_v2.5.h File Reference

Compatibility file for MIDI Library v2.5 Version 3.0.

#### Defines

- #define [MIDI\\_rate](#) MIDI\_BAUDRATE
- #define [ATPoly](#) AfterTouchPoly
- #define [CC](#) ControlChange
- #define [PC](#) ProgramChange
- #define [ATCanal](#) AfterTouchChannel
- #define [SysEx](#) SystemExclusive
- #define [MIDI\\_FILTER\\_OFF](#) 0
- #define [MIDI\\_FILTER\\_FULL](#) 1
- #define [MIDI\\_FILTER\\_CANAL](#) 2
- #define [MIDI\\_FILTER\\_ANTICANAL](#) 3

#### 4.1.1 Detailed Description

Compatibility file for MIDI Library v2.5 Version 3.0.

Project MIDI Library

**Author:**

François Best

**Date:**

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

Definition in file [Compatibility\\_v2.5.h](#).

## 4.1.2 Define Documentation

### 4.1.2.1 **#define ATCanal AfterTouchChannel**

Message type AfterTouch Channel

Definition at line 30 of file Compatibility\_v2.5.h.

### 4.1.2.2 **#define ATPoly AfterTouchPoly**

Message type AfterTouch Poly

Definition at line 24 of file Compatibility\_v2.5.h.

### 4.1.2.3 **#define CC ControlChange**

Message type Control Change

Definition at line 26 of file Compatibility\_v2.5.h.

### 4.1.2.4 **#define MIDI\_FILTER\_ANTICANAL 3**

Definition at line 38 of file Compatibility\_v2.5.h.

### 4.1.2.5 **#define MIDI\_FILTER\_CANAL 2**

Definition at line 37 of file Compatibility\_v2.5.h.

### 4.1.2.6 **#define MIDI\_FILTER\_FULL 1**

Definition at line 36 of file Compatibility\_v2.5.h.

### 4.1.2.7 **#define MIDI\_FILTER\_OFF 0**

Definition at line 35 of file Compatibility\_v2.5.h.

### 4.1.2.8 **#define MIDI\_rate MIDI\_BAUDRATE**

The basic baudrate for MIDI communications.

Definition at line 22 of file Compatibility\_v2.5.h.

### 4.1.2.9 **#define PC ProgramChange**

Message type Program Change

Definition at line 28 of file Compatibility\_v2.5.h.



#### 4.1

/Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/Compatibility\_v2.5.h File

**Reference** `#define SysEx SystemExclusive`

**19**

Message type System Exclusive

Definition at line 32 of file Compatibility\_v2.5.h.

## 4.2 /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILibrary File Reference

MIDI Library for the Arduino.

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

### Variables

- [MIDI\\_Class MIDI](#)

#### 4.2.1 Detailed Description

MIDI Library for the Arduino.

Project MIDI Library

##### Version:

3.0

##### Author:

François Best

##### Date:

24/02/11 GPL Forty Seven Effects - 2011

Definition in file [MIDI.cpp](#).

#### 4.2.2 Variable Documentation

##### 4.2.2.1 MIDI\_Class MIDI

Main instance (the class comes pre-instantiated).

Definition at line 18 of file MIDI.cpp.

## File Reference

MIDI Library for the Arduino Version 3.0.

```
#include <inttypes.h>
#include "Compatibility_v2.5.h"
```

## Classes

- struct [midimsg](#)
- class [MIDI\\_Class](#)

## Defines

- #define [COMPATIBILITY\\_V25](#) 1
- #define [COMPFLAG\\_MIDI\\_IN](#) 1
- #define [COMPFLAG\\_MIDI\\_OUT](#) 1
- #define [USE\\_SERIAL\\_PORT](#) Serial1
- #define [USE\\_RUNNING\\_STATUS](#) 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](#)

## 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](#)

### 4.3.1 Detailed Description

MIDI Library for the Arduino Version 3.0.

Project MIDI Library

**Author:**

François Best

**Date:**

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

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

### 4.3.2 Define Documentation

#### 4.3.2.1 `#define COMPATIBILITY_V25 1`

Definition at line 30 of file MIDI.h.

#### 4.3.2.2 `#define COMPFLAG_MIDI_IN 1`

Definition at line 33 of file MIDI.h.

#### 4.3.2.3 `#define COMPFLAG_MIDI_OUT 1`

Definition at line 34 of file MIDI.h.

#### 4.3.2.4 `#define MIDI_BAUDRATE 31250`

Definition at line 56 of file MIDI.h.

#### 4.3.2.5 `#define MIDI_CHANNEL_OFF 17`

Definition at line 60 of file MIDI.h.

#### 4.3.2.6 `#define MIDI_CHANNEL_OMNI 0`

Definition at line 59 of file MIDI.h.

#### 4.3.2.7 `#define MIDI_SYSEX_ARRAY_SIZE 255`

Definition at line 62 of file MIDI.h.

#### 4.3.2.8 `#define USE_RUNNING_STATUS 1`

Definition at line 44 of file MIDI.h.

## 4.3

/Users/franky/Documents/Dropbox/SVN/embedded/toolbox/libraries/MIDILib/trunk/Arduino/MIDI.h  
File Reference 23

### 4.3.2.9 #define USE\_SERIAL\_PORT Serial1

---

Definition at line 40 of file MIDI.h.

## 4.3.3 Typedef Documentation

### 4.3.3.1 typedef uint8\_t byte

Type definition for practical use (because "unsigned char" is a bit long to write.. )

Definition at line 65 of file MIDI.h.

## 4.3.4 Enumeration Type Documentation

### 4.3.4.1 enum kMIDIType

Enumeration of MIDI types

Enumerator:

*NoteOff*  
*NoteOn*  
*AfterTouchPoly*  
*ControlChange*  
*ProgramChange*  
*AfterTouchChannel*  
*PitchBend*  
*SystemExclusive*  
*TimeCodeQuarterFrame*  
*SongPosition*  
*SongSelect*  
*TuneRequest*  
*Clock*  
*Start*  
*Continue*  
*Stop*  
*ActiveSensing*  
*SystemReset*  
*InvalidType*

Definition at line 68 of file MIDI.h.

```
68      {  
69      NoteOff          = 0x80,    // Note Off  
70      NoteOn          = 0x90,    // Note On  
71      AfterTouchPoly  = 0xA0,    // Polyphonic AfterTouch  
72      ControlChange   = 0xB0,    // Control Change / Channel Mode  
73      ProgramChange   = 0xC0,    // Program Change
```

```

74      AfterTouchChannel    = 0xD0,    // Channel (monophonic) AfterTouch
75      PitchBend            = 0xE0,    // Pitch Bend
76      SystemExclusive      = 0xF0,    // System Exclusive
77      TimeCodeQuarterFrame = 0xF1,    // System Common - MIDI Time Code Quarter Frame
78      SongPosition         = 0xF2,    // System Common - Song Position Pointer
79      SongSelect           = 0xF3,    // System Common - Song Select
80      TuneRequest          = 0xF6,    // System Common - Tune Request
81      Clock                = 0xF8,    // System Real Time - Timing Clock
82      Start                = 0xFA,    // System Real Time - Start
83      Continue             = 0xFB,    // System Real Time - Continue
84      Stop                 = 0xFC,    // System Real Time - Stop
85      ActiveSensing        = 0xFE,    // System Real Time - Active Sensing
86      SystemReset          = 0xFF,    // System Real Time - System Reset
87      InvalidType          = 0x00     // For notifying errors
88  };

```

#### 4.3.4.2 enum kThruFilterMode

Enumeration of Thru filter modes

**Enumerator:**

*Off*

*Full*

*SameChannel*

*DifferentChannel*

Definition at line 91 of file MIDI.h.

```

91      {
92      Off                = 0,    // Thru disabled (nothing passes through).
93      Full              = 1,    // Fully enabled Thru (every incoming message is sent back).
94      SameChannel       = 2,    // Only the messages on the Input Channel will be sent back.
95      DifferentChannel  = 3     // All the messages but the ones on the Input Channel will be sent
96  };

```

### 4.3.5 Variable Documentation

#### 4.3.5.1 MIDI\_Class MIDI

Main instance (the class comes pre-instantiated).

Definition at line 18 of file MIDI.cpp.

# Index

- ~MIDI\_Class
  - MIDI\_Class, [6](#)
- /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/MIDILib/3.0/MIDILib/trunk/Arduino/Compatibility\_v2.5.h, [17](#)
- /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/MIDILib/3.0/MIDILib/trunk/Arduino/MIDI.cpp, [20](#)
- /Users/franky/Documents/Dropbox/SVN/embedded/toolbox/MIDILib/3.0/MIDILib/trunk/Arduino/MIDI.h, [21](#)
- ActiveSensing
  - MIDI.h, [23](#)
- AfterTouchChannel
  - MIDI.h, [23](#)
- AfterTouchPoly
  - MIDI.h, [23](#)
- ATCanal
  - Compatibility\_v2.5.h, [18](#)
- ATPoly
  - Compatibility\_v2.5.h, [18](#)
- begin
  - MIDI\_Class, [6](#)
- byte
  - MIDI.h, [23](#)
- CC
  - Compatibility\_v2.5.h, [18](#)
- channel
  - midimsg, [15](#)
- check
  - MIDI\_Class, [7](#)
- Clock
  - MIDI.h, [23](#)
- Compatibility\_v2.5.h
  - ATCanal, [18](#)
  - ATPoly, [18](#)
  - CC, [18](#)
  - MIDI\_FILTER\_ANTICANAL, [18](#)
  - MIDI\_FILTER\_CANAL, [18](#)
  - MIDI\_FILTER\_FULL, [18](#)
  - MIDI\_FILTER\_OFF, [18](#)
  - MIDI\_rate, [18](#)
  - PC, [18](#)
  - SysEx, [18](#)
- COMPATIBILITY\_V25
  - MIDI.h, [22](#)
- COMPFLAG\_MIDI\_IN
  - MIDI.h, [22](#)
- COMPFLAG\_MIDI\_OUT
  - MIDI.h, [22](#)
- Continue
  - MIDI.h, [22](#)
- ControlChange
  - MIDI.h, [23](#)
- data1
  - midimsg, [15](#)
- data2
  - midimsg, [15](#)
- DifferentChannel
  - MIDI.h, [24](#)
- Full
  - MIDI.h, [24](#)
- getChannel
  - MIDI\_Class, [7](#)
- getData1
  - MIDI\_Class, [7](#)
- getData2
  - MIDI\_Class, [7](#)
- getFilterMode
  - MIDI\_Class, [8](#)
- getInputChannel
  - MIDI\_Class, [8](#)
- getSysExArray
  - MIDI\_Class, [8](#)
- getThruState
  - MIDI\_Class, [8](#)
- getType
  - MIDI\_Class, [8](#)
- InvalidType
  - MIDI.h, [23](#)
- kMIDIType
  - MIDI.h, [23](#)
- kThruFilterMode
  - MIDI.h, [24](#)
- MIDI

- MIDI.cpp, 20
- MIDI.h, 24
- MIDI.cpp
  - MIDI, 20
- MIDI.h
  - ActiveSensing, 23
  - AfterTouchChannel, 23
  - AfterTouchPoly, 23
  - byte, 23
  - Clock, 23
  - COMPATIBILITY\_V25, 22
  - COMPFLAG\_MIDI\_IN, 22
  - COMPFLAG\_MIDI\_OUT, 22
  - Continue, 23
  - ControlChange, 23
  - DifferentChannel, 24
  - Full, 24
  - InvalidType, 23
  - kMIDIType, 23
  - kThruFilterMode, 24
  - MIDI, 24
  - MIDI\_BAUDRATE, 22
  - MIDI\_CHANNEL\_OFF, 22
  - MIDI\_CHANNEL\_OMNI, 22
  - MIDI\_SYSEX\_ARRAY\_SIZE, 22
  - NoteOff, 23
  - NoteOn, 23
  - Off, 24
  - PitchBend, 23
  - ProgramChange, 23
  - SameChannel, 24
  - SongPosition, 23
  - SongSelect, 23
  - Start, 23
  - Stop, 23
  - SystemExclusive, 23
  - SystemReset, 23
  - TimeCodeQuarterFrame, 23
  - TuneRequest, 23
  - USE\_RUNNING\_STATUS, 22
  - USE\_SERIAL\_PORT, 22
- MIDI\_BAUDRATE
  - MIDI.h, 22
- MIDI\_CHANNEL\_OFF
  - MIDI.h, 22
- MIDI\_CHANNEL\_OMNI
  - MIDI.h, 22
- MIDI\_Class, 5
  - ~MIDI\_Class, 6
  - begin, 6
  - check, 7
  - getChannel, 7
  - getData1, 7
  - getData2, 7
  - getFilterMode, 8
  - getInputChannel, 8
  - getSysExArray, 8
  - getThruState, 8
  - getType, 8
  - MIDI\_Class, 6
  - MIDI\_Class, 6
  - read, 8, 9
  - sendAfterTouch, 9
  - sendControlChange, 9
  - sendNoteOff, 9
  - sendNoteOn, 9
  - sendPitchBend, 10
  - sendPolyPressure, 10
  - sendProgramChange, 10
  - sendRealTime, 11
  - sendSongPosition, 11
  - sendSongSelect, 11
  - sendSysEx, 12
  - sendTimeCodeQuarterFrame, 12
  - sendTuneRequest, 13
  - setInputChannel, 13
  - setThruFilterMode, 13
  - turnThruOff, 14
  - turnThruOn, 14
- MIDI\_FILTER\_ANTICANAL
  - Compatibility\_v2.5.h, 18
- MIDI\_FILTER\_CANAL
  - Compatibility\_v2.5.h, 18
- MIDI\_FILTER\_FULL
  - Compatibility\_v2.5.h, 18
- MIDI\_FILTER\_OFF
  - Compatibility\_v2.5.h, 18
- MIDI\_rate
  - Compatibility\_v2.5.h, 18
- MIDI\_SYSEX\_ARRAY\_SIZE
  - MIDI.h, 22
- midimsg, 15
  - channel, 15
  - data1, 15
  - data2, 15
  - sysex\_array, 15
  - type, 16
  - valid, 16
- NoteOff
  - MIDI.h, 23
- NoteOn
  - MIDI.h, 23
- Off
  - MIDI.h, 24
- PC



Compatibility\_v2.5.h, [18](#)

PitchBend  
MIDI.h, [23](#)

ProgramChange  
MIDI.h, [23](#)

read  
MIDI\_Class, [8](#), [9](#)

SameChannel  
MIDI.h, [24](#)

sendAfterTouch  
MIDI\_Class, [9](#)

sendControlChange  
MIDI\_Class, [9](#)

sendNoteOff  
MIDI\_Class, [9](#)

sendNoteOn  
MIDI\_Class, [9](#)

sendPitchBend  
MIDI\_Class, [10](#)

sendPolyPressure  
MIDI\_Class, [10](#)

sendProgramChange  
MIDI\_Class, [10](#)

sendRealTime  
MIDI\_Class, [11](#)

sendSongPosition  
MIDI\_Class, [11](#)

sendSongSelect  
MIDI\_Class, [11](#)

sendSysEx  
MIDI\_Class, [12](#)

sendTimeCodeQuarterFrame  
MIDI\_Class, [12](#)

sendTuneRequest  
MIDI\_Class, [13](#)

setInputChannel  
MIDI\_Class, [13](#)

setThruFilterMode  
MIDI\_Class, [13](#)

SongPosition  
MIDI.h, [23](#)

SongSelect  
MIDI.h, [23](#)

Start  
MIDI.h, [23](#)

Stop  
MIDI.h, [23](#)

SysEx  
Compatibility\_v2.5.h, [18](#)

sysex\_array  
midimsg, [15](#)

SystemExclusive  
MIDI.h, [23](#)

SystemReset  
MIDI.h, [23](#)

TimeCodeQuarterFrame  
MIDI.h, [23](#)

TuneRequest  
MIDI.h, [23](#)

turnThruOff  
MIDI\_Class, [14](#)

turnThruOn  
MIDI\_Class, [14](#)

type  
midimsg, [16](#)

USE\_RUNNING\_STATUS  
MIDI.h, [22](#)

USE\_SERIAL\_PORT  
MIDI.h, [22](#)

valid  
midimsg, [16](#)