

MIDlcvt Developer's Reference Manual

0.2

Generated by Doxygen 1.8.7

Sat Jun 21 2014 21:41:47

Contents

1	midicvt	1
1.1	Introduction	1
2	The libmidifilex Library	3
2.1	Introduction to the MIDI File Library Module	3
3	The libmidipp Library	5
3.1	Introduction to the MIDicvt Library Module	5
4	Licenses, MIDicvt Projects.	7
4.1	License Terms for the midicvt project.	7
4.2	XPC Application License	7
4.3	XPC Library License	8
4.4	XPC Documentation License	8
4.5	XPC Affero License	8
4.6	XPC License Summary	9
5	Todo List	11
6	Data Structure Index	13
6.1	Data Structures	13
7	File Index	15
7.1	File List	15
8	Data Structure Documentation	17
8.1	midipp::csvarray Class Reference	17
8.1.1	Detailed Description	20
8.1.2	Member Typedef Documentation	20
8.1.2.1	Fields	20
8.1.3	Constructor & Destructor Documentation	20
8.1.3.1	csvarray	20
8.1.3.2	csvarray	20
8.1.3.3	csvarray	20

8.1.3.4	<code>~csvarray</code>	21
8.1.4	Member Function Documentation	21
8.1.4.1	<code>empty</code>	21
8.1.4.2	<code>is_comment</code>	21
8.1.4.3	<code>operator=</code>	21
8.1.4.4	<code>readfile</code>	21
8.1.4.5	<code>size</code>	22
8.1.5	Field Documentation	22
8.1.5.1	<code>m_csv_lines</code>	22
8.1.5.2	<code>m_is_valid</code>	22
8.1.5.3	<code>m_separator</code>	22
8.2	<code>midipp::initree</code> Class Reference	23
8.2.1	Detailed Description	25
8.2.2	Member Typedef Documentation	25
8.2.2.1	<code>const_iterator</code>	25
8.2.2.2	<code>Container</code>	25
8.2.2.3	<code>iterator</code>	26
8.2.2.4	<code>Section</code>	26
8.2.3	Constructor & Destructor Documentation	26
8.2.3.1	<code>initree</code>	26
8.2.3.2	<code>initree</code>	26
8.2.3.3	<code>initree</code>	26
8.2.3.4	<code>~initree</code>	26
8.2.4	Member Function Documentation	27
8.2.4.1	<code>begin</code>	27
8.2.4.2	<code>begin</code>	27
8.2.4.3	<code>empty</code>	27
8.2.4.4	<code>end</code>	27
8.2.4.5	<code>end</code>	28
8.2.4.6	<code>find</code>	28
8.2.4.7	<code>find</code>	28
8.2.4.8	<code>insert</code>	28
8.2.4.9	<code>is_comment</code>	29
8.2.4.10	<code>make_section</code>	29
8.2.4.11	<code>operator=</code>	29
8.2.4.12	<code>process_option</code>	29
8.2.4.13	<code>process_section_name</code>	30
8.2.4.14	<code>readfile</code>	30
8.2.4.15	<code>section</code>	31
8.2.4.16	<code>section</code>	31

8.2.4.17	size	32
8.2.5	Field Documentation	32
8.2.5.1	m_sections	32
8.2.5.2	sm_dummy_section	32
8.3	midipp::midimapper Class Reference	32
8.3.1	Detailed Description	35
8.3.2	Constructor & Destructor Documentation	36
8.3.2.1	midimapper	36
8.3.2.2	midimapper	36
8.3.3	Member Function Documentation	36
8.3.3.1	active	36
8.3.3.2	active	36
8.3.3.3	drum_map	37
8.3.3.4	read_channel_section	37
8.3.3.5	read_maps	37
8.3.3.6	read_unnamed_section	38
8.3.3.7	rechanel	38
8.3.3.8	repatch	39
8.3.3.9	repitch	39
8.3.4	Field Documentation	39
8.3.4.1	m_channel_map	39
8.3.4.2	m_device_channel	39
8.3.4.3	m_drum_map	40
8.3.4.4	m_file_style	40
8.3.4.5	m_filter_channel	40
8.3.4.6	m_gm_channel	40
8.3.4.7	m_map_reversed	41
8.3.4.8	m_map_type	41
8.3.4.9	m_patch_map	41
8.3.4.10	m_record_count	41
8.3.4.11	m_setup_name	41
8.4	midipp::stringmap< VALUETYPE > Class Template Reference	42
8.4.1	Detailed Description	44
8.4.2	Member Typedef Documentation	44
8.4.2.1	Container	44
8.4.3	Constructor & Destructor Documentation	44
8.4.3.1	stringmap	44
8.4.3.2	~stringmap	45
8.4.4	Member Function Documentation	45
8.4.4.1	begin	45

8.4.4.2	begin	45
8.4.4.3	empty	45
8.4.4.4	end	46
8.4.4.5	end	46
8.4.4.6	find	46
8.4.4.7	find	46
8.4.4.8	insert	46
8.4.4.9	insert	47
8.4.4.10	operator=	47
8.4.4.11	size	47
8.4.4.12	value	48
8.4.5	Field Documentation	48
8.4.5.1	m_Name	48
9	File Documentation	49
9.1	csvarray.cpp File Reference	49
9.1.1	Detailed Description	50
9.1.2	Macro Definition Documentation	50
9.1.2.1	TOKEN_SPACES	50
9.1.3	Function Documentation	50
9.1.3.1	show	50
9.2	csvarray.hpp File Reference	51
9.2.1	Detailed Description	52
9.2.2	Function Documentation	52
9.2.2.1	show	52
9.3	ininames.hpp File Reference	52
9.3.1	Detailed Description	54
9.4	initree.cpp File Reference	55
9.4.1	Detailed Description	56
9.4.2	Function Documentation	57
9.4.2.1	show	57
9.5	initree.hpp File Reference	57
9.5.1	Detailed Description	58
9.5.2	Function Documentation	59
9.5.2.1	show	59
9.6	iniwriting.cpp File Reference	59
9.6.1	Detailed Description	60
9.7	iniwriting.hpp File Reference	60
9.7.1	Detailed Description	61
9.8	mainpage-reference.dox File Reference	62

9.8.1 Detailed Description	62
9.9 midi_functions.dox File Reference	62
9.9.1 Detailed Description	62
9.10 midicvt_base.c File Reference	62
9.10.1 Detailed Description	65
9.10.2 Function Documentation	67
9.10.2.1 checkchan	67
9.10.2.2 error	67
9.10.2.3 filegetc	68
9.10.2.4 get16val	68
9.10.2.5 getbyte	68
9.10.2.6 getint	68
9.10.2.7 midicvt_compile	68
9.10.2.8 midicvt_setup_compile	69
9.10.2.9 midicvt_setup_mfread	69
9.10.2.10 my_arbitrary	69
9.10.2.11 my_chanpressure	69
9.10.2.12 my_error	70
9.10.2.13 my_header	70
9.10.2.14 my_keysig	71
9.10.2.15 my_meot	71
9.10.2.16 my_mmisc	72
9.10.2.17 my_mseq	72
9.10.2.18 my_mspecial	73
9.10.2.19 my_mtext	73
9.10.2.20 my_noff	74
9.10.2.21 my_non	74
9.10.2.22 my_parameter	75
9.10.2.23 my_pitchbend	75
9.10.2.24 my_pressure	76
9.10.2.25 my_program	76
9.10.2.26 my_smpte	77
9.10.2.27 my_sysex	77
9.10.2.28 my_tempo	78
9.10.2.29 my_timesig	78
9.10.2.30 my_trend	79
9.10.2.31 my_trstart	79
9.10.2.32 my_writetrack	79
9.10.2.33 prhex	80
9.10.2.34 prnote	80

9.10.2.35	prs_error	80
9.10.2.36	prtext	80
9.10.2.37	ptime	81
9.10.2.38	redirect_stdout	81
9.11	midicvt_base.h File Reference	81
9.11.1	Detailed Description	83
9.11.2	Function Documentation	83
9.11.2.1	error	83
9.11.2.2	midicvt_compile	83
9.11.2.3	midicvt_setup_compile	84
9.11.2.4	midicvt_setup_mfread	84
9.12	midicvt_globals.c File Reference	84
9.12.1	Detailed Description	85
9.12.2	Function Documentation	86
9.12.2.1	midicvt_option_compile	86
9.12.2.2	midicvt_option_m2m	86
9.12.3	Variable Documentation	86
9.12.3.1	yyval	86
9.13	midicvt_globals.h File Reference	86
9.13.1	Detailed Description	88
9.13.2	Function Documentation	88
9.13.2.1	midicvt_option_compile	88
9.13.2.2	midicvt_option_m2m	89
9.13.3	Variable Documentation	89
9.13.3.1	yyval	89
9.14	midicvt_helpers.c File Reference	89
9.14.1	Detailed Description	90
9.14.2	Macro Definition Documentation	91
9.14.2.1	MIDICVT_PATH_MAX	91
9.14.3	Function Documentation	91
9.14.3.1	check_option	91
9.14.3.2	midi_file_offset	91
9.14.3.3	midicvt_help	92
9.14.3.4	midicvt_parse	93
9.14.3.5	midicvt_set_input_file	93
9.14.3.6	midicvt_set_output_file	93
9.14.3.7	midicvt_version	93
9.14.3.8	report	94
9.14.4	Variable Documentation	94
9.14.4.1	gs_have_input_file	94

9.14.4.2	gs_have_output_file	94
9.14.4.3	gs_help_usage_1	94
9.14.4.4	gs_help_version	94
9.14.4.5	gs_input_file	95
9.14.4.6	gs_output_file	95
9.15	midicvt_helpers.h File Reference	95
9.15.1	Detailed Description	96
9.15.2	Function Documentation	97
9.15.2.1	check_option	97
9.15.2.2	midi_file_offset	97
9.15.2.3	midicvt_help	97
9.15.2.4	midicvt_parse	97
9.15.2.5	midicvt_set_input_file	97
9.15.2.6	midicvt_set_output_file	98
9.15.2.7	midicvt_version	98
9.16	midicvt_license.dox File Reference	98
9.16.1	Detailed Description	98
9.17	midicvt_m2m.c File Reference	98
9.17.1	Detailed Description	100
9.17.2	Function Documentation	101
9.17.2.1	filegetc	101
9.17.2.2	fileputc	101
9.17.2.3	m2m_arbitrary	101
9.17.2.4	m2m_chanpressure	102
9.17.2.5	m2m_error	102
9.17.2.6	m2m_header	102
9.17.2.7	m2m_keysig	103
9.17.2.8	m2m_meot	104
9.17.2.9	m2m_mmisc	104
9.17.2.10	m2m_mseq	105
9.17.2.11	m2m_mspecial	105
9.17.2.12	m2m_mtext	106
9.17.2.13	m2m_noff	106
9.17.2.14	m2m_non	107
9.17.2.15	m2m_parameter	107
9.17.2.16	m2m_pitchbend	108
9.17.2.17	m2m_pressure	108
9.17.2.18	m2m_program	109
9.17.2.19	m2m_smppte	109
9.17.2.20	m2m_sysex	110

9.17.2.21	m2m_tempo	110
9.17.2.22	m2m_timesig	111
9.17.2.23	m2m_trend	111
9.17.2.24	m2m_trstart	111
9.17.2.25	midicvt_initfuncs_m2m	112
9.18	midicvt_m2m.h File Reference	112
9.18.1	Detailed Description	113
9.18.2	Function Documentation	114
9.18.2.1	m2m_chanpressure	114
9.18.2.2	m2m_noff	114
9.18.2.3	m2m_non	115
9.18.2.4	m2m_parameter	115
9.18.2.5	m2m_pitchbend	116
9.18.2.6	m2m_pressure	116
9.18.2.7	m2m_program	117
9.18.2.8	midicvt_initfuncs_m2m	117
9.19	midicvt_macros.h File Reference	117
9.19.1	Detailed Description	119
9.19.2	Macro Definition Documentation	120
9.19.2.1	MIDI_NOTE_MIN	120
9.19.2.2	not_nullptr	120
9.19.2.3	nullptr	120
9.20	midicvt_main.c File Reference	120
9.20.1	Detailed Description	121
9.20.2	Function Documentation	121
9.20.2.1	main	121
9.21	midicvtp_main.cpp File Reference	122
9.21.1	Detailed Description	123
9.21.2	Function Documentation	124
9.21.2.1	main	124
9.21.2.2	midicvtp_help	125
9.21.2.3	midicvtp_parse	125
9.21.3	Variable Documentation	125
9.21.3.1	s_help_version	125
9.21.3.2	s_ini_in_filename	125
9.21.3.3	s_mapping_name	126
9.21.3.4	s_write_csv_drum	126
9.21.3.5	s_write_csv_patch	126
9.22	midifilex.c File Reference	126
9.22.1	Detailed Description	129

9.22.2	Function Documentation	130
9.22.2.1	badbyte	130
9.22.2.2	biggermsg	130
9.22.2.3	chanmessage	130
9.22.2.4	putc	131
9.22.2.5	metaevent	131
9.22.2.6	mf_sec2ticks	131
9.22.2.7	mf_ticks2sec	132
9.22.2.8	mf_w_header_chunk	132
9.22.2.9	mf_w_meta_event	132
9.22.2.10	mf_w_midi_event	133
9.22.2.11	mf_w_tempo	133
9.22.2.12	mf_w_track_chunk	134
9.22.2.13	mf_w_track_start	134
9.22.2.14	mferror	134
9.22.2.15	mfreadd	134
9.22.2.16	mfreport	135
9.22.2.17	mfreportable	135
9.22.2.18	mftransform	135
9.22.2.19	mfwrite	135
9.22.2.20	midifile	136
9.22.2.21	msg	136
9.22.2.22	msgadd	136
9.22.2.23	msgleng	136
9.22.2.24	read16bit	137
9.22.2.25	read32bit	137
9.22.2.26	readheader	137
9.22.2.27	readmt	137
9.22.2.28	readtrack	138
9.22.2.29	readtrack_m2m	138
9.22.2.30	readvarinum	139
9.22.2.31	sysex	139
9.22.2.32	to16bit	139
9.22.2.33	to32bit	139
9.22.2.34	write16bit	140
9.22.2.35	write32bit	141
9.22.2.36	writevarlen	141
9.22.3	Variable Documentation	141
9.22.3.1	s_chantype	141
9.22.3.2	s_message_buffer	141

9.22.3.3	s_track_header_offset	142
9.23	midifilex.h File Reference	142
9.23.1	Detailed Description	144
9.23.2	Macro Definition Documentation	144
9.23.2.1	note_off	144
9.23.3	Function Documentation	144
9.23.3.1	mf_sec2ticks	144
9.23.3.2	mf_ticks2sec	145
9.23.3.3	mf_w_header_chunk	145
9.23.3.4	mf_w_meta_event	145
9.23.3.5	mf_w_midi_event	146
9.23.3.6	mf_w_tempo	146
9.23.3.7	mf_w_track_chunk	147
9.23.3.8	mf_w_track_start	147
9.23.3.9	mfread	147
9.23.3.10	mftransform	147
9.23.3.11	mfwrite	148
9.23.3.12	midifile	148
9.23.3.13	write32bit	148
9.24	midimapper.cpp File Reference	149
9.24.1	Detailed Description	150
9.24.2	Function Documentation	150
9.24.2.1	midimap_chanpressure	150
9.24.2.2	midimap_init	151
9.24.2.3	midimap_noff	151
9.24.2.4	midimap_non	151
9.24.2.5	midimap_parameter	152
9.24.2.6	midimap_patch	152
9.24.2.7	midimap_pitchbend	152
9.24.2.8	midimap_pressure	153
9.24.2.9	show_maps	153
9.24.3	Variable Documentation	153
9.24.3.1	gs_singleton_midimapper	153
9.25	midimapper.hpp File Reference	153
9.25.1	Detailed Description	154
9.25.2	Function Documentation	156
9.25.2.1	midimap_init	156
9.25.2.2	show_maps	156
9.26	midipp_functions.dox File Reference	156
9.26.1	Detailed Description	156

9.27 stringmap.cpp File Reference	156
9.27.1 Detailed Description	157
9.27.2 Function Documentation	158
9.27.2.1 show	158
9.28 stringmap.hpp File Reference	158
9.28.1 Detailed Description	159
9.28.2 Function Documentation	160
9.28.2.1 show	160
9.29 t2m_no_flex.c File Reference	160
9.29.1 Detailed Description	162
9.29.2 Function Documentation	163
9.29.2.1 yy_create_buffer	163
9.29.2.2 yy_delete_buffer	164
9.29.2.3 yy_flush_buffer	164
9.29.2.4 yy_scan_buffer	164
9.29.2.5 yy_scan_bytes	164
9.29.2.6 yy_scan_string	165
9.29.2.7 yy_switch_to_buffer	165
9.29.2.8 yypop_buffer_state	165
9.29.2.9 yypush_buffer_state	165
9.29.2.10 yyrestart	166
9.29.2.11 yyset_in	166
9.29.2.12 yyset_lineno	166
9.29.3 Variable Documentation	166
9.29.3.1 do_hex	166
9.29.3.2 yy_buffer_stack	166
9.29.3.3 yy_buffer_stack_max	167
9.29.3.4 yy_buffer_stack_top	167
9.29.3.5 yyval	167
9.30 t2mf.h File Reference	167
9.30.1 Detailed Description	168
9.30.2 Macro Definition Documentation	169
9.30.2.1 YY_TYPEDEF YY_SIZE_T	169
9.30.3 Variable Documentation	169
9.30.3.1 do_hex	169
9.30.3.2 yyval	169
9.31 VERSION.h File Reference	169
9.31.1 Detailed Description	169

Chapter 1

midicvt

Author

Chris Ahlstrom 2014-06-21

MIDicvt is a minor cleanup, refactoring, and documentation of the related *midi2text* and *midicomp* projects.

The purpose of these projects is to convert MIDI to and from a human-readable, machine-parsed text format, for easy modifications to MIDI files using standard tools.

In addition to the *midicvt* C program, this project includes a refactored version of *libmidifile* called *libmidifilex*, a new library called *libmidipp*, and a C++ program called *midicvtpp* that includes the functionality of *midicvt* and adds some canned MIDI-to-MIDI transformations to save the user the writing of some scripts.

The current document describes the functions, classes, modules, and other entities used in this project.

1.1 Introduction

The following changes were made to the *midicomp* project to make it easier to understand and modify:

- The ancient Kernighan and Ritchie conventions for C function parameters were used for the parameters of the *libmidifile* functions. We changed these conventions to modern C.
- The comments about functions and values were increased in depth, and were converted to use Doxygen/↔ JavaDocs conventions, so that a reference manual could be generated.
- We regrouped the global variables into semi-private modules, and added accessor functions to those that needed to be used by callers outside of the library.
- We moved code around and added modules so that the code was better organized, and the `main()` function much simpler.
- We rename the library to *libmidifilex* to avoid conflicts with the old library.
- We added a minor reporting facility so that the existing callbacks could dump information.
- We added an additional set of C callbacks that transformed MIDI to MIDI. (These callbacks tend to fix issue in the input MIDI file, as well.)
- We added a C++ class with static callbacks that can be used to remap notes and patches, extract or reject a given MIDI channel, and make other stock changes to a MIDI file.
- We added a C++ program that can perform some MIDI-to-MIDI conversions without the need for scripting.

Chapter 2

The libmidifilex Library

2.1 Introduction to the MIDI File Library Module

This library (*libmidifilex*) is based on the work of Tim Thompson (tjt@blink.att.com), and updates by Michael Czeiszperger (mike@pan.com) and Piet van Oostrum (piet@cs.ruu.nl). Chris Ahlstrom (ahlstromcj@gmail.com) made further updates for readability, documentation, coding conventions, and further bug fixes.

The many global variables were moved into their own modules, and C accessor functions were written for many of them, so that the main routines of programs based on this library did not need the declaration of these variables. This made it a lot easier to figure out the functionality of the code and add new functionality to it.

The following modules are included:

- *midicvt_base* Provides public functions for initializing the private data used in MIDI-to-text and text-to-MIDI conversions.
- *midicvt_globals* Provides access to private options via public pointers or setter/getter (accessor) functions.
- *midicvt_helpers* Provides accessor functions for version information, help, setting up any input or output files, and parsing the command line.
- *midicvt_m2m* Exposes new libmidifilex callback functions for usage in the midicvtp C++ application.
- *midicvt_macros* Defines common macros for MIDI processing, plus some "info-print" macros. Also included are macros to simulate the "bool" type and its values in C++.
- *midicvt_flex* Provides slightly enhanced versions of the support functions and callback function of the older libmidifile library. Also defines the macros defined by the original version of the library.
- *t2mf* These modules specify the text-to-MIDI conversion of the original midicomp program. They are defined using lex (flex) to generate C files that can parse the data. For convenience, a prebuilt module called *t2m_↔no_flex* is provided to avoid having to install flex in order to build this project.

Chapter 3

The libmidipp Library

3.1 Introduction to the MIDlcvT Library Module

This library (*libmidipp*) is merely a convenience library for the C++ application *midicvtpp*.

It provides local versions of the following classes:

- `midipp::csvarray` This class provides a data structure that is a list of items created from a comma-separated-value (CSV) file.
- `midipp::stringmap` This class holds a lookup-table that maps values to string names. It is a fairly reusable class; the `midipp` version is included here to avoid dependencies on other project libraries we maintain. We want the MIDI programs and libraries to stand on their own, for the convenient of other developers.
- `midipp::initree` This class holds data for "sections" of `midipp::stringmap<std::string>` items that represent name/value pairs grouped into sections. This class is created using the data from a special initialization (INI) file.
- `midipp::midimapper` This class provides data structures to remap drum notes, program (patch) values, and other items. This class is created using the data from a `midipp::initree` data structure.

Chapter 4

Licenses, MIDlcvT Projects.

Library:

midicvt_suite and its libraries, applications, and documents

License:

\$XPC_SUITE_GPL_LICENSE\$

Author

Chris Ahlstrom 2014-06-01

4.1 License Terms for the midicvt project.

Wherever the tag \$XPC_SUITE_GPL_LICENSE\$ appears, substitute the appropriate license text, depending on whether the project is a library, application, documentation, or server software. We're not going to include paragraphs of licensing information in every module; you are responsible for coming here to read the licensing information.

These licenses apply to each sub-project and file artifact in the **XPC** library suite.

Wherever the term **XPC** is encountered in this project, it refers to the **XPC Development Suite** on **SourceForge**.↔
net, which is also called the **XPC Library Suite**, and may be provided at other sites.

4.2 XPC Application License

The **XPC** application license is the **GNU GPLv3**.

Copyright (C) 2008-2014 by Chris Ahlstrom

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the

Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301, USA.

The text of the GNU GPL version 3 license can also be found here:

<http://www.gnu.org/licenses/gpl-3.0.txt>

4.3 XPC Library License

The **XPC** library license is the **GNU LGPLv3**.

Copyright (C) 2008-2014 by Chris Ahlstrom

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Lesser Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the

Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301, USA.

The text of the GNU LGPL version 3 license can also be found here:

<http://www.gnu.org/licenses/lgpl-3.0.txt>

4.4 XPC Documentation License

The **XPC** documentation license is the **GNU FDLv1.3**.

Copyright (C) 2014-2014 by Chris Ahlstrom

This documentation is free documentation; you can redistribute it and/or modify it under the terms of the GNU Free Documentation License as published by the Free Software Foundation; either version 1.3 of the License, or (at your option) any later version.

This documentation is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Free Documentation License for more details.

You should have received a copy of the GNU Free Documentation License along with this documentation; if not, write to the

Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301, USA.

The text of the GNU FDL version 1.3 license can also be found here:

<http://www.gnu.org/licenses/fdl.txt>

4.5 XPC Affero License

The **XPC** "Affero" license is the **GNU AGPLv3**.

Copyright (C) 2008-2014 by Chris Ahlstrom

This server software is free server software; you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation; either version 1.3 of the License, or (at your option) any later version.

This documentation is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Free Documentation License for more details.

You should have received a copy of the GNU Affero General Public License along with this server software; if not, write to the

Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301, USA.

The text of the GNU AGPL version 3 license can also be found here:

<http://www.gnu.org/licenses/agpl-3.0.txt>

At the present time, no **XPC** project uses the "Affero" license.

4.6 XPC License Summary

Include one of these licenses in your Doxygen documentation with one of the following Doxygen tags:

```
\ref midicvt_suite_license_subproject  
\ref midicvt_suite_license_application  
\ref midicvt_suite_license_library  
\ref midicvt_suite_license_documentation  
\ref midicvt_suite_license_affero
```

For more information on navigating GNU licensing, see this page:

<http://www.gnu.org/licenses/>

Copies of these licenses (and some logos) are provided in the `licenses` directory of the main project (or you can search for them at *gnu.org*).

Chapter 5

Todo List

Global `m2m_mseq` (short int num)

Need to figure out the portable format of the sequence number bytes.

Class `midipp::stringmap< VALUETYPE >`

Consider implementing lookup by integer index; right now, iterators suffice;

- It would be nice to be able to superimpose a numeric order on the container, somehow.

Global `read32bit` (void)

We need to use an actual 32-bit return type for 64-bit systems.

- Do we need to make the temp variables volatile; can the compiler reorder them?

Global `write32bit` (unsigned long data)

Provide the proper 32-bit data types needed to do this more portably.

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

<code>midipp::csvarray</code>	Assist in parsing a file that has lines of comma-separated values	??
<code>midipp::initree</code>	This class provides a way to read and represent an INI file as a kind of tree structure that can be navigate to look up information	??
<code>midipp::midimapper</code>	This class provides for some basic remappings to be done to MIDI files, using the old and new facilities of libmidifilex	??
<code>midipp::stringmap< VALUETYPE ></code>	Provides an std::map wrapper geared towards using std::string as a key	??

Chapter 7

File Index

7.1 File List

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

csvarray.cpp	Primitive, but useful, CSV-file input parser	??
csvarray.hpp	??
ininames.hpp	Provides some free functions to write INI files, given a midipp::csvarray object	??
initree.cpp	Primitive INI-file input parser derived from the XPC library	??
initree.hpp	Provides a "tree" that represents an INI file with named and unnamed sections, and name/value pairs	??
iniwriting.cpp	Provides some free functions to write INI files, given a midipp::csvarray object	??
iniwriting.hpp	Provides some free functions to write INI files, given a midipp::csvarray object	??
midicvt_base.c	This module provides functions for basic MIDI/text conversions	??
midicvt_base.h	This module provides functions for a basic MIDI/text conversion application	??
midicvt_globals.c	This module provides a place to collect all the global variables used, and functions to set them	??
midicvt_globals.h	This module provides global variables and accessor functions for the midicvt portion of libmidifilex	??
midicvt_helpers.c	This module provides the help text for midicvt	??
midicvt_helpers.h	This module declares the help and version functions for midicvt	??
midicvt_m2m.c	This module provides functions for basic MIDI-to-MIDI conversions	??
midicvt_m2m.h	This module provides functions for a basic MIDI-to-MIDI conversion application	??
midicvt_macros.h	This module provides macros for generating simple messages, MIDI parameters, and more . .	??
midicvt_main.c	This module creates the midicvt program for translating between MIDI files and text files	??
midicvtpp_main.cpp	This module creates the midicvtpp program for translating between MIDI files and text files, and doing some basic remapping operations	??

midifilex.c	
This module provides functions for handling the reading and writing of MIDI files	??
midifilex.h	
This module provides functions for libmidifilex	??
midimapper.cpp	
This module provides functions for advanced MIDI/text conversions	??
midimapper.hpp	
This module provides functions for advanced MIDI/text conversions	??
stringmap.cpp	??
stringmap.hpp	??
t2m_no_flex.c	
This file is to be used instead of the dynamically generated t2mflex.c if there is no flex program installed or if the installed flex is old [e.g 2.5.35 as opposed to 2.5.39 (!)]	??
t2mf.h	
This module provides global variables for the text-to-MIDI conversion portion of libmidifilex . . .	??
t2mflex.c	??
VERSION.h	
This module provides simpler version information for the libmidifilex library	??

Chapter 8

Data Structure Documentation

8.1 midipp::csvarray Class Reference

The csvarray class assist in parsing a file that has lines of comma-separated values.

```
#include <csvarray.hpp>
```

Collaboration diagram for midipp::csvarray:

midipp::csvarray
<ul style="list-style-type: none"> - m_separator - m_source_file - m_name - m_csv_lines - m_minimum_length - m_maximum_length - m_is_valid
<ul style="list-style-type: none"> + csvarray() + csvarray() + csvarray() + operator=() + ~csvarray() + is_valid() + rows() + source_file() + name() + name() + size() + empty() + min_count() + max_count() # readfile() # is_comment() # clear()

Public Types

- `typedef std::vector< std::string > Fields`
Provides a handy type for the data structure that the csvarray class stores in a vector.
- `typedef std::vector< Fields > Rows`
Provides a type that holds a vector of Field elements.

Public Member Functions

- [csvarray](#) ()

- Creates an unnamed and empty csvarray.*
- `csvarray` (const std::string &name, const std::string &filespec, char separator= ',')
- Creates an named, but empty, csvarray.*
- `csvarray` (const `csvarray` &source)
- Copy constructor for csvarray.*
- `csvarray & operator=` (const `csvarray` &source)
- Principal assignment operator for csvarray.*
- virtual `~csvarray` ()
- bool `is_valid` () const
- 'Getter' function for member m_is_valid;*
- const `Rows & rows` () const
- 'Getter' function for member m_csv_lines*
- const std::string & `source_file` () const
- 'Getter' function for member m_source_file*
- const std::string & `name` () const
- 'Getter' function for member m_name*
- void `name` (const std::string &n)
- 'Setter' function for member m_name*
- size_t `size` () const
- bool `empty` () const
- int `min_count` () const
- 'Getter' function for member m_minimum_length*
- int `max_count` () const
- 'Getter' function for member m_maximum_length*

Protected Member Functions

- bool `readfile` (const std::string &filespec)
- Opens a file, and tries to construct an csvarray object, and a number of section objects, in it.*
- bool `is_comment` (char c)
- Checks for a comment character.*
- void `clear` ()
- Allows the container to be emptied of Section objects.*

Private Attributes

- char `m_separator`
- Provides the value for the separator.*
- std::string `m_source_file`
- Provides the file-name of the file from which this data was instantiated.*
- std::string `m_name`
- Provides the name of this object, for future reference.*
- `Rows m_csv_lines`
- Provides a section container.*
- int `m_minimum_length`
- Indicates the minimum number of fields found in a row.*
- int `m_maximum_length`
- Indicates the maximum number of fields found in a row.*
- bool `m_is_valid`
- Indicates if construction succeeded in all respects.*

8.1.1 Detailed Description

The `csvarray` class assist in parsing a file that has lines of comma-separated values.

It creates a vector of vectors out of this file. The main vector contains one element for each usable line in the file. Each element is itself a vector of strings. A record is kept of the number of elements in each vector. Generally, that number should be the same for every line in the file, but this is not required.

Definition at line 37 of file `csvarray.hpp`.

8.1.2 Member Typedef Documentation

8.1.2.1 `typedef std::vector<std::string> midipp::csvarray::Fields`

Provides a handy type for the data structure that the `csvarray` class stores in a vector.

Each string in this vector is called a "field".

Definition at line 48 of file `csvarray.hpp`.

8.1.3 Constructor & Destructor Documentation

8.1.3.1 `midipp::csvarray::csvarray ()`

Creates an unnamed and empty `csvarray`.

The name can be added later with a call to `name()`, and sections can be added with the `insert()` function.

Definition at line 44 of file `csvarray.cpp`.

8.1.3.2 `midipp::csvarray::csvarray (const std::string & name, const std::string & filespec, char separator = ' , ')`

Creates an named, but empty, `csvarray`.

The name can be added or modified later with a call to `name()`, and sections can be added with the `std::vector::push_back()` function.

If a file-name is given, the file is read to fill in the lines. unnamed section. If it isn't used, the overhead isn't significant.

Parameters

<i>name</i>	Provides the name of the <code>csvarray</code> object.
<i>filespec</i>	Provides the optional full path file-specification for the file to be opened and read to create all of the lines specified in that file.
<i>separator</i>	Provides the separator character that demarcates fields. The default value of this parameter is the comma.

Definition at line 78 of file `csvarray.cpp`.

8.1.3.3 `midipp::csvarray::csvarray (const csvarray & source)`

Copy constructor for `csvarray`.

Parameters

<i>source</i>	Provides the original object to be copied.
---------------	--

Definition at line 102 of file `csvarray.cpp`.

8.1.3.4 `virtual midipp::csvarray::~~csvarray () [inline],[virtual]`

Destructor:

Provided as a virtual destructor so that we can derive from this class.

Definition at line 127 of file `csvarray.hpp`.

8.1.4 Member Function Documentation

8.1.4.1 `bool midipp::csvarray::empty () const [inline]`

Accessor:

`m_csv_lines.empty()`

Returns

Returns true if the container is empty.

Definition at line 196 of file `csvarray.hpp`.

8.1.4.2 `bool midipp::csvarray::is_comment (char c) [inline],[protected]`

Checks for a comment character.

Returns

Returns 'true' if the character is in the set # ;

Definition at line 230 of file `csvarray.hpp`.

8.1.4.3 `csvarray & midipp::csvarray::operator= (const csvarray & source)`

Principal assignment operator for `csvarray`.

Parameters

<i>source</i>	Provides the original object to be assigned.
---------------	--

Definition at line 122 of file `csvarray.cpp`.

8.1.4.4 `bool midipp::csvarray::readfile (const std::string & filespec) [protected]`

Opens a file, and tries to construct an `csvarray` object, and a number of section objects, in it.

This function reads a file line by line.

- Blank lines are skipped.
- Lines that start with ";", "#", "!", "", or "" are considered to be blank lines.
- Lines that start with "[" are potentially section-names; if not, then they are considered errors and processing fails.
- Lines that start with alphabetic characters (case-sensitive) are potentially options.
 - If an equal sign "=" follows the first characters, unquoted, then the line is an option+value line.
 - If there is no equal sign, then the option will be treated like a bare flag.
- Values are anything following the equals. Leading and trailing spaces are stripped, unless quoted.

Parameters

<i>filespec</i>	Provides the full path to the file to be processed.
-----------------	---

Returns

Returns 'true' if any legal option was found, and 'false' if anything bad was found.

Definition at line 165 of file csvarray.cpp.

8.1.4.5 `size_t midipp::csvarray::size () const` `[inline]`

Accessor:

`m_csv_lines.size()`

Returns

Returns the number of VALUETYPE objects in the container.

Definition at line 184 of file csvarray.hpp.

8.1.5 Field Documentation

8.1.5.1 `Rows midipp::csvarray::m_csv_lines` `[private]`

Provides a section container.

All of the sections together specify all of the existing sections in an INI file.

Definition at line 84 of file csvarray.hpp.

8.1.5.2 `bool midipp::csvarray::m_is_valid` `[private]`

Indicates if construction succeeded in all respects.

An empty csvarray is not valid. Trying to read a non-existent or corrupt CSV file results in a csvarray that is not valid.

We're not big on throwing exceptions as a means of error handling.

Definition at line 107 of file csvarray.hpp.

8.1.5.3 `char midipp::csvarray::m_separator` `[private]`

Provides the value for the separator.

By default, this is a comma, but other characters, including a tab, can be used as well. The separator character can never be included in a Field.

Definition at line 64 of file csvarray.hpp.

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

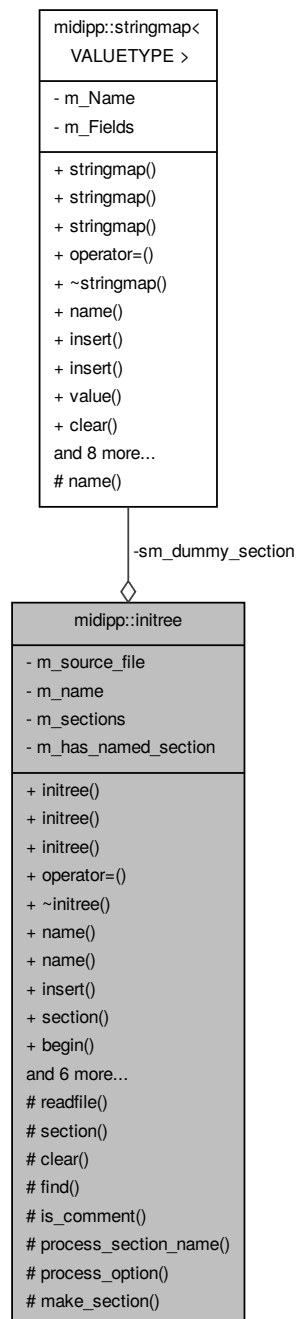
- [csvarray.hpp](#)
- [csvarray.cpp](#)

8.2 midipp::initree Class Reference

This class provides a way to read and represent an INI file as a kind of tree structure that can be navigate to look up information.

```
#include <initree.hpp>
```

Collaboration diagram for midipp::initree:



Public Types

- typedef `midipp::stringmap`
`< std::string > Section`
Provides a handy name for the data structure that the `initree` class stores in a map.
- typedef `std::map< std::string,`
`Section > Container`
Provides a type that holds a map of strings, keyed by strings.
- typedef `Container::const_iterator` `const_iterator`
Provides a constant-iterator type for notational convenience.
- typedef `Container::iterator` `iterator`
Provides an iterator type for notational convenience.
- typedef `std::pair< std::string,`
`Section > pair`
Provides a pair type for notational convenience.

Public Member Functions

- `initree ()`
Creates an unnamed and empty `initree`.
- `initree (const std::string &name, const std::string &filespec)`
Creates a named, but empty, `initree`.
- `initree (const initree &source)`
Creates an unnamed and empty `initree`.
- `initree & operator= (const initree &source)`
Creates an unnamed and empty `initree`.
- virtual `~initree ()`
- `const std::string & name () const`
'Getter' function for member `m_name`
- `void name (const std::string &n)`
'Setter' function for member `m_name`
- `int insert (const std::string §ionname, const Section §ion)`
Allows the insertion of an `stringmap` object into the container.
- `const Section & section (const std::string §ionname) const`
Provides a way to look up a section name and return a `Section` value, as a reference.
- `const_iterator begin () const`
- `const_iterator end () const`
- `iterator begin ()`
- `iterator end ()`
- `size_t size () const`
- `bool empty () const`
- `const_iterator find (const std::string §ionname) const`

Protected Member Functions

- `bool readfile (const std::string &filespec)`
Opens a file, and tries to construct an `initree` object, and a number of section objects, in it.
- `Section & section (const std::string §ionname)`
Provides a way to look up a section name and return a `Section` value, as a reference.
- `void clear ()`
Allows the container to be emptied of `Section` objects.

- [iterator find](#) (const std::string §ionname)
- bool [is_comment](#) (char c)
Checks for a comment character.
- std::string [process_section_name](#) (const std::string &s, std::string::size_type p)
Extracts a section name, which must start with an alphabetic character, and can contain embedded spaces.
- bool [process_option](#) (const std::string &s, std::string::size_type p, const std::string §ionname)
Extracts an option from a line of text, where an option name must start with an alphabetic character, cannot have embedded spaces, and might not have a value.
- bool [make_section](#) (const std::string §ionname)
Created a new section, with the given name, and inserts it into the initree.

Private Attributes

- std::string [m_source_file](#)
Provides the file-name of the file from which this tree of data was instantiated.
- std::string [m_name](#)
Provides the name of this object, for future reference.
- [Container m_sections](#)
Provides a section container.
- bool [m_has_named_section](#)
Indicates if any named sections were added to the tree.

Static Private Attributes

- static [Section sm_dummy_section](#)
Provides an empty section to use as a return value.

8.2.1 Detailed Description

This class provides a way to read and represent an INI file as a kind of tree structure that can be navigate to look up information.

An initree consists of Sections. A Section is a map of string values, keyed by the name of each value. A section can be named or un-named. There can be only one un-named section in an initree.

Definition at line 35 of file initree.hpp.

8.2.2 Member Typedef Documentation

8.2.2.1 typedef Container::const_iterator midipp::initree::const_iterator

Provides a constant-iterator type for notational convenience.

```
typedef typename Container::const_iterator const_iterator;
```

Definition at line 71 of file initree.hpp.

8.2.2.2 typedef std::map<std::string, Section> midipp::initree::Container

Provides a type that holds a map of strings, keyed by strings.

The string key is a section name. A section name comes from the "[Name]" token in an INI file.

The stringmap value is itself a map of strings, keyed by strings, where the key is the name of an option, and the value is the option's value. This pair comes from an entry in the INI file of the form "Name = Value".

Definition at line 63 of file initree.hpp.

8.2.2.3 typedef Container::iterator midipp::initree::iterator

Provides an iterator type for notational convenience.

```
typedef typename Container::iterator iterator;
```

Definition at line 81 of file initree.hpp.

8.2.2.4 typedef midipp::stringmap<std::string> midipp::initree::Section

Provides a handy name for the data structure that the initree class stores in a map.

Makes the code quite a bit easier to grok.

Note that Sections will generally have a name. However, an unnamed section is useful for representing INI files that have no section information.

Definition at line 49 of file initree.hpp.

8.2.3 Constructor & Destructor Documentation

8.2.3.1 midipp::initree::initree ()

Creates an unnamed and empty initree.

The name can be added later with a call to [name\(\)](#), and sections can be added with the [insert\(\)](#) function.

However, to simplify the code, the constructor always creates an unnamed section.

Definition at line 95 of file initree.cpp.

8.2.3.2 midipp::initree::initree (const std::string & name, const std::string & filespec)

Creates a named, but empty, initree.

The name can be added or modified later with a call to [name\(\)](#), and sections can be added with the [insert\(\)](#) function.

If a file-name is given, the file is read to fill in the sections.

To simplify the code, the constructor always creates an unnamed section. If it isn't used, the overhead isn't significant.

Parameters

<i>name</i>	Provides the name of the initree object.
<i>filespec</i>	Provides the optional full path file-specification for the file to be opened and read to create all of the sections specified in that file.

Definition at line 124 of file initree.cpp.

8.2.3.3 midipp::initree::initree (const initree & source)

Creates an unnamed and empty initree.

The name can be added later with a call to [name\(\)](#), and sections can be added with the [insert\(\)](#) function.

Definition at line 146 of file initree.cpp.

8.2.3.4 virtual midipp::initree::~~initree () [inline],[virtual]

Destructor:

Provided as a virtual destructor so that we can derive from this class.

Definition at line 141 of file initree.hpp.

8.2.4 Member Function Documentation

8.2.4.1 `const_iterator midipp::initree::begin () const` `[inline]`

Accessor:

`m_sections.begin()` const Makes this class look more like an STL container.

Returns

Returns a const iterator for the first element of the container, if any. Otherwise `end()` is returned.

Definition at line 211 of file initree.hpp.

8.2.4.2 `iterator midipp::initree::begin ()` `[inline]`

Accessor:

`m_sections.begin()` This function makes this class look more like an STL container, good for using "for each" constructs.

Warning

Only a subset of `std::map` members are reimplemented in the `stringmap` class.

Returns

Returns an iterator for the first element of the container, if any. Otherwise `end()` is returned.

Definition at line 243 of file initree.hpp.

8.2.4.3 `bool midipp::initree::empty () const` `[inline]`

Accessor:

`m_sections.empty()`

Returns

Returns true if the container is empty.

Definition at line 280 of file initree.hpp.

8.2.4.4 `const_iterator midipp::initree::end () const` `[inline]`

Accessor:

`m_sections.end()` const Makes this class look more like an STL container.

Returns

Returns a const iterator indicating the end of the container.

Definition at line 224 of file initree.hpp.

8.2.4.5 `iterator midipp::initree::end () [inline]`

Accessor:

`m_sections.end()` Makes this class look more like an STL container.

Returns

Returns an iterator indicating the end of the container.

Definition at line 256 of file `initree.hpp`.

8.2.4.6 `const_iterator midipp::initree::find (const std::string & sectionname) const [inline]`

Accessor:

`m_sections.find() const` Makes this class look more like an STL container.

Returns

Returns a const iterator for the found element of the container, if any. Otherwise `end()` is returned.

Definition at line 294 of file `initree.hpp`.

8.2.4.7 `iterator midipp::initree::find (const std::string & sectionname) [inline], [protected]`

Accessor:

`m_sections.find()` This function makes this class look more like an STL container.

Returns

Returns an iterator for the found element of the container, if any. Otherwise `end()` is returned.

Definition at line 349 of file `initree.hpp`.

8.2.4.8 `int midipp::initree::insert (const std::string & sectionname, const Section & section) [inline]`

Allows the insertion of an stringmap object into the container.

Parameters

<i>section</i>	The string that is to serve as the lookup value for the inserted object.
<i>value</i>	The value object to be added to the container.

Returns

The size of the container after insertion is returned. If important, the caller should check that the size is one larger.

Definition at line 179 of file `initree.hpp`.

8.2.4.9 `bool midipp::initree::is_comment (char c) [inline], [protected]`

Checks for a comment character.

Comment characters are:

- Hash mark "#". The comment character for shell scripts.
- Semicolon ";". The comment character for assembly language.
- Exclamation point "!" The comment character for Fluxbox configuration files..
- Single quote. The comment character for Visual BASIC.
- Double quote. The comment character for vim scripts.

Parameters

<code>c</code>	The character to be checked.
----------------	------------------------------

Returns

Returns 'true' if the character is in the comment-character set.

Definition at line 373 of file initree.hpp.

8.2.4.10 `bool midipp::initree::make_section (const std::string & sectionname) [protected]`

Created a new section, with the given name, and inserts it into the initree.

Parameters

<code>sectionname</code>	Provides the name of new section that is to be created.
--------------------------	---

Returns

Returns 'true' if the section was successfully inserted into the initree.

Side-effect(s):

If the created section has a name, then `m_has_named_section` is set to 'true'.

Definition at line 503 of file initree.cpp.

8.2.4.11 `initree & midipp::initree::operator= (const initree & source)`

Creates an unnamed and empty initree.

The name can be added later with a call to [name\(\)](#), and sections can be added with the [insert\(\)](#) function.

Definition at line 163 of file initree.cpp.

8.2.4.12 `bool midipp::initree::process_option (const std::string & s, std::string::size_type p, const std::string & sectionname) [protected]`

Extracts an option from a line of text, where an option name must start with an alphabetic character, cannot have embedded spaces, and might not have a value.

This function handles options of the following forms:

```

option
option = value
option = "multiple-token value"
option = ""

```

The legal characters in an option name are letters, digits, a hyphen (minus), and underscore.

The "=" is mandatory, though the spaces around it are optional.

Every option has a string value. If the value is not provided, then it is set to empty. If the string value is in double-quotes, it is translated via `snprintf()`. Use double quotes if spaces and special characters are part of the string.

Comment characters other than single- or double-quotes end a value.

Parameters

<i>s</i>	Provides the current line of text, which starts with a letter.
<i>p</i>	Provides the position of the first letter.
<i>currentsection</i>	Points to the current Section (which might be an unnamed section). Option entries are added to this section. This pointer is checked elsewhere, and will not be null here unless we haven't yet hit a section. In that case, this function will create an unnamed section, if one does not yet exist.

Returns

Returns 'true' if the option was proper and properly processed. Otherwise, 'false' is returned.

Definition at line 376 of file `initree.cpp`.

8.2.4.13 `std::string midipp::initree::process_section_name (const std::string & s, std::string::size_type p)` [protected]

Extracts a section name, which must start with an alphabetic character, and can contain embedded spaces.

This function processes a line of the follow formats to extract the section name:

```

[Name]
[ Name ]
[ Section Name ]

```

Leading and trailing spaces are not included in the section name. Embedded spaces are included. Note that the trailing ']' is required, or the line is considered to be malformed.

Parameters

<i>s</i>	Provides the current line of text, which starts with '['.
<i>p</i>	Provides the position of the '[' character.

Returns

Returns the name of the newly-created section. If this name is empty, then an error occurred.

Definition at line 301 of file `initree.cpp`.

8.2.4.14 `bool midipp::initree::readfile (const std::string & filespec)` [protected]

Opens a file, and tries to construct an `initree` object, and a number of section objects, in it.

This function reads a file line by line.

- Blank lines are skipped.
- Lines that start with ";", "#", "!", "", or double-quote are considered to be blank lines.
- Lines that start with "[" are potentially section-names; if not, then they are considered errors and processing fails.
- Lines that start with alphabetic characters (case-sensitive) are potentially options.
 - If an equal sign "=" follows the first characters, unquoted, then the line is an option+value line.
 - If there is no equal sign, then the option will be treated like a bare flag.
- Values are anything following the equals. Leading and trailing spaces are stripped, unless quoted.

Parameters

<i>filespec</i>	Provides the full path to the file to be processed.
-----------------	---

Returns

Returns 'true' if any legal option was found, and 'false' if anything bad was found.

The code checks only for white-space characters; it is possible to embed control characters and have them treated as tokens. At your own risk, baby!

Definition at line 204 of file initree.cpp.

8.2.4.15 `const Section& midipp::initree::section (const std::string & sectionname) const` `[inline]`

Provides a way to look up a section name and return a Section value, as a reference.

This is the const version, meant for outsiders to use.

Parameters

<i>sectionname</i>	Provides the section name to be looked up.
--------------------	--

Returns

Returns a constant reference to the Section found.

Definition at line 197 of file initree.hpp.

8.2.4.16 `Section& midipp::initree::section (const std::string & sectionname)` `[inline], [protected]`

Provides a way to look up a section name and return a Section value, as a reference.

Parameters

<i>sectionname</i>	Provides the section name to be looked up.
--------------------	--

Returns

Returns a reference to the Section found. If it was not found, then a default-constructor (empty) Section is returned. It is the author's responsibility to check if the Section is useful or not.

Definition at line 322 of file initree.hpp.

8.2.4.17 `size_t midipp::initree::size () const [inline]`

Accessor:

`m_sections.size()`

Returns

Returns the number of VALUETYPE objects in the container.

Definition at line 268 of file `initree.hpp`.

8.2.5 Field Documentation

8.2.5.1 `Container midipp::initree::m_sections [private]`

Provides a section container.

All of the sections together specify all of the existing sections in an INI file.

Definition at line 109 of file `initree.hpp`.

8.2.5.2 `initree::Section midipp::initree::sm_dummy_section [static],[private]`

Provides an empty section to use as a return value.

Must provide an initialization for this static member of `initree`.

This dummy value helps us avoid the need for pointers.

Definition at line 122 of file `initree.hpp`.

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

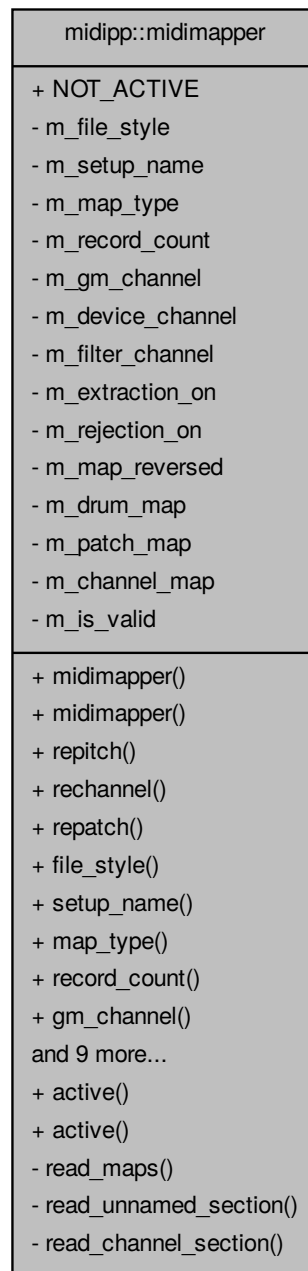
- [initree.hpp](#)
- [initree.cpp](#)

8.3 midipp::midimapper Class Reference

This class provides for some basic remappings to be done to MIDI files, using the old and new facilities of `libmidifilex`.

```
#include <midimapper.hpp>
```

Collaboration diagram for midipp::midimapper:



Public Member Functions

- [midimapper](#) ()
This constructor creates an unnamed, no-change mapping object.
- [midimapper](#) (const std::string &name, const std::string &filespec="", bool reverse_it=false, int [filter_↔](#) channel=NOT_ACTIVE, bool reject_it=false)
This constructor creates an named mapping object, reading the mapping from an INI file.

- int [repitch](#) (int channel, int input)
Changes a note value based on the note-mapping that was provided.
- int [rechannel](#) (int channel)
Changes a channel value based on the channel-mapping that was provided.
- int [repatch](#) (int program)
Changes a program/patch value based on the patch-mapping that was provided.
- const std::string & [file_style](#) () const
'Getter' function for member m_file_style
- const std::string & [setup_name](#) () const
'Getter' function for member m_setup_name
- const std::string & [map_type](#) () const
'Getter' function for member m_map_type
- int [record_count](#) () const
'Getter' function for member m_record_count
- int [gm_channel](#) () const
'Getter' function for member m_gm_channel
- int [device_channel](#) () const
'Getter' function for member m_device_channel
- int [filter_channel](#) () const
'Getter' function for member m_filter_channel
- bool [extract](#) () const
'Getter' function for member m_extraction_on
- bool [reject](#) () const
'Getter' function for member m_rejection_on
- bool [valid](#) () const
'Getter' function for member m_is_valid;
- const std::map< int, int > & [drum_map](#) () const
'Getter' function for member m_drum_map Returns a reference to the note map.
- const std::map< int, int > & [patch_map](#) () const
'Getter' function for member m_patch_map Returns a reference to the patch map.
- const std::map< int, int > & [channel_map](#) () const
'Getter' function for member m_channel_map Returns a reference to the channel map.
- bool [map_reversed](#) () const
'Getter' function for member m_map_reversed

Static Public Member Functions

- static bool [active](#) (int value)
Determines if the value parameter is usable, or "active".
- static bool [active](#) (int v1, int v2)
Determines if both value parameters are usable, or "active".

Static Public Attributes

- static const int [NOT_ACTIVE](#) = -1
Provides a constant to indicate an inactive or invalid integer value.

Private Member Functions

- bool [read_maps](#) (const std::string &filename)
Creates an integer mapping map from the given INI file.
- bool [read_unnamed_section](#) (const initree &it)
This function searches through the unnamed section at the top of the file.
- bool [read_channel_section](#) (const initree &it)
This function searches through the Channel section, if it exists.

Private Attributes

- std::string [m_file_style](#)
Provides the style of the INI file.
- std::string [m_setup_name](#)
Provides a nice tag name for the setup, nothing more.
- std::string [m_map_type](#)
Indicates what kind of mapping is allegedly provided by the file.
- int [m_record_count](#)
Provides the number of records (lines) or sections in the INI file.
- int [m_gm_channel](#)
Provides the channel to use for General MIDI drums.
- int [m_device_channel](#)
Provides the channel that is used by the native device.
- int [m_filter_channel](#)
Provides an optional channel number to filter in the MIDI file.
- bool [m_extraction_on](#)
A faster way to check if m_filter_channel is enabled.
- bool [m_rejection_on](#)
Changes the extraction-channel to a rejection channel, where only the channel that matches is dropped from the output MIDI file.
- bool [m_map_reversed](#)
Indicates that the mapping should occur in the reverse direction.
- std::map< int, int > [m_drum_map](#)
Provides the mapping between pitches.
- std::map< int, int > [m_patch_map](#)
Provides the mapping between patches (programs).
- std::map< int, int > [m_channel_map](#)
Provides the mapping between channels (optional).
- bool [m_is_valid](#)
Indicates if the setup is valid.

8.3.1 Detailed Description

This class provides for some basic remappings to be done to MIDI files, using the old and new facilities of libmidifilex. It works by holding all sorts of standard C++ map objects that are used to translate from one numeric value to another.

For use in the midicvtp application, a single global instance of this object is created, and is used in static C-style callback functions that can be used in the C library libmidifilex.

Definition at line 109 of file midimapper.hpp.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 `midipp::midimapper::midimapper ()`

This constructor creates an unnamed, no-change mapping object.

This object can be used for testing.

Definition at line 83 of file `midimapper.cpp`.

8.3.2.2 `midipp::midimapper::midimapper (const std::string & name, const std::string & filespec = " ", bool reverse_it = false, int filter_channel = NOT_ACTIVE, bool reject_it = false)`

This constructor creates an named mapping object, reading the mapping from an INI file.

This object can be used for testing.

Parameters

<i>name</i>	Provides a handy name to refer to when reading the output of this object.
<i>filespec</i>	Provides the full file-path specification of an INI-style file to be read. See the example files in the <code>tests/inifiles</code> directory.
<i>reverse_it</i>	Indicates if the numeric values are to be swapped. Thus, applying the MIDI mapper operation for a given INI file, and then reversing it, should yield something like the original MIDI file.
<i>filter_channel</i>	Provides a single channel to be treated specially, being either saved alone, or dropped from the MIDI file. This value should range from 1 to 16, and is converted internally to the 0 to 15 range. The default value is NOT_ACTIVE (-1), which means no channel filtering is to be done.
<i>reject_it</i>	If the <i>filter_channel</i> is valid, this boolean parameter indicates that the channel is to be dropped from the output MIDI file. If false (the default value), the channel is the only one kept in the output MIDI file.

Definition at line 138 of file `midimapper.cpp`.

8.3.3 Member Function Documentation

8.3.3.1 `static bool midipp::midimapper::active (int value) [inline], [static]`

Determines if the value parameter is usable, or "active".

Parameters

<i>value</i>	The integer value to be checked.
--------------	----------------------------------

Returns

Returns true if the value is not NOT_ACTIVE.

Definition at line 316 of file `midimapper.hpp`.

8.3.3.2 `static bool midipp::midimapper::active (int v1, int v2) [inline], [static]`

Determines if both value parameters are usable, or "active".

Parameters

<i>v1</i>	The first integer value to be checked.
<i>v2</i>	The second integer value to be checked.

Returns

Returns true if both of the values are not NOT_ACTIVE.

Definition at line 334 of file midimapper.hpp.

8.3.3.3 `const std::map<int, int> & midipp::midimapper::drum_map () const` `[inline]`

'Getter' function for member *m_drum_map* Returns a reference to the note map.

The weird thing was that, when I had left the reference operator out, `show_note_map()` would show a lot of notes missing from the map, as if copying the map wasn't working properly!!!

Definition at line 441 of file midimapper.hpp.

8.3.3.4 `bool midipp::midimapper::read_channel_section (const initree & it)` `[private]`

This function searches through the Channel section, if it exists.

Each entry must be of the following format, or an error will occur:

```
ch_01 = 10
```

Parameters

<i>it</i>	Provides the tree of INI name/value pairs to analyze.
-----------	---

Returns

Returns true if there was no channel section to bother with, or if there was, and the processing of it succeeded. This function assumes the [read_unnamed_section\(\)](#) function was already called, and succeeded.

Definition at line 474 of file midimapper.cpp.

8.3.3.5 `bool midipp::midimapper::read_maps (const std::string & filename)` `[private]`

Creates an integer mapping map from the given INI file.

An initree object is temporarily created in order to read the file and provide the settings we need. We don't need to use shared-pointers, since this function is written in a safe manner.

This function iterates through the sections. Note than an unnamed section is treated differently: The "gm-channel" and "dev-channel" values are looked up and set.

Inserts the pair of (gm_note, device_note), depending on the *m_map_reversed* flag. If false (the normal case), then we want the key to be the gm_note, and the value to be the device_note. That is, given a MIDI file scored for a non-GM device, we want the notes to be remapped to the notes that GM needs to make a the sound intended by the device-note.

If the *m_map_reversed* flag is true, then we want to take a GM MIDI file and re-map it for the corresponding device note.

Parameters

<i>filename</i>	Provides the full path specification of the file to be read.
-----------------	--

Returns

Returns true if the operation succeeded.

Definition at line 202 of file midimapper.cpp.

8.3.3.6 `bool midipp::midimapper::read_unnamed_section (const initree & it) [private]`

This function searches through the unnamed section at the top of the file.

Note that an unnamed section is treated differently: The "gm_channel" and "device_channel" values are looked up and set.

Parameters

<i>it</i>	Provides the tree of INI name/value pairs to analyze.
-----------	---

Returns

Returns true if the whole tree has any elements. If it has an unnamed section, then 'true' also indicates that no errors occurred in processing those values.

Definition at line 357 of file midimapper.cpp.

8.3.3.7 `int midipp::midimapper::rechannel (int channel)`

Changes a channel value based on the channel-mapping that was provided.

Note that there are actually two possible channel-mappings. The first one is based on the values of gm-channel and dev-channel, and is used to map drum tracks (e.g. from channel 16 to channel 10). The second mapping allows more general mapping, and is applied only if the channel isn't affected by the drum mapping. The second form is specified by the "[Channels]" section.

The reversal option is accounted for at setup time for both of these types of mappings, as was done for the drum (note) value mappings.

If the `m_filter_channel` value is valid (between 0 and 15), and the input channel does *not* match this value, then -1 ([midimapper::NOT_ACTIVE](#)) is returned to indicate to reject the event.

Parameters

<i>channel</i>	Provides the value of the input channel, as obtained from the input MIDI file.
----------------	--

Returns

Returns one of the following values:

- The output channel value corresponding to the input channel value if the input channel matches `m_↔gm_channel`, and `m_gm_channel` is set.
- The original input channel if the input channel is not one that matches `m_gm_channel` in the channel mapping, or if `m_gm_channel` is not activated (i.e. set to -1 [[NOT_ACTIVE](#)]).
- [NOT_ACTIVE](#) (-1) is returned if the input channel doesn't match `m_filter_channel` and `m_extraction_on` is true. In other words, only one channel is allowed to be output
- [NOT_ACTIVE](#) (-1) is returned if the input channel does match `m_filter_channel` and `m_rejection_on` is true. In other words, one channel is systematically rejected from the output.

Only if `m_gm_channel` and `m_device_channel` have been set are they used to detect and alter the channel number of the event. If they aren't active, or if the channel isn't the drum channel to be remapped, then the "[Channels]" map is checked.

Definition at line 613 of file `midimapper.cpp`.

8.3.3.8 `int midipp::midimapper::repatch (int program)`

Changes a program/patch value based on the patch-mapping that was provided.

Since the reversal option for note values is set up during the construction of the note map, the `m_map_reversed` flag does not need to be checked in this function.

Parameters

<i>program</i>	Provides the value of the input patch, as obtained from the input MIDI file.
----------------	--

Returns

Returns the output patch value corresponding to the input patch value. If the input patch is not found in the map, it is returned unaltered. The map is empty if an empty INI file-name was passed to the constructor.

Definition at line 666 of file `midimapper.cpp`.

8.3.3.9 `int midipp::midimapper::repitch (int channel, int input)`

Changes a note value based on the note-mapping that was provided.

Since the reversal option for note values is set up during the construction of the note map, the `m_map_reversed` flag does not need to be checked in this function.

Parameters

<i>channel</i>	Provides the channel for the input note. Only if the channel matches the one channel to be remapped, will repitching occur.
<i>input</i>	Provides the value of the input note, as obtained from the input MIDI file.

Returns

Returns the output note value corresponding to the input note value. If the input note is not found in the map, it is returned unaltered. The map is empty if an empty INI file-name was passed to the constructor.

Definition at line 562 of file `midimapper.cpp`.

8.3.4 Field Documentation

8.3.4.1 `std::map<int, int> midipp::midimapper::m_channel_map [private]`

Provides the mapping between channels (optional).

If `m_map_reversed` is true, then the mapping of channels is reversed.

Definition at line 282 of file `midimapper.hpp`.

8.3.4.2 `int midipp::midimapper::m_device_channel [private]`

Provides the channel that is used by the native device.

Older MIDI equipment sometimes used channel 16 for percussion.

The name of this attribute in the INI file is "dev-channel". Case is significant.

Definition at line 204 of file midimapper.hpp.

8.3.4.3 `std::map<int, int> midipp::midimapper::m_drum_map` [private]

Provides the mapping between pitches.

If `m_map_reversed` is false, then the key is the GM pitch/note, and the value is the device pitch/note (which is the GM note needed to produce the same sound in GM as the device would have produced). If `m_map_reversed` is true, then the key is the GM pitch/note, and the value is the device pitch/note, so that the MIDI file will be converted from GM mapping to device mapping.

Definition at line 262 of file midimapper.hpp.

8.3.4.4 `std::string midipp::midimapper::m_file_style` [private]

Provides the style of the INI file.

This is one of the following strings:

- "simple". All of the mapping information is contained in a single comma-separated value. We probably won't end up supporting this option, as it is a bit harder for humans to read. See `doc/GM_PSS-790_↔Drums-save.ini`.
- "sectioned". The mapping information for each note is contained in its own "Drum" section. See `doc/GM_↔PSS-790_Drums.ini`.

The name of this attribute in the INI file is "file-style". Case is significant.

Definition at line 139 of file midimapper.hpp.

8.3.4.5 `int midipp::midimapper::m_filter_channel` [private]

Provides an optional channel number to filter in the MIDI file.

If this value is specified, then one of two kinds of channel filtering is turned on:

1. If `m_extraction_on` is true, then only channel-based events on the filtered channel will be written to the output MIDI file.
2. If `m_rejection_on` is true, then only channel-based events *not* on the filtered channel will be written to the output MIDI file.

This attribute is called "extract-channel" or "reject-channel". This is a command-line option. By default this value is -1, which means to not filter by channel.

Definition at line 223 of file midimapper.hpp.

8.3.4.6 `int midipp::midimapper::m_gm_channel` [private]

Provides the channel to use for General MIDI drums.

This value is usually 9, meaning MIDI channel 10. However, be careful, as externally, this value is always on a 1-16 scale, while internally it is reduced by 1 (a 0-15 scale) to save endless decrements.

The name of this attribute in the INI file is "gm-channel". Case is significant.

Definition at line 194 of file midimapper.hpp.

8.3.4.7 `bool midipp::midimapper::m_map_reversed` `[private]`

Indicates that the mapping should occur in the reverse direction.

That is, instead of mapping the notes from the device pitches and channel to General MIDI, the notes and channel should be mapped from General MIDI back to the device. This option is useful for playing back General MIDI files on old equipment.

Note that this option is an INI option ("reverse"), as well as a command-line option. It is specified by alternate means, such as a command-line parameter like "--reverse".

Definition at line 250 of file `midimapper.hpp`.

8.3.4.8 `std::string midipp::midimapper::m_map_type` `[private]`

Indicates what kind of mapping is allegedly provided by the file.

This can be one of the following values:

- "drum". The file describes mapping one pitch/channel to another pitch/channel, used mostly for coercing old drum machines to something akin to a General MIDI kit.
- "patch". The file describes program (patch) mappings, used to map old devices patch change values to General MIDI.
- "multi". The file describes both "drum" and "patch" mappings.

The name of this attribute in the INI file is "map-type". Case is significant.

Definition at line 168 of file `midimapper.hpp`.

8.3.4.9 `std::map<int, int> midipp::midimapper::m_patch_map` `[private]`

Provides the mapping between patches (programs).

If `m_map_reversed` is false, then the key is the GM patch/program, and the value is the device patch/program (which is the GM note needed to produce the same sound in GM as the device would have produced). If `m_map_reversed` is true, then the key is the GM patch/program, and the value is the device patch/program so that the MIDI file will be converted from GM mapping to device mapping.

Definition at line 275 of file `midimapper.hpp`.

8.3.4.10 `int midipp::midimapper::m_record_count` `[private]`

Provides the number of records (lines) or sections in the INI file.

Indicates the number of items being remapped.

This attribute ("record-count") does not appear in the INI file, as it is calculated as the file is read.

Warning

Only applies to "drum" mappings at present. MUST FIX!

Definition at line 181 of file `midimapper.hpp`.

8.3.4.11 `std::string midipp::midimapper::m_setup_name` `[private]`

Provides a nice tag name for the setup, nothing more.

It can be anything. A common value is the name of the program that the INI file is meant for.

The name of this attribute in the INI file is "setup-name". Case is significant.

Definition at line 150 of file midimapper.hpp.

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

- [midimapper.hpp](#)
- [midimapper.cpp](#)

8.4 midipp::stringmap< VALUETYPE > Class Template Reference

Provides an std::map wrapper geared towards using std::string as a key.

```
#include <stringmap.hpp>
```

Collaboration diagram for midipp::stringmap< VALUETYPE >:

midipp::stringmap< VALUETYPE >
- m_Name - m_Fields
+ stringmap() + stringmap() + stringmap() + operator=() + ~stringmap() + name() + insert() + insert() + value() + clear() and 8 more... # name()

Public Types

- typedef std::map< std::string, VALUETYPE > [Container](#)
Defines the type of container used by stringmap.
- typedef Container::const_iterator [const_iterator](#)
Provides a constant-iterator type for notational convenience.
- typedef Container::iterator [iterator](#)

Provides an iterator type for notational convenience.

- typedef std::pair< std::string,
VALUETYPE > pair

Provides a pair type for notational convenience.

Public Member Functions

- stringmap ()
Default constructor Creates an empty and unnamed Container.
- stringmap (const std::string &name)
Default constructor Creates an empty, but named Container.
- stringmap (const stringmap &source)
Copy constructor Copies the Container.
- stringmap & operator= (const stringmap &source)
- virtual ~stringmap ()
- const std::string & name () const
'Getter' function for member m_Name
- int insert (const std::string &key, const VALUETYPE &value)
Allows the insertion of a VALUETYPE object into the container.
- int insert (const VALUETYPE &value)
Manufactures a new key based on the current size of the container, and inserts the given value with this key.
- VALUETYPE value (const std::string &key) const
Provides a way to look up a string key and return a value.
- void clear ()
Allows the container to be emptied of VALUETYPE objects.
- iterator begin ()
- const_iterator begin () const
- iterator end ()
- const_iterator end () const
- size_t size () const
- bool empty () const
- iterator find (const std::string &key)
- const_iterator find (const std::string &key) const

Protected Member Functions

- void name (const std::string &n)
'Setter' function for member m_Name

Private Attributes

- std::string m_Name
Provides the name of the stringmap.
- Container m_Fields
Provides the actual container for which the stringmap template class is a wrapper.

8.4.1 Detailed Description

```
template<class VALUETYPE>class midipp::stringmap< VALUETYPE >
```

Provides an `std::map` wrapper geared towards using `std::string` as a key.

The problem with using integers is that they are not very easy to decipher while debugging, nor are they easy to understand in code (without using ugly `define-macros`).

Also, strings are a frequent lookup mechanism. A wrapper is needed so that we can disable the automatic insertion of objects that occurs when operator `[]` is applied to a key that does not yet exist in the `std::map` container.

Efficiency? Well, we should test that. :-) We're using copy semantic for the value part. If you use pointers for the value, the management of them is up to you!

This template class supports looking up a container of `VALUETYPE` objects by `std::string`.

The `VALUETYPE` must support and publicize the following operations:

- Default constructor. In addition, the default constructor should create an object that indicates an error state. This feature is necessary so that we can avoid throwing exceptions when doing map lookups.
- Copy constructor.
- Principal assignment operator.
- [show\(\)](#). The object must have its own overload of the global [show\(\)](#) function. See the [stringmap.cpp](#) module.

The type of container is given by the `Container` typedef. The `stringmap` template is essentially a wrapper for this class.

- Todo**
- Consider implementing lookup by integer index; right now, iterators suffice;
 - It would be nice to be able to superimpose a numeric order on the container, somehow.

Definition at line 82 of file `stringmap.hpp`.

8.4.2 Member Typedef Documentation

8.4.2.1 `template<class VALUETYPE > typedef std::map<std::string, VALUETYPE> midipp::stringmap< VALUETYPE >::Container`

Defines the type of container used by `stringmap`.

We want to be able to use array notation, yet be able to look up items (e.g. database fields) by name.

Definition at line 94 of file `stringmap.hpp`.

8.4.3 Constructor & Destructor Documentation

8.4.3.1 `template<class VALUETYPE > midipp::stringmap< VALUETYPE >::stringmap (const stringmap< VALUETYPE > & source) [inline]`

Copy constructor Copies the Container.

Parameters

<i>source</i>	The stringmap to be copied.
---------------	-----------------------------

Definition at line 162 of file stringmap.hpp.

8.4.3.2 `template<class VALUETYPE > virtual midipp::stringmap< VALUETYPE >::~~stringmap () [inline],
[virtual]`

Destructor:

Provided as a virtual destructor so that we can derive from this class.

Definition at line 198 of file stringmap.hpp.

8.4.4 Member Function Documentation

8.4.4.1 `template<class VALUETYPE > iterator midipp::stringmap< VALUETYPE >::begin () [inline]`

Accessor:

`m_Fields.begin()` This function makes this class look more like an STL container, good for using "for each" constructs.

Warning

Only a subset of `std::map` members are reimplemented in the stringmap class.

Returns

Returns an iterator for the first element of the container, if any. Otherwise `end()` is returned.

Definition at line 306 of file stringmap.hpp.

8.4.4.2 `template<class VALUETYPE > const_iterator midipp::stringmap< VALUETYPE >::begin () const
[inline]`

Accessor:

`m_Fields.begin() const` Makes this class look more like an STL container.

Returns

Returns a const iterator for the first element of the container, if any. Otherwise `end()` is returned.

Definition at line 320 of file stringmap.hpp.

8.4.4.3 `template<class VALUETYPE > bool midipp::stringmap< VALUETYPE >::empty () const [inline]`

Accessor:

`m_Fields.empty()`

Returns

Returns true if the container is empty.

Definition at line 370 of file stringmap.hpp.

8.4.4.4 `template<class VALUETYPE > iterator midipp::stringmap< VALUETYPE >::end () [inline]`

Accessor:

`m_Fields.end()` Makes this class look more like an STL container.

Returns

Returns an iterator indicating the end of the container.

Definition at line 333 of file stringmap.hpp.

8.4.4.5 `template<class VALUETYPE > const_iterator midipp::stringmap< VALUETYPE >::end () const [inline]`

Accessor:

`m_Fields.end() const` Makes this class look more like an STL container.

Returns

Returns a const iterator indicating the end of the container.

Definition at line 346 of file stringmap.hpp.

8.4.4.6 `template<class VALUETYPE > iterator midipp::stringmap< VALUETYPE >::find (const std::string & key) [inline]`

Accessor:

`m_Fields.find()` This function makes this class look more like an STL container.

Returns

Returns an iterator for the found element of the container, if any. Otherwise `end()` is returned.

Definition at line 384 of file stringmap.hpp.

8.4.4.7 `template<class VALUETYPE > const_iterator midipp::stringmap< VALUETYPE >::find (const std::string & key) const [inline]`

Accessor:

`m_Fields.find() const` Makes this class look more like an STL container.

Returns

Returns a const iterator for the found element of the container, if any. Otherwise `end()` is returned.

Definition at line 398 of file stringmap.hpp.

8.4.4.8 `template<class VALUETYPE > int midipp::stringmap< VALUETYPE >::insert (const std::string & key, const VALUETYPE & value) [inline]`

Allows the insertion of a VALUETYPE object into the container.

Parameters

<i>key</i>	The string that is to serve as the lookup value for the inserted object.
<i>value</i>	The value object to be added to the container.

Returns

The size of the container after insertion is returned. If important, the caller should check that the size is one larger.

Definition at line 227 of file stringmap.hpp.

8.4.4.9 `template<class VALUETYPE > int midipp::stringmap< VALUETYPE >::insert (const VALUETYPE & value)`
`[inline]`

Manufactures a new key based on the current size of the container, and inserts the given value with this key.

This function is meant to parallel the integer-key version of the `insert()` function. It could also be called `append()`.

Parameters

<i>value</i>	The value object to be added to the container.
--------------	--

Returns

The size of the container after insertion is returned.

Definition at line 247 of file stringmap.hpp.

8.4.4.10 `template<class VALUETYPE > stringmap& midipp::stringmap< VALUETYPE >::operator= (const stringmap< VALUETYPE > & source)`
`[inline]`

Principal Assignment Operator:

Copies one Container into the current object's Container.

Parameters

<i>source</i>	The stringmap to be copied into the current stringmap.
---------------	--

Returns

A reference to the destination object is returned, in order to support multiple assignments in one statement.

Definition at line 182 of file stringmap.hpp.

8.4.4.11 `template<class VALUETYPE > size_t midipp::stringmap< VALUETYPE >::size () const`
`[inline]`

Accessor:

`m_Fields.size()`

Returns

Returns the number of VALUETYPE objects in the container.

Definition at line 358 of file stringmap.hpp.

8.4.4.12 `template<class VALUETYPE > VALUETYPE midipp::stringmap< VALUETYPE >::value (const std::string & key) const [inline]`

Provides a way to look up a string key and return a value.

We can't just return "m_Fields[key]" because that causes an insertion if the key doesn't exist in the map. That is not the semantics we want.

Note

This statement can change the container, so is not const.

```
return m_Fields[key];
```

Parameters

<i>key</i>	Provides the string key to be looked up.
------------	--

Returns

Returns the VALUETYPE found. If it was not found, then a default-constructor VALUETYPE is returned. It is the author's responsibility to provide an error-checking facility in VALUETYPE.

Definition at line 274 of file stringmap.hpp.

8.4.5 Field Documentation

8.4.5.1 `template<class VALUETYPE > std::string midipp::stringmap< VALUETYPE >::m_Name [private]`

Provides the name of the stringmap.

This name can be used to add the stringmap to a stringmap full of stringmaps.

Definition at line 123 of file stringmap.hpp.

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

- [stringmap.hpp](#)

Chapter 9

File Documentation

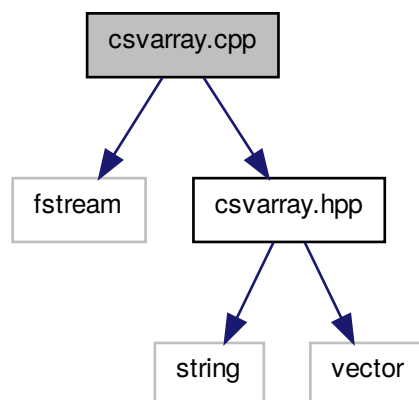
9.1 csvarray.cpp File Reference

Primitive, but useful, CSV-file input parser.

```
#include <fstream>
```

```
#include <csvarray.hpp>
```

Include dependency graph for csvarray.cpp:



Macros

- `#define` `TOKEN_SPACES` `" \t\\0"`

Provides a consistent definition of the whitespace characters.

Functions

- `void` `show` (`const` `std::string` &tag, `const` `midipp::csvarray` &container)

Writes out the contents of the csvarray container to stderr.

9.1.1 Detailed Description

Primitive, but useful, CSV-file input parser.

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-23

Last Edits:

2014-05-21

Version

\$Revision\$

License:

\$XPC_SUITE_GPL_LICENSE\$

Provides a way to convert a simple CSV (comma-separated value) file into an array of data.

Warning

1. This module assumes only ASCII text. This is a big limitation for internationalization.

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

9.1.2 Macro Definition Documentation

9.1.2.1 `#define TOKEN_SPACES " \t0"`

Provides a consistent definition of the whitespace characters.

This define does not bother to try to trap all the characters that could be considered whitespace or outside the range of legal section and option names.

Definition at line 32 of file `csvarray.cpp`.

9.1.3 Function Documentation

9.1.3.1 `void show (const std::string & tag, const midipp::csvarray & container)`

Writes out the contents of the csvarray container to stderr.

This implementation is a `for_each` style of looping through the container.

Parameters

<i>tag</i>	Identifies the object in the human-readable output.
<i>container</i>	The stringmap through which iteration is done for showing.

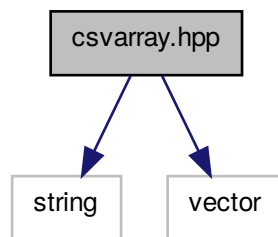
Definition at line 240 of file csvarray.cpp.

9.2 csvarray.hpp File Reference

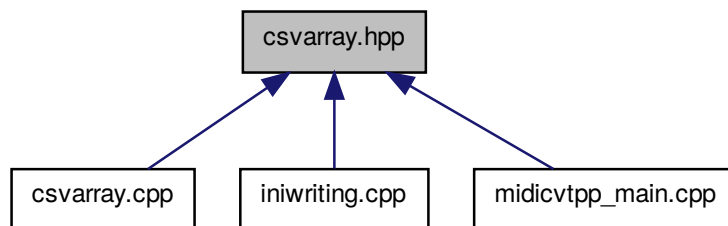
```
#include <string>
```

```
#include <vector>
```

Include dependency graph for csvarray.hpp:



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



Data Structures

- class [midipp::csvarray](#)

The csvarray class assist in parsing a file that has lines of comma-separated values.

Functions

- void [show](#) (const std::string &tag, const [midipp::csvarray](#) &container)

Writes out the contents of the csvarray container to stderr.

9.2.1 Detailed Description

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-23

Last Edits:

2014-05-21

Version

\$Revision\$

License:

\$MIDIPP_SUITE_GPL_LICENSE\$

Provides a way to convert a simple CSV (comma-separated value) file into an array of data, and into an INI format.

The input CSV file is an unadorned file. The caller must know what the file is for, so that the results can be properly annotated.

Definition in file [csvarray.hpp](#).

9.2.2 Function Documentation

9.2.2.1 void show (const std::string & tag, const midipp::csvarray & container)

Writes out the contents of the csvarray container to stderr.

This implementation is a for_each style of looping through the container.

Parameters

<i>tag</i>	Identifies the object in the human-readable output.
<i>container</i>	The stringmap through which iteration is done for showing.

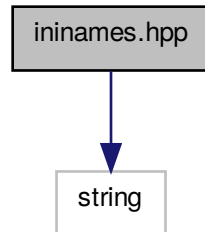
Definition at line 240 of file csvarray.cpp.

9.3 ininames.hpp File Reference

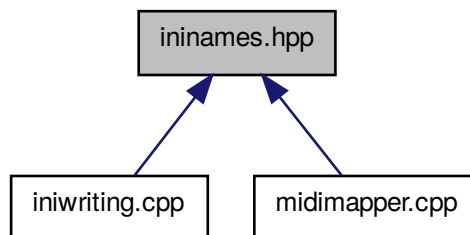
Provides some free functions to write INI files, given a [midipp::csvarray](#) object.

```
#include <string>
```

Include dependency graph for ininames.hpp:



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



Enumerations

- enum **gm_ini_section_t** {
midipp::INI_SECTION_DRUM,
midipp::INI_SECTION_PATCH,
midipp::INI_SECTION_CHANNEL,
midipp::INI_SECTION_UNKNOWN }

Provides a selection switch for the supported INI section types.

Variables

- const std::string **midipp::DRUM_SECTION** = "Drum"
Provides the prefix for the initial part of the drum-section name.
- const std::string **midipp::DRUM_LABEL_GM_NAME** = "gm-name"
Provides the string for the name of the DRUM_INDEX_GM_NAME field as shown in the INI file.
- const std::string **midipp::DRUM_LABEL_GM_NOTE** = "gm-note"
Provides the string for the name of the DRUM_INDEX_GM_NOTE field as shown in the INI file.
- const std::string **midipp::DRUM_LABEL_DEV_NAME** = "dev-name"

- Provides the string for the name of the DRUM_INDEX_DEV_NAME field as shown in the INI file.*
- `const std::string midipp::DRUM_LABEL_DEV_NOTE = "dev-note"`
- Provides the string for the name of the DRUM_INDEX_DEV_NAME field as shown in the INI file.*
- `const std::string midipp::DRUM_LABEL_GM_EQUIV = "gm-equiv"`
- Provides the string for the name of the DRUM_INDEX_GM_EQUIV field as shown in the INI file.*
- `const std::string midipp::PATCH_SECTION = "Patch"`
- Provides the prefix for the initial part of the patch-section name.*
- `const std::string midipp::PATCH_LABEL_GM_NAME = "gm-name"`
- Provides the string for the name of the PATCH_INDEX_GM_NAME field as shown in the INI file.*
- `const std::string midipp::PATCH_LABEL_GM_PATCH = "gm-patch"`
- Provides the string for the name of the PATCH_INDEX_GM_PATCH field as shown in the INI file.*
- `const std::string midipp::PATCH_LABEL_DEV_NAME = "dev-name"`
- Provides the string for the name of the PATCH_INDEX_DEV_NAME field as shown in the INI file.*
- `const std::string midipp::PATCH_LABEL_DEV_PATCH = "dev-patch"`
- Provides the string for the name of the PATCH_INDEX_DEV_NAME field as shown in the INI file.*
- `const std::string midipp::PATCH_LABEL_GM_EQUIV = "gm-equiv"`
- Provides the string for the name of the PATCH_INDEX_GM_EQUIV field as shown in the INI file.*
- `const std::string midipp::CHANNEL_SECTION = "Channels"`
- Provides the prefix for the channel-section name.*
- `const std::string midipp::CHANNEL_TOKEN = "ch_"`
- Provides the initial part of the name for each entry in the channel-section.*
- `const std::string midipp::GM_INI_FILE_STYLE = "file-style"`
- Provides a consistent string for the file-style attribute.*
- `const std::string midipp::GM_INI_SETUP_NAME = "setup-name"`
- Provides a consistent string for the setup-name attribute.*
- `const std::string midipp::GM_INI_MAP_TYPE = "map-type"`
- Provides a consistent string for the map-type attribute.*
- `const std::string midipp::GM_INI_GM_CHANNEL = "gm-channel"`
- Provides a consistent string for the GM-channel attribute.*
- `const std::string midipp::GM_INI_DEV_CHANNEL = "dev-channel"`
- Provides a consistent string for the device-channel attribute.*
- `const std::string midipp::GM_INI_EXTRACT_CHANNEL = "extract-channel"`
- Provides a consistent string for the extract-channel attribute.*
- `const std::string midipp::GM_INI_REJECT_CHANNEL = "reject-channel"`
- Provides a consistent string for the reject-channel attribute.*
- `const std::string midipp::GM_INI_REVERSE = "reverse"`
- Provides a consistent string for the reverse attribute.*
- `const std::string midipp::GM_INI_TESTING = "testing"`
- Provides a consistent string for the testing attribute.*
- `const std::string midipp::GM_INI_NO_VALUE = "none"`
- Provides a consistent string for the "none" value.*

9.3.1 Detailed Description

Provides some free functions to write INI files, given a `midipp::csvarray` object.

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-05-06

Last Edits:

2014-05-17

Version

\$Revision\$

License:

\$XPC_SUITE_GPL_LICENSE\$

Provides a way to write various kinds of MIDI-related INI files.

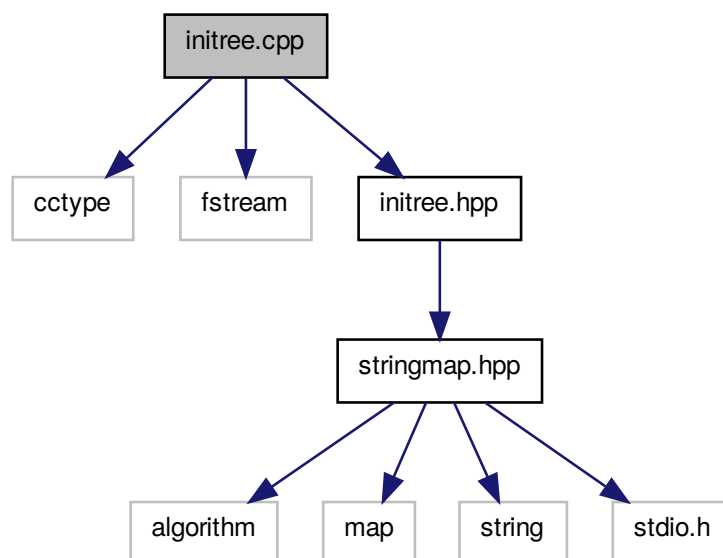
Also provides some names and macros.

Definition in file [ininames.hpp](#).

9.4 initree.cpp File Reference

Primitive INI-file input parser derived from the XPC library.

```
#include <cctype>
#include <fstream>
#include <initree.hpp>
Include dependency graph for initree.cpp:
```



Functions

- void [show](#) (const std::string &tag, const [midipp::initree](#) &container)
Writes out the contents of the stringmap container.

9.4.1 Detailed Description

Primitive INI-file input parser derived from the XPC library.

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-22

Last Edits:

2014-05-21

Version

\$Revision\$

License:

\$XPC_SUITE_GPL_LICENSE\$

Provides a way to set the application options from a Windows-style "INI" file, by parsing that configuration file and making it available as if it were a set of command-line parameters.

The file is in simple INI format. Each line is one of the following:

- *Blank. Blank lines are ignored.*
- *Comment. Lines whose first non-white-space character is one of hash-mark, semi-colon, exclamation-point, single-quote, or double-quote, are ignored.*
- *Section. "[Section Name]". If present, this item is used to indicate which group of initializations are acceptable to the caller.*
- *Option + value. optionname = optionvalue.*
- *Option without value. This item is treated like a simple flag, since there is no value specified.*

Warning

1. *This module assumes only ASCII text. This is a big limitation and severely hurts internationalization.*
2. *The code cannot really handle mixes of quotes and comment characters on an option line. So, generally prefer blocks of comments between options, rather than trailing comments on an option line. See the tests/initree.ini file.*
3. *Comments are not saved anywhere, so when we get around to writing out an INI file, they will get dropped.*

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

9.4.2 Function Documentation

9.4.2.1 void show (const std::string & tag, const midipp::initree & container)

Writes out the contents of the stringmap container.

This implementation is a for_each style of looping through the container.

Parameters

<i>tag</i>	Identifies the object in the human-readable output.
<i>container</i>	The stringmap through which iteration is done for showing.

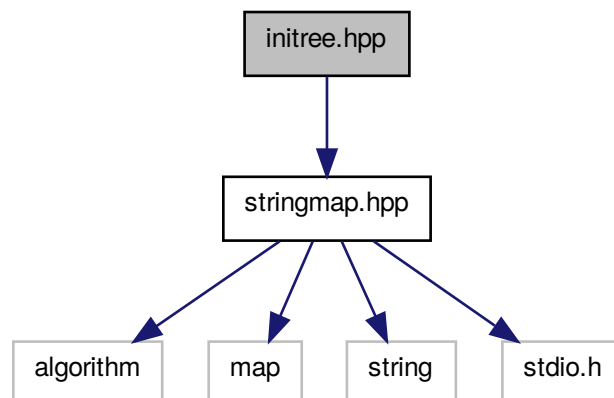
Definition at line 533 of file initree.cpp.

9.5 initree.hpp File Reference

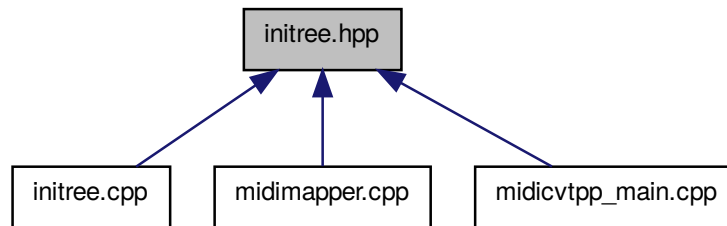
Provides a "tree" that represents an INI file with named and unnamed sections, and name/value pairs.

```
#include <stringmap.hpp>
```

Include dependency graph for initree.hpp:



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



Data Structures

- class [midipp::initree](#)

This class provides a way to read and represent an INI file as a kind of tree structure that can be navigate to look up information.

Functions

- void [show](#) (const std::string &tag, const [midipp::initree](#) &container)

Writes out the contents of the stringmap container.

9.5.1 Detailed Description

Provides a "tree" that represents an INI file with named and unnamed sections, and name/value pairs.

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-22

Last Edits:

2014-05-21

Version

\$Revision\$

License:

\$MIDIPP_SUITE_GPL_LICENSE\$

Provides a way to create an options tree using a DOS/Windows INI-style configuration file.

Definition in file [initree.hpp](#).

9.5.2 Function Documentation

9.5.2.1 void show (const std::string & tag, const midipp::initree & container)

Writes out the contents of the stringmap container.

This implementation is a for_each style of looping through the container.

Parameters

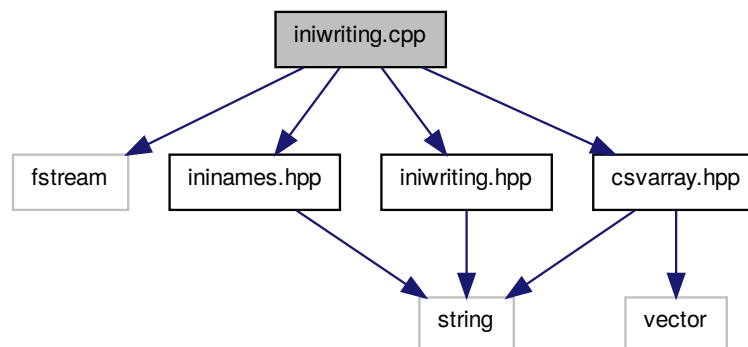
<i>tag</i>	Identifies the object in the human-readable output.
<i>container</i>	The stringmap through which iteration is done for showing.

Definition at line 533 of file initree.cpp.

9.6 iniwriting.cpp File Reference

Provides some free functions to write INI files, given a [midipp::csvarray](#) object.

```
#include <fstream>
#include <csvarray.hpp>
#include <ininames.hpp>
#include <iniwriting.hpp>
Include dependency graph for iniwriting.cpp:
```



Macros

- `#define _NL std::endl` /* defined for brevity/formatting */
Provides a less cluttered way to add a newline to string output.

Functions

- `bool midipp::write_simple_drum_file` (const std::string &filespec, const csvarray &csv)
Writes a csvarray object to a file in a simple INI format.
- `bool midipp::write_sectioned_drum_file` (const std::string &filespec, const csvarray &csv, bool writefooter)
Writes a csvarray object to a drum file in a sectioned INI format.
- `bool midipp::write_sectioned_patch_file` (const std::string &filespec, const csvarray &csv, bool write-header)

Writes a `csvarray` object to a patch file in a sectioned INI format.

Variables

- static const std::string **midipp::s_explanatory_header**

Provides explanatory text for the top of the INI file.

9.6.1 Detailed Description

Provides some free functions to write INI files, given a [midipp::csvarray](#) object.

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-23

Last Edits:

2014-05-21

Version

\$Revision\$

License:

\$XPC_SUITE_GPL_LICENSE\$

Provides a way to create INI files from the contents of a [midipp::csvarray](#) object.

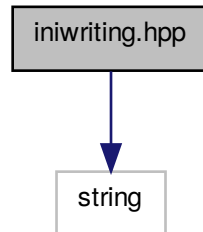
Definition in file [iniwriting.cpp](#).

9.7 iniwriting.hpp File Reference

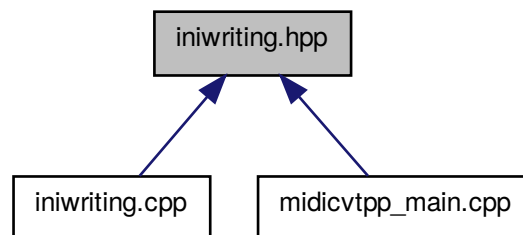
Provides some free functions to write INI files, given a [midipp::csvarray](#) object.

```
#include <string>
```

Include dependency graph for iniwriting.hpp:



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



Functions

- **bool midipp::write_simple_drum_file** (const std::string &filespec, const csvarray &csv)
Writes a csvarray object to a file in a simple INI format.
- **bool midipp::write_sectioned_drum_file** (const std::string &filespec, const csvarray &csv, bool writefooter)
Writes a csvarray object to a drum file in a sectioned INI format.
- **bool midipp::write_sectioned_patch_file** (const std::string &filespec, const csvarray &csv, bool writeheader)
Writes a csvarray object to a patch file in a sectioned INI format.

9.7.1 Detailed Description

Provides some free functions to write INI files, given a [midipp::csvarray](#) object.

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-23

Last Edits:

2014-05-08

Version

\$Revision\$

License:

\$XPC_SUITE_GPL_LICENSE\$

Provides a way to write various kinds of MIDI-related INI files.

Also provides some names and macros.

Definition in file [iniwriting.hpp](#).

9.8 mainpage-reference.dox File Reference

This document describes the modules and functions of the libmidifilex library and the midicvt and midicvtp applications.

9.8.1 Detailed Description

This document describes the modules and functions of the libmidifilex library and the midicvt and midicvtp applications.

Definition in file [mainpage-reference.dox](#).

9.9 midi_functions.dox File Reference

This document describes the functions and parameters of the libmidifilex modules.

9.9.1 Detailed Description

This document describes the functions and parameters of the libmidifilex modules.

Definition in file [midi_functions.dox](#).

9.10 midicvt_base.c File Reference

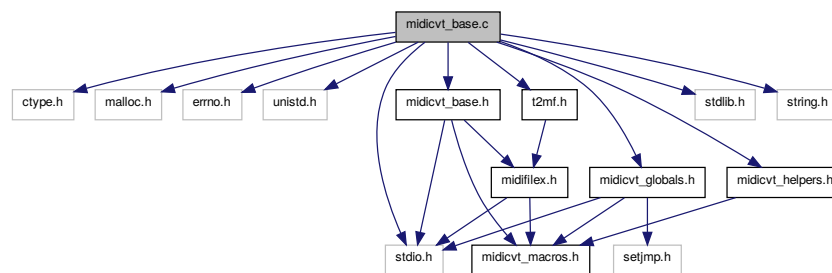
This module provides functions for basic MIDI/text conversions.

```

#include <ctype.h>
#include <malloc.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <midicvt_base.h>
#include <midicvt_globals.h>
#include <midicvt_helpers.h>
#include <t2mf.h>

```

Include dependency graph for midicvt_base.c:



Functions

- static void [checkchan](#) (void)
Checks to make sure that yylex() == CH or yylex() == INT, and yyval >= 1 or yyval 16.
- static void [checknote](#) (void)
Checks the incoming value from flex.
- static int [fileputc](#) (unsigned char c)
Callback function implementing Mf_putc().
- static int [filegetc](#) (void)
Callback function implementing Mf_getc().
- static int [getbyte](#) (char *mess)
Gets the current byte value as contained in yyval.
- static int [getint](#) (char *mess)
Returns the current byte value as contained in yyval.
- static void [get16val](#) (void)
- static int [prs_error](#) (char *s)
Outputs error messages to standard error.
- static void [syntax](#) (void)
Calls prs_error() with a message of "Syntax error".
- void [error](#) (const char *s)
Writes an obvious error string to standard error.
- static void [ptime](#) (void)
Writes the time of the current MIDI event to standard output in text format.
- static void [prtext](#) (unsigned char *p, int leng)
Writes the provided text to standard output.
- static void [prhex](#) (unsigned char *p, int leng)
Writes the provided text to standard output in hexadecimal format.

- static char * [prnote](#) (int pitch)
Writes a note value to standard output.
- static int [my_header](#) (int format, int ntrks, int division)
Callback function implementing Mf_header().
- static int [my_trstart](#) (void)
Callback function implementing Mf_starttrack().
- static int [my_trend](#) (long header_offset, unsigned long track_size)
Callback function implementing Mf_endtrack().
- static int [my_non](#) (int chan, int pitch, int vol)
Callback function implementing Mf_on().
- static int [my_noff](#) (int chan, int pitch, int vol)
Callback function implementing Mf_off().
- static int [my_pressure](#) (int chan, int pitch, int pressure)
Callback function implementing Mf_pressure().
- static int [my_parameter](#) (int chan, int control, int value)
Callback function implementing Mf_parameter().
- static int [my_pitchbend](#) (int chan, int lsb, int msb)
Callback function implementing Mf_pitchbend().
- static int [my_program](#) (int chan, int program)
Callback function implementing Mf_program().
- static int [my_chanpressure](#) (int chan, int pressure)
Callback function implementing Mf_chanpressure().
- static int [my_sysex](#) (int leng, char *mess)
Callback function implementing Mf_sysex().
- static int [my_mmisc](#) (int typecode, int leng, char *mess)
Callback function implementing Mf_metamisc().
- static int [my_mspecial](#) (int leng, char *mess)
Callback function implementing Mf_sqspecific().
- static int [my_mtext](#) (int type, int leng, char *mess)
Callback function implementing Mf_text().
- static int [my_mseq](#) (short int num)
Callback function implementing Mf_seqnum().
- static int [my_meot](#) (void)
Callback function implementing Mf_eot().
- static int [my_keysig](#) (int sf, int mi)
Callback function implementing Mf_keysig().
- static int [my_tempo](#) (long tempo)
Callback function implementing Mf_tempo().
- static int [my_timesig](#) (int nn, int dd, int cc, int bb)
Callback function implementing Mf_timesig().
- static int [my_smppte](#) (int hr, int mn, int se, int fr, int ff)
Callback function implementing Mf_smppte().
- static int [my_arbitrary](#) (int leng, char *mess)
Callback function implementing Mf_arbitrary().
- static int [my_error](#) (const char *s)
Callback function implementing Mf_error().
- static int [my_writetrack](#) (void)
Callback function implementing Mf_wtrack() and Mf_wtempotrack().
- void [midicvt_initfuncs_t2mf](#) (void)
Makes the function assignments needed by the midifile library when converting a text file to MIDI.
- void [midicvt_initfuncs_mf2t](#) (void)

Makes the function assignments needed by the midifile library when converting a MIDI file to text.

- void `midicvt_compile` (void)

This function makes sure the MFile token is found.

- static `cbool_t redirect_stdout` (const char *filename, const char *mode)

Redirects stdout to a file.

- static `cbool_t revert_stdout` ()

Holds the file descriptor for stdout for later restoration.

- `cbool_t midicvt_setup_compile` (void)

Sets up the incoming text file for compiling into MIDI.

- void `midicvt_close_compile` (void)

Closes the input file (yyin) and the output file (g_io_file) for the `-compile` option.

- `cbool_t midicvt_setup_mfread` (void)

Sets up the file handles for both reading and writing.

- void `midicvt_close_mfread` (void)

Checks g_io_file for an error status, closes this file handle if necessary, and restores stdout to norml if necessary.

Variables

- static unsigned char `gs_data` [5]

Holds up to four bytes of data that comprise the data values for a MIDI operation.

- static int `gs_chan` = 0

Holds the channel number of the MIDI channel currently being processed.

- static int `gs_saved_stdout` = -1

Holds the file descriptor for stdout for later restoration.

- static `fpos_t gs_saved_stdout_pos`

Holds the file position for stdout for later restoration.

9.10.1 Detailed Description

This module provides functions for basic MIDI/text conversions.

Library:

midicvt application

Author

Chris Ahlstrom and many other authors

Date

2014-04-09

Last Edits:

2014-05-21

Version`$Revision$`**License:**

GNU GPL

Summary of MIDI messages:

We want to provide a very brief reference to the MIDI messages handled by the library and application in this project.

Channel Voice Messages.

The first nybble is the message (e.g. note on), and the second nybble (denoted by "c") is the channel number.

One or more parameter bytes follow, and the first bit is 0, so that the values range from 0 to 127.

"kk" means "key" or note number, "vv" means "velocity" or some other "value", "pp" means "program" or "patch". Other variations are noted. All didits are hexadecimal.

- 8c kk vv. Note Off.
- 9c kk vv. Note On.
- Ac kk vv. Aftertouch (Polyphonic Key Pressure).
- Bc cc vv. Control (cc) Change.
- Cc pp . Program/patch (cc) Change.
- Dc pp . Aftertouch (Channel Pressure).
- Ec ll mm. Pitch Wheel Change. Provides the least significant and most significant 14 bits.
- Fc mm . System Common Messages. See below.

System Common Messages:

- | | |
|---|--------------------------|
| - <code><code> F0 </code>.</code> | System Exclusive. |
| - <code><code> F1 </code>.</code> | Undefined. |
| - <code><code> F2 </code>.</code> | Song Position Pointer. |
| - <code><code> F3 </code>.</code> | Song Select. |
| - <code><code> F4 </code>.</code> | Undefined. |
| - <code><code> F5 </code>.</code> | Undefined. |
| - <code><code> F6 </code>.</code> | Tune Request. |
| - <code><code> F7 </code>.</code> | End-Of-Exclusive marker. |

System Real-Time Messages:

- | | |
|---|---|
| - <code><code> F8 </code>.</code> | Timing Clock. |
| - <code><code> F9 </code>.</code> | Undefined. |
| - <code><code> FA </code>.</code> | Start. |
| - <code><code> FB </code>.</code> | Continue. |
| - <code><code> FC </code>.</code> | Stop. |
| - <code><code> FD </code>.</code> | Undefined. |
| - <code><code> FE </code>.</code> | Active Sensing. |
| - <code><code> FF </code>.</code> | Reset. In a MIDI file, this code is used as an escape character to introduce "meta events". |

Meta Events:

"ln" indicates the length byte. "tx" indicates all of the bytes that are part of the text of the meta-event. This text is ASCII in some cases, binary in the case of sequencer-specific data.

- FF 00 02 ss ss or FF 00 00. Sequence Number. Optional, but must occur at the beginning of an MTrk.

- FF 01 ln tx . Text Event.
- FF 02 ln tx . Copyright Notice.
- FF 03 ln tx . Sequence/Track Name.
- FF 04 ln tx . Instrument Name.
- FF 05 ln tx . Lyric.
- FF 06 ln tx . Marker.
- FF 07 ln tx . Cue Point.
- FF 08 ln tx to FF 0F ln tx . Indeterminate/unrecognized text event.
- FF 20 01 cc . MIDI Channel Prefix.
- FF 2F 00 . End of Track.
- FF 51 03 tt tt tt. Set tempo.
- FF 54 05 hr mn se fr ff. SMPTE Offset.
- FF 58 04 nn dd cc bb. Time Signature.
- FF 59 02 sf mi . Key Signature.
- FF 7F ln tx . Sequencer-specific binary event.

That's all for now.

Definition in file [midicvt_base.c](#).

9.10.2 Function Documentation

9.10.2.1 static void checkchan (void) [static]

Checks to make sure that yylex() == CH or yylex() == INT, and yyval >= 1 or yyval 16.

Side-effect(s):

The global variable gs_chan is set to yyval - 1.

Definition at line 1698 of file midicvt_base.c.

9.10.2.2 void error (const char * s)

Writes an obvious error string to standard error.

Public Note that this is an public function used in other modules, and declared in the [midicvt_base.h](#) file.

Parameters

s	Provides the null-terminated error message to be written.
---	---

Definition at line 161 of file midicvt_base.c.

9.10.2.3 `static int filegetc (void) [static]`

Callback function implementing [Mf_getc\(\)](#).

This function increments the offset into the MIDI file, and then calls `getc(g_io_file)`.

Returns

Returns the value returned by `getc()`.

Definition at line 2044 of file `midicvt_base.c`.

9.10.2.4 `static void get16val (void) [static]`

Question:

Does this function need to have and use a char pointer???

Definition at line 1804 of file `midicvt_base.c`.

9.10.2.5 `static int getbyte (char * mess) [static]`

Gets the current byte value as contained in `yyval`.

Parameters

<i>mess</i>	The message to print if <code>yyval</code> is greater than 127.
-------------	---

Returns

The value of `yyval` is returned. It is set to 0 if an error occurred.

Definition at line 1644 of file `midicvt_base.c`.

9.10.2.6 `static int getint (char * mess) [static]`

Returns the current byte value as contained in `yyval`.

Parameters

<i>mess</i>	The message to print if <code>yylex()</code> is not equal to <code>INT</code> .
-------------	---

Returns

The value of `yyval` is returned. It is set to 0 if an error occurred.

Definition at line 1677 of file `midicvt_base.c`.

9.10.2.7 `void midicvt_compile (void)`

This function makes sure the MFile token is found.

It then gathers up some status information and passes it to [mfwrite\(\)](#).

Note

This function used to be called `translate()`, which was a bit ambiguous.

Definition at line 1611 of file `midicvt_base.c`.

9.10.2.8 cbool_t midicvt_setup_compile (void)

Sets up the incoming text file for compiling into MIDI.

Returns

Returns true if all is well.

Definition at line 2143 of file midicvt_base.c.

9.10.2.9 cbool_t midicvt_setup_mfread (void)

Sets up the file handles for both reading and writing.

Returns

Returns true if all steps succeeded.

Definition at line 2207 of file midicvt_base.c.

9.10.2.10 static int my_arbitrary (int *leng*, char * *mess*) [static]

Callback function implementing Mf_arbitrary().

This function is called in [readtrack\(\)](#) if there are additional bytes that are not part of a system exclusive continuation.

I have to confess that, right now, I don't know what that means.

This function writes the time, "Arb", followed by the message bytes, followed by the length. Why doesn't the length come first?

Parameters

<i>leng</i>	Provides the number of bytes in the message.
<i>mess</i>	Points to the bytes in the message.

Returns

Returns true, always.

Definition at line 1181 of file midicvt_base.c.

9.10.2.11 static int my_chanpressure (int *chan*, int *pressure*) [static]

Callback function implementing Mf_chanpressure().

Command: 0xDn .

A channel-pressure command has the following 3 bytes:

1. Delta-time byte. A typical value is 0x60 (96).
2. Channel-pressure byte. 0xDn .
3. Channel-pressure value byte. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pressure</i>	Provides the patch/program number to write.

Returns

Returns true, always.

Definition at line 749 of file `midicvt_base.c`.

9.10.2.12 static int my_error (const char * s) [static]

Callback function implementing `Mf_error()`.

This fuction checks for garbage at the end of the MIDI file. If it does not find such garbage, then the error is simply printed.

Parameters

<i>s</i>	Provides the error string to report.
----------	--------------------------------------

Returns

Returns true, always.

Definition at line 1203 of file `midicvt_base.c`.

9.10.2.13 static int my_header (int format, int ntrks, int division) [static]

Callback function implementing `Mf_header()`.

Provides a libmidifilex callback function to write the MIDI header data to standard output in text format.

This function is called by [readheader\(\)](#) in `midifilex.c`, which reads:

1. "MThd" marker, 4 bytes, unterminated
2. Size (32-bits)
3. MIDI format (16-bits)
4. Number of tracks (16-bits)
5. Time division (16-bits)

and then calls this callback function, after which it flushes and additional bytes specified by the size parameter.

Parameters

<i>format</i>	Provides the byte describing the format of the MIDI file. This value will be 0, 1, or 2.
<i>ntrks</i>	Provides the byte describing the number of tracks in the MIDI file. This value ranges from 1 to 65536. SMF 0 files will have only 1 track. SMF 1 has multiple tracks, with the first track containing song information.

<i>division</i>	Provides the time-division value. The first byte is either 0 (indicates the time format is ticks per quarter-note) or 1 (the time format is negative SMPTE). The second byte is the number of "ticks" per frame. For example, 0x80 indicates 128 ticks per frame.
-----------------	---

Returns

Returns true, always.

Definition at line 424 of file midicvt_base.c.

9.10.2.14 static int my_keysig (int *sf*, int *mi*) [static]

Callback function implementing Mf_keysig().

This function prints "KeySig", followed by the *sf* parameter, followed by "major" or "minor", to standard output.

Command: FF 59 02 *sf* *mi* .

The format of this event is:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. End-of-track marker. 0x59 .
4. Length byte. 0x02 .
5. Data bytes:
 - (a) *sf* = signature flats
 - -7: 7 flats
 - -1: 7 flat
 - 0: Key of C
 - 1: 1 sharp
 - 7: 7 sharps
 - (b) *mi* = minor flag (0 = major; 1 = minor)

Parameters

<i>sf</i>	Provides the code for the keys above and below C. (I don't really understand what that mean, right now.)
<i>mi</i>	True (1) indicates the key is minor; otherwise it is major.

Returns

Returns true, always.

Definition at line 1022 of file midicvt_base.c.

9.10.2.15 static int my_meot (void) [static]

Callback function implementing Mf_eot().

This function writes "Mata TrkEnd" to standard output.

Command: FF 00 .

The format of this event is:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. End-of-track marker. 0x0F .
4. Length byte. 0x00 .

There is no data for this command, so the length byte is 0.

Returns

Returns true, always.

Definition at line 980 of file midicvt_base.c.

9.10.2.16 `static int my_mmisc (int typecode, int leng, char * mess)` `[static]`

Callback function implementing Mf_metamisc().

This function prints "Meta", "0xnn" (the type of event), the length of the event as a byte, and then a stream of bytes, in hex format, to standard output.

Command: 0xFF .

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. Event-type byte. A 7-bit value, highest bit is 0. These range from 01 to 08 for the known text events. See the list of other values else where in this document.
4. Length byte.
5. Data bytes. Delimited by the length byte.

Parameters

<i>typecode</i>	Provides the type of meta event.
<i>leng</i>	Provides the number of bytes in the event.
<i>mess</i>	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 824 of file midicvt_base.c.

9.10.2.17 `static int my_mseq (short int num)` `[static]`

Callback function implementing Mf_seqnum().

This function writes "SeqNr" and the sequence number to standard output.

Command: FF 00 02 .

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).

2. Meta-event byte. 0xFF .
3. Sequence-number marker. 0x00 .
4. Length byte. 0x02 .
5. Data bytes. Two bytes of data, I believe, at this time. As for ordering

Parameters

<i>num</i>	Provides the number to be converted to a sequence-number specification.
------------	---

Returns

Returns true, always.

Definition at line 952 of file midicvt_base.c.

9.10.2.18 static int my_mspecial (int *leng*, char * *mess*) [static]

Callback function implementing Mf_sqspecific().

This function prints "SeqSpec", the size of the event as a byte, and then a stream of bytes, in hex format, to standard output.

Command: 0xFF 0x7F

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. Event-type byte. 0x7F .
4. Length byte.
5. Data bytes. Delimited by the length byte.

Parameters

<i>leng</i>	Provides the number of bytes in the event.
<i>mess</i>	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 859 of file midicvt_base.c.

9.10.2.19 static int my_mtext (int *type*, int *leng*, char * *mess*) [static]

Callback function implementing Mf_text().

This function prints "SeqSpec", the size of the event as a byte, and then a stream of bytes, in hex format, to standard output.

Command: 0xFF .

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).

2. Meta-event byte. `0xFF` .
3. Event-type byte. `0xnn` .
4. Length byte.
5. Data bytes. Delimited by the length byte.

Parameters

<i>type</i>	Here, nn ranges from 01 to 0F. 01 to 08 are the known types of text event, while 09 to 0F are unrecognized text events. Other values are treat as in my_mmisc() .
<i>leng</i>	Provides the number of bytes in the event.
<i>mess</i>	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 899 of file `midicvt_base.c`.

9.10.2.20 `static int my_noff (int chan, int pitch, int vol)` `[static]`

Callback function implementing `Mf_off()`.

Command: `0x8n` .

Very similar to Note-On messages [see [my_non\(\)](#)].

In our sample file, `ex1.mid`, there are no Note-Off messages, only Note-On messages with a velocity of 0.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>vol</i>	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 584 of file `midicvt_base.c`.

9.10.2.21 `static int my_non (int chan, int pitch, int vol)` `[static]`

Callback function implementing `Mf_on()`.

Command: `0x9n` .

This function outputs the time using [ptime\(\)](#), then writes the channel, note, and velocity using the `g_option_Onmsg` format specifier.

The MIDI bytes for a note on message are 4 in number:

1. Delta-time byte. A typical value is `0x60` (96).
2. Note on byte. `0x9n` . This byte is a status byte having two parts:
 - The first (most significant) nybble of this byte is `0x9`.
 - The least significant nybble holds the channel number, ranging from `0x0` to `0xF`. However, in our sample file, `ex1.mid`, there see to be notes in which the note-on byte is *missing*!!! This may be correlated to having a note velocity of 0, but not sure about that. There are no Note-Off messages in that file, by the way.

3. Note value byte. This value ranges from 0 to 127.
4. Note velocity byte. This value ranges from 0 to 127. It seems that a note off can be made by setting this value to zero. Again, not sure about that.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>vol</i>	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 553 of file midicvt_base.c.

9.10.2.22 `static int my_parameter (int chan, int control, int value) [static]`

Callback function implementing Mf_parameter().

Command: 0xBn .

This function prints the Control Change message. This message has different parameters than the Note On/Off messages.

The MIDI bytes for a control-change message are 4 in number:

1. Delta-time byte. A typical value is 0x60 (96).
2. Control-change byte. 0xBn . This byte is a status byte having two parts, the command nybble and the channel nybble.
3. Control-change value byte. This value ranges from 0 to 127. It determines which controller (e.g. pitch or sustain) is changed.
4. Controller value byte. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>control</i>	Provides the MIDI controller number to write.
<i>value</i>	Provides the MIDI controller parameter value to write.

Returns

Returns true, always.

Definition at line 653 of file midicvt_base.c.

9.10.2.23 `static int my_pitchbend (int chan, int lsb, int msb) [static]`

Callback function implementing Mf_pitchbend().

Command: 0xEn .

- Ec 11 mm. Pitch Wheel Change. Provides the least significant and most significant 14 bits.

1. Delta-time byte. A typical value is 0x60 (96).

2. Pitch-bend byte. `0xEb` .
3. Pitch, least-significant value byte. This value ranges from 0 to 127. It provides the lowest 7 bits of the pitch-bend parameter.
4. Pitch, most-significant value byte. It provides the highest 7 bits of the pitch-bend parameter.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>lsb</i>	Provides the least-significant bits of the MIDI pitch-wheel value to write.
<i>msb</i>	Provides the most-significant bits of the MIDI pitch-wheel value to write.

Returns

Returns true, always.

Definition at line 691 of file `midicvt_base.c`.

9.10.2.24 `static int my_pressure (int chan, int pitch, int pressure)` `[static]`

Callback function implementing `Mf_pressure()`.

Command: `0xAn` .

Very similar to Note-On messages [see [my_non\(\)](#)].

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>pressure</i>	Provides the MIDI polyphonic key pressure value to write.

Returns

Returns true, always.

Definition at line 612 of file `midicvt_base.c`.

9.10.2.25 `static int my_program (int chan, int program)` `[static]`

Callback function implementing `Mf_program()`.

Command: `0xCn` .

A program-change (patch change) command has the following 3 bytes:

1. Delta-time byte. A typical value is `0x60` (96).
2. Program-change byte. `0xCn` .
3. Program/patch number byte. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
-------------	---------------------------------------

<i>program</i>	Provides the patch/program number to write.
----------------	---

Returns

Returns true, always.

Definition at line 720 of file midicvt_base.c.

9.10.2.26 static int my_smpte (int *hr*, int *mn*, int *se*, int *fr*, int *ff*) [static]

Callback function implementing Mf_smpte().

Prints "SMPTE" and each of the parameters shown below, to standard output.

Parameters

<i>hr</i>	Provides the hour of the SMPTE time at which the track chunk is supposed to start. It should be present at the beginning of the track, before any non-zero delta times, and before any transmittable MIDI events.
<i>mn</i>	Provides the minutes of the SMPTE time.
<i>se</i>	Provides the seconds of the SMPTE time.
<i>fr</i>	Provides the frames of the SMPTE time.
<i>ff</i>	Provides the fractional frames of the SMPTE time, in 100ths of a frame, even in SMPTE-based tracks which specify as different frame subdivision for delta times..

Returns

Returns true, always.

Definition at line 1152 of file midicvt_base.c.

9.10.2.27 static int my_sysex (int *leng*, char * *mess*) [static]

Callback function implementing Mf_sysex().

This function prints "SysEx" and then a stream of bytes, in hex format, to standard output.

Command: 0xF0 .

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).
2. System-exclusive byte. 0xF0 .
3. Manufacturer's code. A 7-bit value, highest bit is 0.
4. Data bytes. A series of 7-bit values.
5. EOS (End-of-System-exclusive) byte. 0xF7 .

Parameters

<i>leng</i>	Provides the number of bytes in the message.
<i>mess</i>	Provides a pointer to the message bytes.

Returns

Returns true, always.

Definition at line 783 of file midicvt_base.c.

9.10.2.28 static int my_tempo (long tempo) [static]

Callback function implementing Mf_tempo().

Prints the string "Tempo" followed by the tempo value, to standard output.

Command: FF 51 03 tt tt tt.

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. End-of-track marker. 0x51 .
4. Length byte. 0x03 .
5. Data bytes. There are three data bytes that specify the tempo in "microseconds per MIDI quarter-note" or "24ths of a microsecond per MIDI clock". Chew on this conversion: 07 a1 20 == 500000 usec/quarter note == 1/2 second per quarter note == 120 bpm (beats per minute), where each quarter note is a beat.

Parameters

<i>tempo</i>	Provides the tempo value as a single integer.
--------------	---

Returns

Returns true, always.

Definition at line 1061 of file midicvt_base.c.

9.10.2.29 static int my_timesig (int nn, int dd, int cc, int bb) [static]

Callback function implementing Mf_timesig().

Prints the string "TimeSig" followed by the four value needed for a key signature, to standard output.

Command: FF 58 04 nn dd cc bb.

As an example, this sequence of bytes:

FF 58 04 06 03 24 08

This is the complete event for 6/8 time, where the metronome clicks every three eighth-notes, but there are 24 clocks per quarter-note, 72 to the bar. That is, 6/8 time (8 is 2 to the 3rd power, so this is 06 03), 36 MIDI clocks per dotted-quarter (24 hex!), and eight notated 32nd-notes per quarter-note.

Parameters

<i>nn</i>	Provides the numerator of the time signature as it would be notated.
<i>dd</i>	Provides the denominator of the time signature as it would be notated. The denominator is a negative power of two, where 2 represents a quarter note, 3 represents an eighth note....
<i>cc</i>	Provides the number of MIDI clock in a metronome click.
<i>bb</i>	The number of notated 32nd notes in a MIDI quarter-note (a MIDI quarter note is 24 MIDI clocks).

Returns

Returns true, always.

Definition at line 1106 of file midicvt_base.c.

9.10.2.30 static int my_trend (long header_offset, unsigned long track_size) [static]

Callback function implementing Mf_endtrack().

This function write "TrkEnd" and a newline to standard output, and decrements the global "tracks to do" counter.

In MIDI, the end-of-track marker is three bytes, ff 2f 00. One file has a 0a as well.

Parameters

<i>header_offset</i>	Offset of the header of the track that is now ending. This
<i>track_size</i>	Provides the actual size of the track. This parameter is not used in this callback.

Returns

Returns true, always.

Definition at line 491 of file midicvt_base.c.

9.10.2.31 static int my_trstart (void) [static]

Callback function implementing Mf_starttrack().

This function writes "MTrk" and a newline to standard output and increments the global track number counter.

This marker is a 4-byte unterminated ASCII marker.

Returns

Returns true, always.

Definition at line 462 of file midicvt_base.c.

9.10.2.32 static int my_writetrack (void) [static]

Callback function implementing Mf_wtrack() and Mf_wtempotrack().

This function is called for writing MIDI to text and for text to MIDI:

```
Mf_wtrack      = my_writetrack;    // text-to-MIDI and vice versa
Mf_wtempotrack = my_writetrack;    // text-to-MIDI only
```

However, this function also <i> reads </i> data!!!
 This function scans the return value of the flex function yylex()
 until it finds it is equal to MTRK (which happens if the string
 "MTrk" is found). A bunch of other markers ("TrkEnd", "SysEx",
 "Meta", "Arb", and more (see t2mf.fl), are found and processed.

For writing MIDI, the following functions from the midifilex.c
 module are called:

- mf_w_midi_event(). Writes the "delta time", the MIDI event type plus channel byte, plus the event data.,
- mf_w_sysex_event(). Writes the delta time and the SYSEX or "arbitrary" data for the event.
- mf_w_meta_event(). Writes the delta time and the meta-event. There are a number of meta-events that are written.

Returns

Returns -1 upon an error, otherwise returns true (1).

Definition at line 1245 of file midicvt_base.c.

9.10.2.33 `static void prhex (unsigned char * p, int leng) [static]`

Writes the provided text to standard output in hexadecimal format.

It also handles the `–fold` option.

Parameters

<i>p</i>	Provides the characters to be written.
<i>leng</i>	Provides the number of characters to be written.

Definition at line 323 of file `midicvt_base.c`.

9.10.2.34 `static char* prnote (int pitch) [static]`

Writes a note value to standard output.

If the `–verbose` option was given, then the note is written as a letter value with the octave number following it. Otherwise, the note is written as a integer MIDI note value.

Note

Not sure why this function isn't called "prnote()", so I renamed it from "mknote()".

Not threadsafe Due to the static buffer `buf[8]` used inside this function.

Parameters

<i>pitch</i>	Provides the MIDI note value to be written.
--------------	---

Returns

Returns a pointer to the beginning of the static buffer `buf[]`.

Definition at line 371 of file `midicvt_base.c`.

9.10.2.35 `static int prs_error (char * s) [static]`

Outputs error messages to standard error.

This function also skips thto the end of the line of the input. If the end-of-file is encountered, then `exit(1)` is called. If there is an error status, then a `longjmp()` to `g_status_erjump` is called.

Parameters

<i>s</i>	Provides the error string. Note that <code>yytext</code> and <code>yytext</code> are global inputs to this function, as well.
----------	---

Returns

Returns true, always.

Definition at line 1568 of file `midicvt_base.c`.

9.10.2.36 `static void prtext (unsigned char * p, int leng) [static]`

Writes the provided text to standard output.

The text is enclosed in quotes, and the backslash escape character is emitted where needed.

This function will also employ the `–fold` option.

Parameters

<i>p</i>	Provides the characters to be written. It is unsigned, but the characters should all, I think, be straight 7-bit ASCII printing characters.
<i>leng</i>	Provides the number of characters to be written.

Definition at line 249 of file midicvt_base.c.

9.10.2.37 static void prtime (void) [static]

Writes the time of the current MIDI event to standard output in text format.

This function handles the "absolute times" (`-time`) and "verbose notes" (`-verbose`) options as well.

Definition at line 199 of file midicvt_base.c.

9.10.2.38 static cbool_t redirect_stdout (const char * filename, const char * mode) [static]

Redirects stdout to a file.

Parameters

<i>filename</i>	Provides the filename to be use for the out that would have gone to stdout.
<i>mode</i>	Provides the mode to use on the file. Probably makes sense only in Windows, which this application doesn't support yet. Should be either "w" or "wb", but this is not verified. You're a big boy.

Returns

Returns true (1) if the setup succeeded. Do not rely on using stdout in the rest of the program if this function returns false (0).

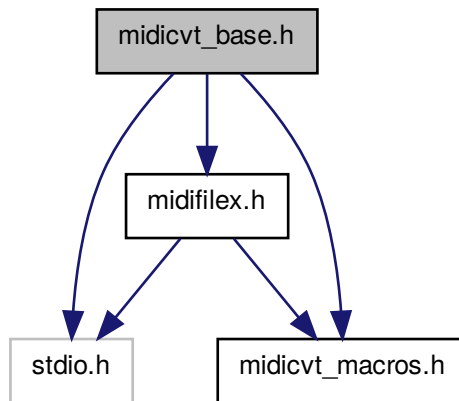
Definition at line 2068 of file midicvt_base.c.

9.11 midicvt_base.h File Reference

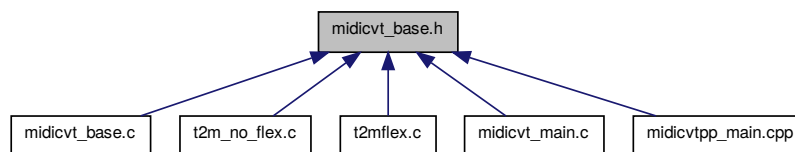
This module provides functions for a basic MIDI/text conversion application.

```
#include <stdio.h>
#include <midicvt_macros.h>
#include <midifilex.h>
```

Include dependency graph for `midicvt_base.h`:



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



Functions

- void `error` (const char *s)
Writes an obvious error string to standard error.
- void `midicvt_initfuncs_t2mf` (void)
Makes the function assignments needed by the midifile library when converting a text file to MIDI.
- void `midicvt_initfuncs_mf2t` (void)
Makes the function assignments needed by the midifile library when converting a MIDI file to text.
- `cbool_t` `midicvt_setup_compile` (void)
Sets up the incoming text file for compiling into MIDI.
- void `midicvt_close_compile` (void)
Closes the input file (yyin) and the output file (g_io_file) for the `--compile` option.
- void `midicvt_compile` (void)
This function makes sure the MFile token is found.
- `cbool_t` `midicvt_setup_mfread` (void)
Sets up the file handles for both reading and writing.
- void `midicvt_close_mfread` (void)
Checks g_io_file for an error status, closes this file handle if necessary, and restores stdout to norml if necessary.

9.11.1 Detailed Description

This module provides functions for a basic MIDI/text conversion application.

Library:

midicvt application portion of libmidifilex

Author

Chris Ahlstrom and many others; see documentation

Date

2014-04-09

Last Edits:

2014-05-20

Version

\$Revision\$

License:

GNU GPL

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

9.11.2 Function Documentation

9.11.2.1 void error (const char * s)

Writes an obvious error string to standard error.

Public Note that this is an public function used in other modules, and declared in the [midicvt_base.h](#) file.

Parameters

s	Provides the null-terminated error message to be written.
---	---

Definition at line 161 of file midicvt_base.c.

9.11.2.2 void midicvt_compile (void)

This function makes sure the MFile token is found.

It then gathers up some status information and passes it to [mfwrite\(\)](#).

Note

This function used to be called translate(), which was a bit ambiguous.

Definition at line 1611 of file midicvt_base.c.

9.11.2.3 `cbool_t midicvt_setup_compile (void)`

Sets up the incoming text file for compiling into MIDI.

Returns

Returns true if all is well.

Definition at line 2143 of file `midicvt_base.c`.

9.11.2.4 `cbool_t midicvt_setup_mfread (void)`

Sets up the file handles for both reading and writing.

Returns

Returns true if all steps succeeded.

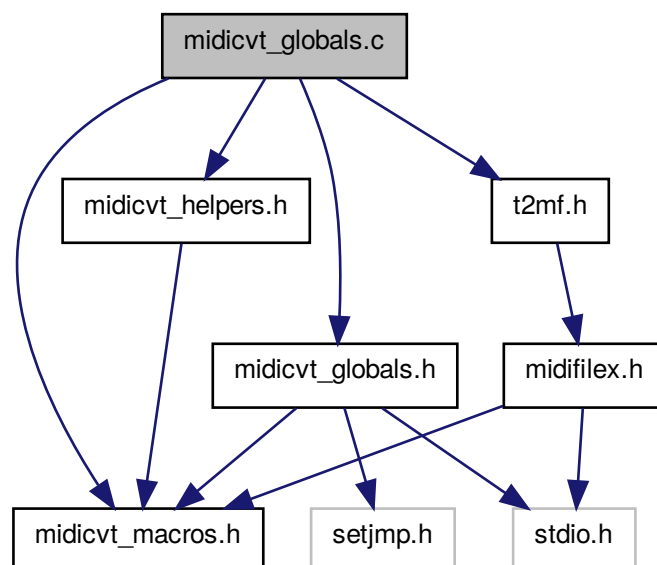
Definition at line 2207 of file `midicvt_base.c`.

9.12 `midicvt_globals.c` File Reference

This module provides a place to collect all the global variables used, and functions to set them.

```
#include <midicvt_macros.h>
#include <midicvt_globals.h>
#include <midicvt_helpers.h>
#include <t2mf.h>
```

Include dependency graph for `midicvt_globals.c`:



Functions

- void [midicvt_set_defaults](#) (void)
Sets some defaults for status variables.
- void [midicvt_set_option_fold](#) (int f)
'Setter' function for member g_option_fold
- int [midicvt_option_fold](#) (void)
'Getter' function for member g_option_fold
- void [midicvt_set_option_verbose](#) (cbool_t f)
'Setter' function for member g_option_verbose
- cbool_t [midicvt_option_verbose](#) (void)
'Getter' function for member g_option_verbose
- void [midicvt_set_option_verbose_notes](#) (cbool_t f)
'Setter' function for member g_option_verbose_notes
- cbool_t [midicvt_option_verbose_notes](#) (void)
'Getter' function for member g_option_verbose_notes
- void [midicvt_set_option_absolute_times](#) (cbool_t f)
'Setter' function for member g_option_absolute_times
- cbool_t [midicvt_option_absolute_times](#) (void)
'Getter' function for member g_option_absolute_times
- void [midicvt_set_option_debug](#) (cbool_t f)
'Setter' function for member g_option_debug
- cbool_t [midicvt_option_debug](#) (void)
'Getter' function for member g_option_debug
- void [midicvt_set_option_compile](#) (cbool_t f)
'Setter' function for member g_option_docompile
- cbool_t [midicvt_option_compile](#) (void)
'Getter' function for member g_option_docompile
- void [midicvt_set_option_m2m](#) (cbool_t f)
'Setter' function for member g_option_midi2midi
- cbool_t [midicvt_option_m2m](#) (void)
'Getter' function for member g_option_midi2midi

Variables

- long [yyval](#) = 0UL
Use externs from an include file! The flex-generated code is a bit messy.

9.12.1 Detailed Description

This module provides a place to collect all the global variables used, and functions to set them.

Library:

midicvt application

Author

Chris Ahlstrom

Date

2014-04-09

Last Edits:

2014-05-20

Version

\$Revision\$

License:

GNU GPL

Definition in file [midicvt_globals.c](#).

9.12.2 Function Documentation

9.12.2.1 `cbool_t midicvt_option_compile (void)`

*'Getter' function for member `g_option_docompile`*We need to expose this value for [main\(\)](#) to use.Definition at line 236 of file `midicvt_globals.c`.

9.12.2.2 `cbool_t midicvt_option_m2m (void)`

*'Getter' function for member `g_option_midi2midi`*We need to expose this value for [main\(\)](#) to use.Definition at line 258 of file `midicvt_globals.c`.

9.12.3 Variable Documentation

9.12.3.1 `long yyval = 0UL`

Use externs from an include file! The flex-generated code is a bit messy.

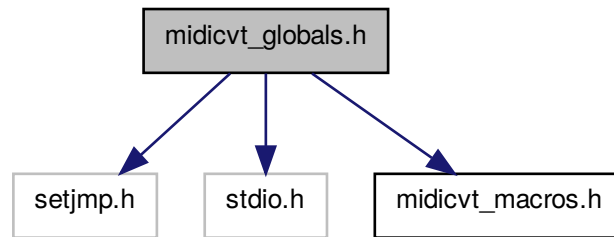
The generated [t2mflex.c](#) file defines these variables, we declare them in `t2fm.h`, and have to define `yyval` here. Bleh.Definition at line 80 of file `midicvt_globals.c`.

9.13 `midicvt_globals.h` File Reference

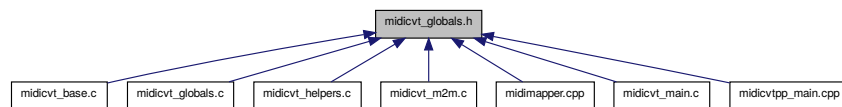
This module provides global variables and accessor functions for the `midicvt` portion of `libmidifilex`.

```
#include <setjmp.h>
#include <stdio.h>
#include <midicvt_macros.h>
```

Include dependency graph for midicvt_globals.h:



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



Functions

- void [midicvt_set_defaults](#) (void)
Sets some defaults for status variables.
- void [midicvt_set_option_fold](#) (int f)
'Setter' function for member g_option_fold
- int [midicvt_option_fold](#) (void)
'Getter' function for member g_option_fold
- void [midicvt_set_option_verbose](#) (cbool_t f)
'Setter' function for member g_option_verbose
- cbool_t [midicvt_option_verbose](#) (void)
'Getter' function for member g_option_verbose
- void [midicvt_set_option_verbose_notes](#) (cbool_t f)
'Setter' function for member g_option_verbose_notes
- cbool_t [midicvt_option_verbose_notes](#) (void)
'Getter' function for member g_option_verbose_notes
- void [midicvt_set_option_absolute_times](#) (cbool_t f)
'Setter' function for member g_option_absolute_times
- cbool_t [midicvt_option_absolute_times](#) (void)
'Getter' function for member g_option_absolute_times
- void [midicvt_set_option_debug](#) (cbool_t f)
'Setter' function for member g_option_debug
- cbool_t [midicvt_option_debug](#) (void)
'Getter' function for member g_option_debug
- void [midicvt_set_option_compile](#) (cbool_t f)

- 'Setter' function for member g_option_docompile*
- `cbool_t midicvt_option_compile` (void)
- 'Getter' function for member g_option_docompile*
- void `midicvt_set_option_m2m` (cbool_t f)
- 'Setter' function for member g_option_midi2midi*
- `cbool_t midicvt_option_m2m` (void)
- 'Getter' function for member g_option_midi2midi*

Variables

- long `yyval`

Use externs from an include file! The flex-generated code is a bit messy.

9.13.1 Detailed Description

This module provides global variables and accessor functions for the midicvt portion of libmidifilex.

Library:

midicvt application portion of libmidifilex

Author

Chris Ahlstrom and others; see documentation

Date

2014-04-09

Last Edits:

2014-05-20

Version

\$Revision\$

License:

GNU GPL

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

9.13.2 Function Documentation

9.13.2.1 `cbool_t midicvt_option_compile (void)`

'Getter' function for member g_option_docompile

We need to expose this value for [main\(\)](#) to use.

Definition at line 236 of file `midicvt_globals.c`.

9.13.2.2 `cbool_t midicvt_option_m2m (void)`

'Getter' function for member `g_option_midi2midi`

We need to expose this value for `main()` to use.

Definition at line 258 of file `midicvt_globals.c`.

9.13.3 Variable Documentation

9.13.3.1 `long yyval`

Use externs from an include file! The flex-generated code is a bit messy.

The generated `t2mflex.c` file defines these variables, we declare them in `t2fm.h`, and have to define `yyval` here. Bleh.

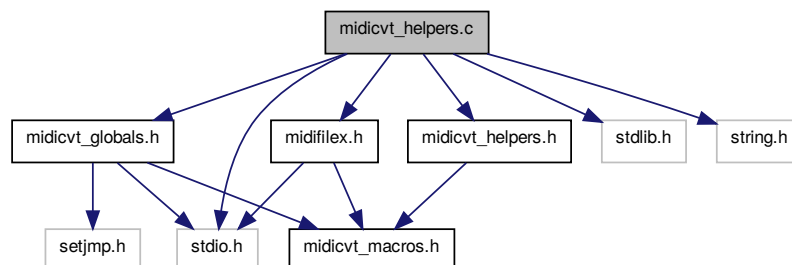
Definition at line 80 of file `midicvt_globals.c`.

9.14 `midicvt_helpers.c` File Reference

This module provides the help text for `midicvt`.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <midifilex.h>
#include <midicvt_globals.h>
#include <midicvt_helpers.h>
```

Include dependency graph for `midicvt_helpers.c`:



Macros

- `#define MIDICVT_PATH_MAX 1024`
Maximum path-name length.

Functions

- void `midicvt_version` (const char *version)
Provides the version text for the midicvt-related programs.
- void `midicvt_help` (const char *version)
Provides the help text for the midicvt program.

- `cbool_t midicvt_have_input_file ()`
'Getter' function for member gs_have_input_file Provides a safe accessor for this global variable.
- `const char * midicvt_input_file ()`
'Getter' function for member gs_input_file Provides a safe accessor for this global variable.
- `cbool_t midicvt_set_input_file (const char *inputfile)`
Sets the input file-name.
- `cbool_t midicvt_have_output_file ()`
'Getter' function for member gs_have_output_file Safe accessor.
- `const char * midicvt_output_file ()`
'Getter' function for member gs_output_file Safe accessor.
- `cbool_t midicvt_set_output_file (const char *outputfile)`
Sets the output file-name.
- `long midi_file_offset (void)`
Exposes the file-offset counter, useful in debugging.
- `void midi_file_offset_clear (void)`
Exposes the file-offset counter, useful in debugging.
- `void midi_file_offset_increment (void)`
Increments the file-offset counter.
- `static int report (const char *msg)`
Provides an optional and simple reporting function to be used by libmidifile.
- `cbool_t check_option (const char *source, const char *shortopt, const char *longopt)`
Checks the option argument to see if it matches a short or a long option.
- `cbool_t midicvt_parse (int argc, char *argv[], const char *version)`
Parses the midicvt (C) command line.

Variables

- `static const char *const gs_help_version = "midicvt v 0.2"`
Default version string.
- `static const char *const gs_help_usage_1`
Help string.
- `static cbool_t gs_have_input_file = false`
Indicates if an input file has yet been specified on the command line.
- `static char gs_input_file [MIDICVT_PATH_MAX]`
Holds the value of the input file name.
- `static cbool_t gs_have_output_file = false`
Indicates if an output file has yet been specified on the command line.
- `static char gs_output_file [MIDICVT_PATH_MAX]`
Holds the value of the output file name.
- `static long gs_file_offset = 0`
Provides the current offset into the global input file.

9.14.1 Detailed Description

This module provides the help text for midicvt.

Library:

midicvt application

Author

Chris Ahlstrom

Date

2014-04-19

Last Edits:

2014-06-01

Version

\$Revision\$

License:

GNU GPL

We've offloaded the help text of midicvt to this module so that it can be re-used in an upcoming C++ version of midicvt.

Also for code re-use, the flex and file-access code from [main\(\)](#) have been moved to this module.

Definition in file [midicvt_helpers.c](#).

9.14.2 Macro Definition Documentation**9.14.2.1 #define MIDICVT_PATH_MAX 1024**

Maximum path-name length.

(We don't really support Windows yet.)

Definition at line 117 of file midicvt_helpers.c.

9.14.3 Function Documentation**9.14.3.1 cbool_t check_option (const char * source, const char * shortopt, const char * longopt)**

Checks the option argument to see if it matches a short or a long option.

None of these values are checked for being null, since they are hardwired in the code, and the programmer will fix them to make the program work. :-D

Parameters

<i>source</i>	Provides the source argument, taken from the command line. If empty, the check will fail.
<i>shortopt</i>	Provides the short version of the option, such as "-c". If empty, this value is ignored.
<i>longopt</i>	Provides the long version of the option, such as "--compile". If empty, this value is ignored.

Definition at line 377 of file midicvt_helpers.c.

9.14.3.2 long midi_file_offset (void)

Exposes the file-offset counter, useful in debugging.

Since this value is used in reporting on the character just gotten, and the offset is already incremented at that point, we deduct one from the offset.

Definition at line 303 of file midicvt_helpers.c.

9.14.3.3 void midicvt_help (const char * *version*)

Provides the help text for the midicvt program.

Parameters

<i>version</i>	Provides the optional version information.
----------------	--

Definition at line 181 of file midicvt_helpers.c.

9.14.3.4 `cbool_t midicvt_parse (int argc, char * argv[], const char * version)`

Parses the midicvt (C) command line.

Parameters

<i>argc</i>	Provides the command-line argument count, including the name of the program.
<i>argv</i>	Provides the command-line argument list, including the name of the program.
<i>version</i>	Provides the actual version string for the <code>--version</code> and <code>--help</code> options to display.

Returns

Returns true if there was no `--help` or `--version` option, and the other options were legal options for the midicvt program.

Definition at line 415 of file midicvt_helpers.c.

9.14.3.5 `cbool_t midicvt_set_input_file (const char * inputfile)`

Sets the input file-name.

Parameters

<i>inputfile</i>	Provides the full path specification for the input file.
------------------	--

Returns

Returns true if the file-name was valid and able to fit in the file-name buffer.

Definition at line 225 of file midicvt_helpers.c.

9.14.3.6 `cbool_t midicvt_set_output_file (const char * outputfile)`

Sets the output file-name.

Parameters

<i>outputfile</i>	Provides the full path specification for the output file.
-------------------	---

Returns

Returns true if the file-name was valid and able to fit in the file-name buffer.

Definition at line 277 of file midicvt_helpers.c.

9.14.3.7 `void midicvt_version (const char * version)`

Provides the version text for the midicvt-related programs.

Parameters

<i>version</i>	Provides the optional version information.
----------------	--

Definition at line 165 of file `midicvt_helpers.c`.

9.14.3.8 `static int report (const char * msg) [static]`

Provides an optional and simple reporting function to be used by `libmidifile`.

Parameters

<i>msg</i>	Provides the basic message to be written to standard error.
------------	---

Returns

Returns 1 if the result of `midi_file_offset()` is greater-than-or-equal to 0, and it returns 0 otherwise.

Definition at line 341 of file `midicvt_helpers.c`.

9.14.4 Variable Documentation

9.14.4.1 `cbool_t gs_have_input_file = false [static]`

Indicates if an input file has yet been specified on the command line.

This value is usually the first non-option argument specified, or the parameter for the `-i` (–input) or `-c` (–compile) options.

Definition at line 127 of file `midicvt_helpers.c`.

9.14.4.2 `cbool_t gs_have_output_file = false [static]`

Indicates if an output file has yet been specified on the command line.

This value is usually the second non-option argument specified, or the parameter for the `-o` (–output) option.

Definition at line 142 of file `midicvt_helpers.c`.

9.14.4.3 `const char* const gs_help_usage_1 [static]`

Initial value:

```
=
"\n"
"midicvt refactors the midicomp program for translating between MIDI/SMF and\n"
"text files. Compare it to the midi2text project at code.google.com.\n"
"\n"
"Command line argument usage:\n"
```

Help string.

Because of legacy C rules, we have to define 5 different help strings to avail ourselves of enough characters.

Definition at line 61 of file `midicvt_helpers.c`.

9.14.4.4 `const char* const gs_help_version = "midicvt v 0.2" [static]`

Default version string.

Normally, the caller of `midicvt_version()` will provide a string, though.

Definition at line 54 of file `midicvt_helpers.c`.

9.14.4.5 `char gs_input_file[MIDICVT_PATH_MAX]` `[static]`

Holds the value of the input file name.

Restricted to 1024 bytes for now, but we should make it a heap variable soon.

Definition at line 134 of file `midicvt_helpers.c`.

9.14.4.6 `char gs_output_file[MIDICVT_PATH_MAX]` `[static]`

Holds the value of the output file name.

Restricted to 1024 bytes for now, but we should make it a heap variable soon.

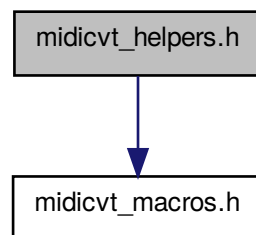
Definition at line 149 of file `midicvt_helpers.c`.

9.15 `midicvt_helpers.h` File Reference

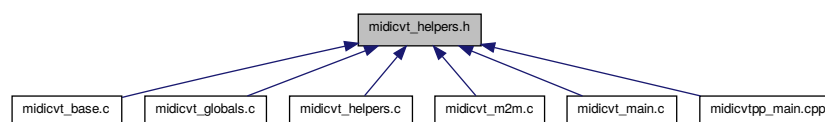
This module declares the help and version functions for `midicvt`.

```
#include <midicvt_macros.h>
```

Include dependency graph for `midicvt_helpers.h`:



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



Functions

- `cbool_t check_option` (`const char *source`, `const char *shoropt`, `const char *longopt`)
Checks the option argument to see if it matches a short or a long option.
- `void midicvt_version` (`const char *version`)
Provides the version text for the `midicvt`-related programs.

- void [midicvt_help](#) (const char *version)
Provides the help text for the midicvt program.
- [cbool_t midicvt_set_input_file](#) (const char *inputfile)
Sets the input file-name.
- [cbool_t midicvt_have_input_file](#) ()
'Getter' function for member gs_have_input_file Provides a safe accessor for this global variable.
- const char * [midicvt_input_file](#) ()
'Getter' function for member gs_input_file Provides a safe accessor for this global variable.
- [cbool_t midicvt_set_output_file](#) (const char *outputfile)
Sets the output file-name.
- [cbool_t midicvt_have_output_file](#) ()
'Getter' function for member gs_have_output_file Safe accessor.
- const char * [midicvt_output_file](#) ()
'Getter' function for member gs_output_file Safe accessor.
- [cbool_t midicvt_parse](#) (int argc, char *argv[], const char *version)
Parses the midicvt (C) command line.
- long [midi_file_offset](#) (void)
Exposes the file-offset counter, useful in debugging.
- void [midi_file_offset_clear](#) (void)
Exposes the file-offset counter, useful in debugging.
- void [midi_file_offset_increment](#) (void)
Increments the file-offset counter.

9.15.1 Detailed Description

This module declares the help and version functions for midicvt.

Library:

midicvt application

Author

Chris Ahlstrom

Date

2014-04-19

Last Edits:

2014-04-23

Version

\$Revision\$

License:

GNU GPL

We've offloaded the help text of midicvt to this module so that it can be re-used in an upcoming C++ version of midicvt.

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

9.15.2 Function Documentation

9.15.2.1 `cbool_t check_option (const char * source, const char * shortopt, const char * longopt)`

Checks the option argument to see if it matches a short or a long option.

None of these values are checked for being null, since they are hardwired in the code, and the programmer will fix them to make the program work. :-D

Parameters

<i>source</i>	Provides the source argument, taken from the command line. If empty, the check will fail.
<i>shortopt</i>	Provides the short version of the option, such as "-c". If empty, this value is ignored.
<i>longopt</i>	Provides the long version of the option, such as "--compile". If empty, this value is ignored.

Definition at line 377 of file midicvt_helpers.c.

9.15.2.2 `long midi_file_offset (void)`

Exposes the file-offset counter, useful in debugging.

Since this value is used in reporting on the character just gotten, and the offset is already incremented at that point, we deduct one from the offset.

Definition at line 303 of file midicvt_helpers.c.

9.15.2.3 `void midicvt_help (const char * version)`

Provides the help text for the midicvt program.

Parameters

<i>version</i>	Provides the optional version information.
----------------	--

Definition at line 181 of file midicvt_helpers.c.

9.15.2.4 `cbool_t midicvt_parse (int argc, char * argv[], const char * version)`

Parses the midicvt (C) command line.

Parameters

<i>argc</i>	Provides the command-line argument count, including the name of the program.
<i>argv</i>	Provides the command-line argument list, including the name of the program.
<i>version</i>	Provides the actual version string for the --version and --help options to display.

Returns

Returns true if there was no "--help" or "--version" option, and the other options were legal options for the midicvt program.

Definition at line 415 of file midicvt_helpers.c.

9.15.2.5 `cbool_t midicvt_set_input_file (const char * inputfile)`

Sets the input file-name.

Parameters

<i>inputfile</i>	Provides the full path specification for the input file.
------------------	--

Returns

Returns true if the file-name was valid and able to fit in the file-name buffer.

Definition at line 225 of file `midicvt_helpers.c`.

9.15.2.6 `cbool_t midicvt_set_output_file (const char * outputfile)`

Sets the output file-name.

Parameters

<i>outputfile</i>	Provides the full path specification for the output file.
-------------------	---

Returns

Returns true if the file-name was valid and able to fit in the file-name buffer.

Definition at line 277 of file `midicvt_helpers.c`.

9.15.2.7 `void midicvt_version (const char * version)`

Provides the version text for the midicvt-related programs.

Parameters

<i>version</i>	Provides the optional version information.
----------------	--

Definition at line 165 of file `midicvt_helpers.c`.

9.16 `midicvt_license.dox` File Reference

This file is a copy of the main license file.

9.16.1 Detailed Description

This file is a copy of the main license file.

Definition in file [midicvt_license.dox](#).

9.17 `midicvt_m2m.c` File Reference

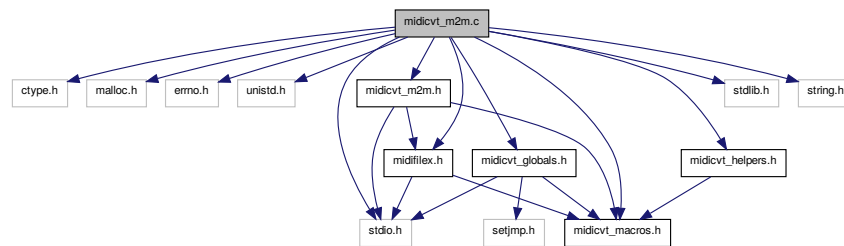
This module provides functions for basic MIDI-to-MIDI conversions.


```

#include <ctype.h>
#include <malloc.h>
#include <errno.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <midicvt_macros.h>
#include <midicvt_m2m.h>
#include <midicvt_globals.h>
#include <midicvt_helpers.h>
#include <midifilex.h>

```

Include dependency graph for midicvt_m2m.c:



Functions

- static int [fileputc](#) (unsigned char c)
Callback function implementing `Mf_putc()` for MIDI-to-MIDI conversions.
- static int [filegetc](#) (void)
Callback function implementing `Mf_getc()`.
- static int [m2m_error](#) (const char *s)
Callback function implementing `Mf_error()`.
- static int [m2m_header](#) (int format, int ntrks, int division)
Callback function implementing `Mf_header()`.
- static int [m2m_trstart](#) (void)
Callback function implementing `Mf_starttrack()`.
- static int [m2m_trend](#) (long header_offset, unsigned long track_size)
Callback function implementing `Mf_endtrack()`.
- int [m2m_non](#) (int chan, int pitch, int vol)
Callback function implementing `Mf_on()`.
- int [m2m_noff](#) (int chan, int pitch, int vol)
Callback function implementing `Mf_off()`.
- int [m2m_pressure](#) (int chan, int pitch, int pressure)
Callback function implementing `Mf_pressure()`.
- int [m2m_parameter](#) (int chan, int control, int value)
Callback function implementing `Mf_parameter()`.
- int [m2m_pitchbend](#) (int chan, int lsb, int msb)
Callback function implementing `Mf_pitchbend()`.
- int [m2m_program](#) (int chan, int program)
Callback function implementing `Mf_program()`.
- int [m2m_chanpressure](#) (int chan, int pressure)

- Callback function implementing *Mf_chanpressure()*.
 - static int `m2m_sysex` (int leng, char *mess)
 - Callback function implementing *Mf_sysex()*.
 - static int `m2m_mmisc` (int typecode, int leng, char *mess)
 - Callback function implementing *Mf_metamisc()*.
 - static int `m2m_mspecial` (int leng, char *mess)
 - Callback function implementing *Mf_sqspecific()*.
 - static int `m2m_mtext` (int type, int leng, char *mess)
 - Callback function implementing *Mf_text()*.
 - static int `m2m_mseq` (short int num)
 - Callback function implementing *Mf_seqnum()*.
 - int `m2m_meot` (void)
 - Callback function implementing *Mf_eot()*.
 - static int `m2m_keysig` (int sf, int mi)
 - Callback function implementing *Mf_keysig()*.
 - static int `m2m_tempo` (long tempo)
 - Callback function implementing *Mf_tempo()*.
 - static int `m2m_timesig` (int nn, int dd, int cc, int bb)
 - Callback function implementing *Mf_timesig()*.
 - static int `m2m_smppte` (int hr, int mn, int se, int fr, int ff)
 - Callback function implementing *Mf_smppte()*.
 - static int `m2m_arbitrary` (int leng, char *mess)
 - Callback function implementing *Mf_arbitrary()*.
 - void `midicvt_initfuncs_m2m` (void)
 - Makes the function assignments needed by the midifile library when converting a MIDI file to text.

9.17.1 Detailed Description

This module provides functions for basic MIDI-to-MIDI conversions.

Library:

midicvt application

Author

Chris Ahlstrom and many other authors

Date

2014-04-27

Last Edits:

2014-05-20

Version

\$Revision\$

License:

GNU GPL

Why do we want to convert MIDI-to-MIDI? Well, for one thing, as we have seen with some of the test files, that conversion can fix MIDI files.

For another thing, we can insert transformations directly into the midicvt program, and call the functions here to write out the transformed MIDI. This work will give users some stock functionality without having to dealing with pipes and scripts.

A program that transforms MIDI data will define a callback function that first transforms the MIDI data, and then calls one of the functions here to write the MIDI.

Definition in file [midicvt_m2m.c](#).

9.17.2 Function Documentation**9.17.2.1 static int filegetc (void) [static]**

Callback function implementing [Mf_getc\(\)](#).

This function reads from the `g_io_file` FILE pointer.

Returns

Returns the value returned by `getc()`, or a -1 upon error.

Definition at line 99 of file `midicvt_m2m.c`.

9.17.2.2 static int fileputc (unsigned char c) [static]

Callback function implementing `Mf_putc()` for MIDI-to-MIDI conversions.

This function writes to the `g_redirect_file` FILE pointer.

Parameters

<code>c</code>	Provides the character to be written.
----------------	---------------------------------------

Returns

Returns the value returned by `putc()`, or a -1 upon error.

Definition at line 77 of file `midicvt_m2m.c`.

9.17.2.3 static int m2m_arbitrary (int leng, char * mess) [static]

Callback function implementing `Mf_arbitrary()`.

This function is called in [readtrack\(\)](#) if there are additional bytes that are not part of a system exclusive continuation.

I have to confess that, right now, I don't know what that means.

Parameters

<i>leng</i>	Provides the number of bytes in the message.
<i>mess</i>	Points to the bytes in the message.

Returns

Returns true, always.

Definition at line 991 of file midicvt_m2m.c.

9.17.2.4 `int m2m_chanpressure (int chan, int pressure)`

Callback function implementing `Mf_chanpressure()`.

No longer static, so that it can be using in the C++ program midicvtp.

Command: `0xDn` .

A channel-pressure command has the following 3 bytes:

1. Delta-time byte. A typical value is `0x60` (96).
2. Channel-pressure byte. `0xDn` .
3. Channel-pressure value byte. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pressure</i>	Provides the patch/program number to write.

Returns

Returns true, always.

Definition at line 561 of file midicvt_m2m.c.

9.17.2.5 `static int m2m_error (const char * s) [static]`

Callback function implementing `Mf_error()`.

This fuction checks for garbage at the end of the MIDI file. If it does not find such garbage, then the error is simply printed.

Parameters

<i>s</i>	Provides the error string to report.
----------	--------------------------------------

Returns

Returns true, always.

Definition at line 128 of file midicvt_m2m.c.

9.17.2.6 `static int m2m_header (int format, int ntrks, int division) [static]`

Callback function implementing `Mf_header()`.

Provides a libmidifilex callback function to write the MIDI header data to standard output in MIDI format.

This function is called by `readheader()` in the `midifilex.c` module. It merely needs to pass the parameters to the `mf_w_header_chunk()` function defined in that module.

Parameters

<i>format</i>	Provides the byte describing the format (0, 1, or 2) of the MIDI file.
<i>ntrks</i>	Provides the byte describing the number of tracks in the MIDI file. This value ranges from 1 to 65536. SMF 0 files have 1 track, SMF 1 has multiple tracks, with the first track containing song information.
<i>division</i>	Provides the time-division value. The first byte is either 0 (indicates the time format is ticks per quarter-note) or 1 (the time format is negative SMPTE). The second byte is the number of "ticks" per frame. For example, 0x80 indicates 128 ticks per frame.

Returns

Returns true, always.

Definition at line 168 of file midicvt_m2m.c.

9.17.2.7 static int m2m_keysig (int *sf*, int *mi*) [static]

Callback function implementing Mf_keysig().

This function prints "KeySig", followed by the *sf* parameter, followed by "major" or "minor", to standard output.

Command: FF 59 02 *sf* *mi* .

The format of this event is:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. End-of-track marker. 0x59 .
4. Length byte. 0x02 .
5. Data bytes:
 - (a) *sf* = signature flats
 - -7: 7 flats
 - -1: 7 flat
 - 0: Key of C
 - 1: 1 sharp
 - 7: 7 sharps
 - (b) *mi* = minor flag (0 = major; 1 = minor)

Parameters

<i>sf</i>	Provides the code for the keys above and below C. (I don't really understand what that mean, right now.)
<i>mi</i>	True (1) indicates the key is minor; otherwise it is major.

Returns

Returns true, always.

Definition at line 835 of file midicvt_m2m.c.

9.17.2.8 `int m2m_meot (void)`

Callback function implementing `Mf_eot()`.

This function writes "Meta TrkEnd" to standard output. However the `m2m_trend()` function takes care of this for the MIDI-to-MIDI conversion.

So this function is unused; we make it non-static (but don't expose it in the header file) just to avoid a compiler warning.

Command: `FF 00` .

The format of this event is:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. `0xFF` .
3. End-of-track marker. `0x0F` .
4. Length byte. `0x00` .

There is no data for this command, so the length byte is 0.

Returns

Returns true, always.

Definition at line 792 of file `midicvt_m2m.c`.

9.17.2.9 `static int m2m_mmisc (int typecode, int leng, char * mess) [static]`

Callback function implementing `Mf_metamisc()`.

This function prints "Meta", "0xnn" (the type of event), the length of the event as a byte, and then a stream of bytes, in hex format, to standard output.

Command: `0xFF` .

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. `0xFF` .
3. Event-type byte. A 7-bit value, highest bit is 0. These range from 01 to 08 for the known text events. See the list of other values else where in this document.
4. Length byte.
5. Data bytes. Delimited by the length byte.

Parameters

<i>typecode</i>	Provides the type of meta event.
<i>leng</i>	Provides the number of bytes in the event.
<i>mess</i>	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 635 of file `midicvt_m2m.c`.

9.17.2.10 static int m2m_mseq (short int *num*) [static]

Callback function implementing Mf_seqnum().

This function writes "SeqNr" and the sequence number to standard output.

Command: FF 00 02 ss ss or FF 00 00.

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. Sequence-number marker. 0x00 .
4. Length byte. 0x02 .
5. Data bytes. Two byte of data, I believe, at this time.

Todo Need to figure out the portable format of the sequence number bytes.

Returns

Returns true, always.

Definition at line 747 of file midicvt_m2m.c.

9.17.2.11 static int m2m_mspecial (int *leng*, char * *mess*) [static]

Callback function implementing Mf_sqspecific().

This function prints "SeqSpec", the size of the event as a byte, and then a stream of bytes, in hex format, to standard output.

Command: 0xFF 0x7F

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. Event-type byte. 0x7F .
4. Length byte.
5. Data bytes. Delimited by the length byte.

Parameters

<i>leng</i>	Provides the number of bytes in the event.
<i>mess</i>	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 671 of file midicvt_m2m.c.

9.17.2.12 static int m2m_mtext (int *type*, int *leng*, char * *mess*) [static]

Callback function implementing Mf_text().

This function prints "SeqSpec", the size of the event as a byte, and then a stream of bytes, in hex format, to standard output.

Command: 0xFF .

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is 0x60 (96).
2. Meta-event byte. 0xFF .
3. Event-type byte. 0xnn .
4. Length byte.
5. Data bytes. Delimited by the length byte.

```
(void) mf_w_meta_event(Mf_currttime, text_event, mess, (long) leng);
```

Parameters

<i>type</i>	Here, nn ranges from 01 to 0F. 01 to 08 are the known types of text event, while 09 to 0F are unrecognized text events. Other values are treat as in m2m_mmisc() .
<i>leng</i>	Provides the number of bytes in the event.
<i>mess</i>	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 714 of file midicvt_m2m.c.

9.17.2.13 int m2m_noff (int *chan*, int *pitch*, int *vol*)

Callback function implementing Mf_off().

No longer static, so that it can be using in the C++ program midicvtpp.

Command: 0x8n .

Very similar to Note-On messages [see [m2m_non\(\)](#)].

In our sample file, ex1.mid, there are no Note-Off messages, only Note-On messages with a velocity of 0.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>vol</i>	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 371 of file midicvt_m2m.c.

9.17.2.14 int m2m_non (int *chan*, int *pitch*, int *vol*)

Callback function implementing Mf_on().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0x9n .

The MIDI bytes for a note on message are 4 in number:

1. Delta-time byte. A typical value is 0x60 (96).
2. Note on byte. 0x9n . This byte is a status byte having two parts:
 - The first (most significant) nybble of this byte is 0x9.
 - The least significant nybble holds the channel number, ranging from 0x0 to 0xF. However, in our sample file, ex1.mid, there see to be notes in which the note-on byte is *missing!!!* This may be correlated to having a note velocity of 0, but not sure about that. There are no Note-Off messages in that file, by the way.
3. Note value byte. This value ranges from 0 to 127.
4. Note velocity byte. This value ranges from 0 to 127. It seems that a note off can be made by setting this value to zero. Again, not sure about that.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>vol</i>	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 335 of file midicvt_m2m.c.

9.17.2.15 int m2m_parameter (int *chan*, int *control*, int *value*)

Callback function implementing Mf_parameter().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0xBn .

This function prints the Control Change message. This message has different parameters than the Note On/Off messages.

The MIDI bytes for a control-change message are 4 in number:

1. Delta-time byte. A typical value is 0x60 (96).
2. Control-change byte. 0xBn . This byte is a status byte having two parts, the command nybble and the channel nybble.
3. Control-change value byte. This value ranges from 0 to 127. It determines which controller (e.g. pitch or sustain) is changed.
4. Controller value byte. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>control</i>	Provides the MIDI controller number to write.
<i>value</i>	Provides the MIDI controller parameter value to write.

Returns

Returns true, always.

Definition at line 450 of file midicvt_m2m.c.

9.17.2.16 int m2m_pitchbend (int *chan*, int *lsb*, int *msb*)

Callback function implementing Mf_pitchbend().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0xE_n .

- E_C 11 mm. Pitch Wheel Change. Provides the least significant and most significant 14 bits.

1. Delta-time byte. A typical value is 0x60 (96).
2. Pitch-bend byte. 0xE_n .
3. Pitch, least-significat value byte. This value ranges from 0 to 127. It provides the lowest 7 bits of the pitch-bend parameter.
4. Pitch, most-significat value byte. It provides the highest 7 bits of the pitch-bend parameter.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>lsb</i>	Provides the least-significant bits of the MIDI pitch-wheel value to write.
<i>msb</i>	Provides the most-significant bits of the MIDI pitch-wheel value to write.

Returns

Returns true, always.

Definition at line 493 of file midicvt_m2m.c.

9.17.2.17 int m2m_pressure (int *chan*, int *pitch*, int *pressure*)

Callback function implementing Mf_pressure().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0xA_n .

Very similar to Note-On messages [see [m2m_non\(\)](#)].

Parameters

<i>chan</i>	Provides the channel number to write.
-------------	---------------------------------------

<i>pitch</i>	Provides the MIDI note number to write.
<i>pressure</i>	Provides the MIDI polyphonic key pressure value to write.

Returns

Returns true, always.

Definition at line 404 of file midicvt_m2m.c.

9.17.2.18 int m2m_program (int *chan*, int *program*)

Callback function implementing Mf_program().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0xCn pp.

A program-change (patch change) command has the following 3 bytes:

1. Delta-time byte. A typical value is 0x60 (96).
2. Program-change byte. 0xCn .
3. Program/patch number byte nn. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>program</i>	Provides the patch/program number to write.

Returns

Returns true, always.

Definition at line 528 of file midicvt_m2m.c.

9.17.2.19 static int m2m_smpte (int *hr*, int *mn*, int *se*, int *fr*, int *ff*) [static]

Callback function implementing Mf_smpte().

Prints "SMPTE" and each of the parameters shown below, to standard output.

Parameters

<i>hr</i>	Provides the hour of the SMPTE time at which the track chunk is supposed to start. It should be present at the beginning of the track, before any non-zero delta times, and before any transmittable MIDI events.
<i>mn</i>	Provides the minutes of the SMPTE time.
<i>se</i>	Provides the seconds of the SMPTE time.
<i>fr</i>	Provides the frames of the SMPTE time.
<i>ff</i>	Provides the fractional frames of the SMPTE time, in 100ths of a frame, even in SMPTE-based tracks which specify as different frame subdivision for delta times..

Returns

Returns true, always.

Definition at line 960 of file midicvt_m2m.c.

9.17.2.20 `static int m2m_sysex (int leng, char * mess) [static]`

Callback function implementing `Mf_sysex()`.

This function prints "SysEx" and then a stream of bytes, in hex format, to standard output.

Command: `0xF0` .

The format of this event is roughly as follows:

1. Delta-time byte. A typical value is `0x60` (96).
2. System-exclusive byte. `0xF0` .
3. Manufacturer's code. A 7-bit value, highest bit is 0.
4. Data bytes. A series of 7-bit values.
5. EOS (End-of-System-exclusive) byte. `0xF7` .

Parameters

<i>leng</i>	Provides the number of bytes in the message.
<i>mess</i>	Provides a pointer to the message bytes.

Returns

Returns true, always.

Definition at line 596 of file `midicvt_m2m.c`.

9.17.2.21 `static int m2m_tempo (long tempo) [static]`

Callback function implementing `Mf_tempo()`.

Prints the string "Tempo" followed by the tempo value, to standard output.

Command: `FF 51 03 tt tt tt`.

1. Delta-time byte. A typical value is `0x60` (96).
2. Meta-event byte. `0xFF` .
3. End-of-track marker. `0x51` .
4. Length byte. `0x03` .
5. Data bytes. There are three data bytes that specify the tempo in "microseconds per MIDI quarter-note" or "24ths of a microsecond per MIDI clock". Chew on this conversion: `07 a1 20 == 500000 usec/quarter note == 1/2 second per quarter note == 120 bpm` (beats per minute), where each quarter note is a beat.

Note that we don't need to use `mf_w_meta_event()` here, as a more specific function, `mf_w_tempo()` can be used.

Parameters

<i>tempo</i>	Provides the tempo value as a single integer.
--------------	---

Returns

Returns true, always.

Definition at line 874 of file `midicvt_m2m.c`.

9.17.2.22 static int m2m_timesig (int *nn*, int *dd*, int *cc*, int *bb*) [static]

Callback function implementing Mf_timesig().

Prints the string "TimeSig" followed by the four value needed for a key signature, to standard output.

Command: FF 58 04 nn dd cc bb.

As an example, this sequence of bytes:

FF 58 04 06 03 24 08

This is the complete event for 6/8 time, where the metronome clicks every three eighth-notes, but there are 24 clocks per quarter-note, 72 to the bar. That is, 6/8 time (8 is 2 to the 3rd power, so this is 06 03), 36 MIDI clocks per dotted-quarter (24 hex!), and eight notated 32nd-notes per quarter-note.

Parameters

<i>nn</i>	Provides the numerator of the time signature as it would be notated.
<i>dd</i>	Provides the denominator of the time signature as it would be notated. The denominator is a negative power of two, where 2 represents a quarter note, 3 represents an eighth note....
<i>cc</i>	Provides the number of MIDI clock in a metronome click.
<i>bb</i>	The number of notated 32nd notes in a MIDI quarter-note (a MIDI quarter note is 24 MIDI clocks).

Returns

Returns true, always.

Definition at line 918 of file midicvt_m2m.c.

9.17.2.23 static int m2m_trend (long *header_offset*, unsigned long *track_size*) [static]

Callback function implementing Mf_endtrack().

This function write "TrkEnd" and a newline to standard output, and decrements the global "tracks to do" counter. Note that there is also a [m2m_meot\(\)](#) function that we do not use. That function outputs the end-of-track found in the input MIDI file, but we need to manufacture our own end-of-track..

In MIDI, the end-of-track marker is three bytes, ff 2f 00.

Oddity:

Some MIDI files end with a 0a byte as well.

Parameters

<i>header_offset</i>	Provides the offset in the output file where the "MTrk" and track length were tentatively written.
<i>track_size</i>	Provides the track-size that libmidifile actually generated. However, note that this size should also include the end-of-track meta-event written in this function, and doesn't, so we have to add the length of that event. We don't really need this parameter, since we have access (currently) to the global counter s_Mf_numbyteswritten.

Returns

Returns true, unless the file-seeks into the stdout-redirect file fail.

Definition at line 256 of file midicvt_m2m.c.

9.17.2.24 static int m2m_trstart (void) [static]

Callback function implementing Mf_starttrack().

This function writes "MTrk" and a newline to standard output and increments the global track number counter. This marker is a 4-byte unterminated ASCII marker.

Returns

Returns true, always.

Definition at line 203 of file `midicvt_m2m.c`.

9.17.2.25 void `midicvt_initfuncs_m2m` (void)

Makes the function assignments needed by the midifile library when converting a MIDI file to text.

We already handle the end of track. See `m2m_trend` for an explanation.

`Mf_eot = m2m_meot;`

Definition at line 1011 of file `midicvt_m2m.c`.

9.18 `midicvt_m2m.h` File Reference

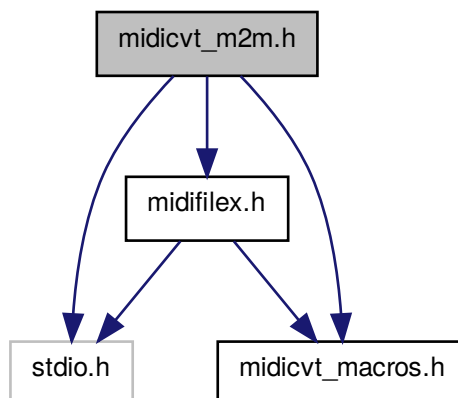
This module provides functions for a basic MIDI-to-MIDI conversion application.

```
#include <stdio.h>
```

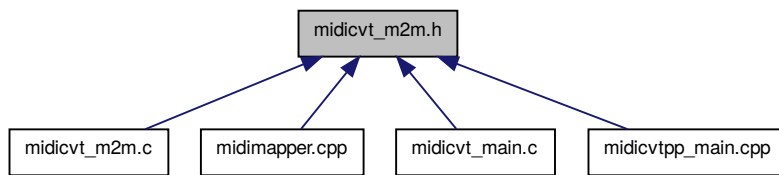
```
#include <midicvt_macros.h>
```

```
#include <midifilex.h>
```

Include dependency graph for `midicvt_m2m.h`:



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



Functions

- void [midicvt_initfuncs_m2m](#) (void)
Makes the function assignments needed by the midifile library when converting a MIDI file to text.
- int [m2m_non](#) (int chan, int pitch, int vol)
Callback function implementing `Mf_on()`.
- int [m2m_noff](#) (int chan, int pitch, int vol)
Callback function implementing `Mf_off()`.
- int [m2m_pressure](#) (int chan, int pitch, int pressure)
Callback function implementing `Mf_pressure()`.
- int [m2m_program](#) (int chan, int program)
Callback function implementing `Mf_program()`.
- int [m2m_parameter](#) (int chan, int control, int value)
Callback function implementing `Mf_parameter()`.
- int [m2m_pitchbend](#) (int chan, int lsb, int msb)
Callback function implementing `Mf_pitchbend()`.
- int [m2m_chanpressure](#) (int chan, int pressure)
Callback function implementing `Mf_chanpressure()`.

9.18.1 Detailed Description

This module provides functions for a basic MIDI-to-MIDI conversion application.

Library:

midicvt application portion of libmidifilex

Author

Chris Ahlstrom and many others; see documentation

Date

2014-04-29

Last Edits:

2014-05-13

Version

\$Revision\$

License:

GNU GPL

Definition in file [midicvt_m2m.h](#).

9.18.2 Function Documentation

9.18.2.1 int m2m_chanpressure (int *chan*, int *pressure*)

Callback function implementing Mf_chanpressure().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0xDn .

A channel-pressure command has the following 3 bytes:

1. Delta-time byte. A typical value is 0x60 (96).
2. Channel-pressure byte. 0xDn .
3. Channel-pressure value byte. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pressure</i>	Provides the patch/program number to write.

Returns

Returns true, always.

Definition at line 561 of file midicvt_m2m.c.

9.18.2.2 int m2m_noff (int *chan*, int *pitch*, int *vol*)

Callback function implementing Mf_off().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0x8n .

Very similar to Note-On messages [see [m2m_non\(\)](#)].

In our sample file, ex1.mid, there are no Note-Off messages, only Note-On messages with a velocity of 0.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>vol</i>	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 371 of file midicvt_m2m.c.

9.18.2.3 int m2m_non (int *chan*, int *pitch*, int *vol*)

Callback function implementing Mf_on().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0x9n .

The MIDI bytes for a note on message are 4 in number:

1. Delta-time byte. A typical value is 0x60 (96).
2. Note on byte. 0x9n . This byte is a status byte having two parts:
 - The first (most significant) nybble of this byte is 0x9.
 - The least significant nybble holds the channel number, ranging from 0x0 to 0xF. However, in our sample file, ex1.mid, there see to be notes in which the note-on byte is *missing!!!* This may be correlated to having a note velocity of 0, but not sure about that. There are no Note-Off messages in that file, by the way.
3. Note value byte. This value ranges from 0 to 127.
4. Note velocity byte. This value ranges from 0 to 127. It seems that a note off can be made by setting this value to zero. Again, not sure about that.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>vol</i>	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 335 of file midicvt_m2m.c.

9.18.2.4 int m2m_parameter (int *chan*, int *control*, int *value*)

Callback function implementing Mf_parameter().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0xBn .

This function prints the Control Change message. This message has different parameters than the Note On/Off messages.

The MIDI bytes for a control-change message are 4 in number:

1. Delta-time byte. A typical value is 0x60 (96).
2. Control-change byte. 0xBn . This byte is a status byte having two parts, the command nybble and the channel nybble.
3. Control-change value byte. This value ranges from 0 to 127. It determines which controller (e.g. pitch or sustain) is changed.
4. Controller value byte. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>control</i>	Provides the MIDI controller number to write.
<i>value</i>	Provides the MIDI controller parameter value to write.

Returns

Returns true, always.

Definition at line 450 of file `midicvt_m2m.c`.

9.18.2.5 int m2m_pitchbend (int *chan*, int *lsb*, int *msb*)

Callback function implementing `Mf_pitchbend()`.

No longer static, so that it can be using in the C++ program `midicvtp`.

Command: `0xEn` .

- `EC 11` mm. Pitch Wheel Change. Provides the least significant and most significant 14 bits.

1. Delta-time byte. A typical value is `0x60` (96).
2. Pitch-bend byte. `0xEn` .
3. Pitch, least-significat value byte. This value ranges from 0 to 127. It provides the lowest 7 bits of the pitch-bend parameter.
4. Pitch, most-significat value byte. It provides the highest 7 bits of the pitch-bend parameter.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>lsb</i>	Provides the least-significant bits of the MIDI pitch-wheel value to write.
<i>msb</i>	Provides the most-significant bits of the MIDI pitch-wheel value to write.

Returns

Returns true, always.

Definition at line 493 of file `midicvt_m2m.c`.

9.18.2.6 int m2m_pressure (int *chan*, int *pitch*, int *pressure*)

Callback function implementing `Mf_pressure()`.

No longer static, so that it can be using in the C++ program `midicvtp`.

Command: `0xAn` .

Very similar to Note-On messages [see [m2m_non\(\)](#)].

Parameters

<i>chan</i>	Provides the channel number to write.
-------------	---------------------------------------

<i>pitch</i>	Provides the MIDI note number to write.
<i>pressure</i>	Provides the MIDI polyphonic key pressure value to write.

Returns

Returns true, always.

Definition at line 404 of file midicvt_m2m.c.

9.18.2.7 int m2m_program (int *chan*, int *program*)

Callback function implementing Mf_program().

No longer static, so that it can be using in the C++ program midicvtp.

Command: 0xCn pp.

A program-change (patch change) command has the following 3 bytes:

1. Delta-time byte. A typical value is 0x60 (96).
2. Program-change byte. 0xCn .
3. Program/patch number byte nn. This value ranges from 0 to 127.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>program</i>	Provides the patch/program number to write.

Returns

Returns true, always.

Definition at line 528 of file midicvt_m2m.c.

9.18.2.8 void midicvt_initfuncs_m2m (void)

Makes the function assignments needed by the midifile library when converting a MIDI file to text.

We already handle the end of track. See m2m_trend for an explanation.

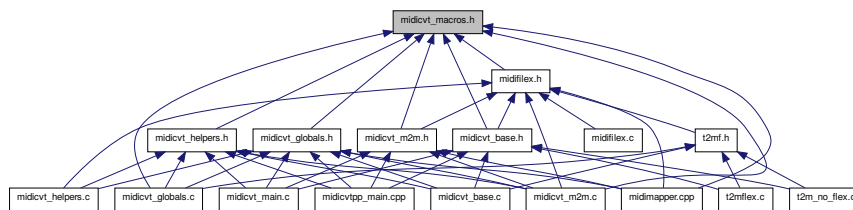
Mf_eot = m2m_meot;

Definition at line 1011 of file midicvt_m2m.c.

9.19 midicvt_macros.h File Reference

This module provides macros for generating simple messages, MIDI parameters, and more.

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



Macros

- `#define MIDI_NOTE_MIN 0`
MIDI Manifest Constants:
- `#define MIDI_NOTE_COUNT (MIDI_NOTE_MAX + 1)`
Defines the maximum number of unique MIDI notes.
- `#define MIDI_EVENT_NOTE_OFF 0`
Defines the values of the MIDI events "note off" and "note on".
- `#define MIDI_NOTE_OCTAVE 12`
Defines the number of notes in a MIDI octave.
- `#define MIDI_NOTE(x) ((char) (x))`
Converts an integer expression to a MIDI note.
- `#define MIDI_VELOCITY_MIN 0`
Defines some standard velocity values.
- `#define MIDI_UNINITIALIZED (-1)`
Defines a value used to indicate that certain values are still in their uninitialized state.
- `#define MIDI_NOTE_ILLEGAL (-1)`
Defines a value used to indicate that a note function returned an illegal note.
- `#define nullptr 0`
Language macros:
- `#define not_nullptr(x) ((x) != nullptr)`
Provides a way to declare functions as having either a C++ or C interface.
- `#define is_nullptr(x) ((x) == nullptr)`
Test for being an invalid pointer.
- `#define false 0`
Provides the "false" value of the wbool_t type definition.
- `#define true 1`
Provides the "true" value of the wbool_t type definition.
- `#define errprint(x) fprintf(stderr, "? %s\n", x)`
Provides an error reporting macro (which happens to match Chris's XPC error function).
- `#define errprintf(fmt, x) fprintf(stderr, fmt, x)`
Provides an error reporting macro that requires a sprintf() format specifier as well.
- `#define warnprint(x) fprintf(stderr, "! %s\n", x)`
Provides a warning reporting macro (which happens to match Chris's XPC error function).
- `#define warnprintf(fmt, x) fprintf(stderr, fmt, x)`
Provides an error reporting macro that requires a sprintf() format specifier as well.
- `#define infoprint(x) fprintf(stderr, "* %s\n", x)`
Provides an information reporting macro (which happens to match Chris's XPC information function).
- `#define infoprintf(fmt, x) fprintf(stderr, fmt, x)`
Provides an error reporting macro that requires a sprintf() format specifier as well.

Typedefs

- typedef int [cbool_t](#)

A more obvious boolean type.

9.19.1 Detailed Description

This module provides macros for generating simple messages, MIDI parameters, and more.

Library:

libmidifilex

Author

Chris Ahlstrom and other authors; see documentation

Date

2013-11-17

Last Edits:

2014-04-20

Version

\$Revision\$

License:

GNU GPL

The macros in this file cover:

- Default values of waonc parameters.
- MIDI manifest constants.
- Language-support macros.
- Error and information output macros.

Copyright (C) 2013-2013 Chris Ahlstrom ahlstrom@users.sourceforge.net

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

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

9.19.2 Macro Definition Documentation

9.19.2.1 `#define MIDI_NOTE_MIN 0`

MIDI Manifest Constants:

The first set of macros provides default values for the minimum/maximum detection code. We want to be able to know the highest and lowest MIDI notes that were generated, as a help to running the application again.

Definition at line 50 of file `midicvt_macros.h`.

9.19.2.2 `#define not_nullptr(x) ((x) != nullptr)`

Provides a way to declare functions as having either a C++ or C interface.

Test for being a valid pointer.

Definition at line 136 of file `midicvt_macros.h`.

9.19.2.3 `#define nullptr 0`

Language macros:

Provides an alternative to `NULL`.

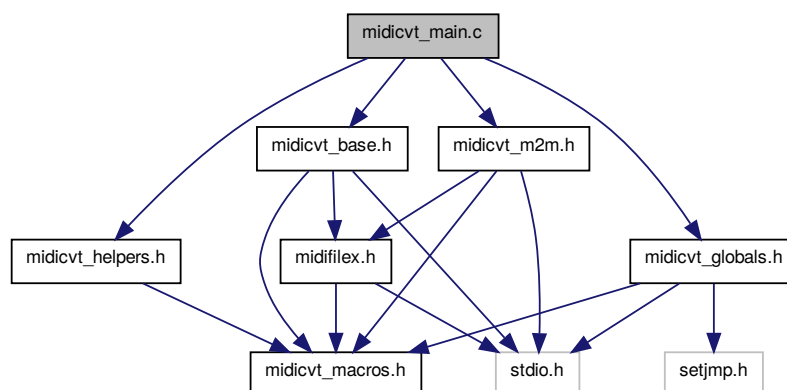
Definition at line 110 of file `midicvt_macros.h`.

9.20 `midicvt_main.c` File Reference

This module creates the `midicvt` program for translating between MIDI files and text files.

```
#include <midicvt_base.h>
#include <midicvt_helpers.h>
#include <midicvt_globals.h>
#include <midicvt_m2m.h>
```

Include dependency graph for `midicvt_main.c`:



Functions

- int [main](#) (int argc, char *argv[])
Provides the entry-point for the midicvt program.

Variables

- static const char *const [s_help_version](#) = "midicvt v 0.2 2014-05-08"
Version string.

9.20.1 Detailed Description

This module creates the midicvt program for translating between MIDI files and text files.

Library:

midicvt application

Author

Major modifications by Chris Ahlstrom

Date

2014-04-09

Last Edits:

2014-05-08

Version

\$Revision\$

License:

GNU GPL

By translating a MIDI file to a text file, midicvt allows one to use standard UNIX/Linux/OSX/Windows tools to modify a MIDI file. For example, one can write a script to re-map the track that contains a non-standard MIDI drum kit into a General MIDI drumkit. This can free the user from having to buy a proprietary application that includes that ability as one of its "features".

This program was derived from the midicomp project, which itself has a heritage of legacy code, some dating back to the Atari ST! Another project derived from midicomp is the midi2text project. That project takes some iffy shortcuts, so the only features taken from that project are bug-fixes, and [SOON] the ability to call the program using different app-names.

This program also takes advantage of GNU autotools, and refactors the global variables into their own module, and the midifile project into its own library.

Finally, Doxygen markup is being added to make it a bit easier to grok this project. It's all just cleanup, but I had fun doing it.

Definition in file [midicvt_main.c](#).

9.20.2 Function Documentation

9.20.2.1 int main (int argc, char * argv[])

Provides the entry-point for the midicvt program.

Parameters

<i>argc</i>	Provides the standard count of the number of command-line arguments, including the name of the program.
<i>argv</i>	Provides the command-line arguments as an array of pointers.

Returns

Returns a 0 value if the application succeeds, and a non-zero value otherwise.

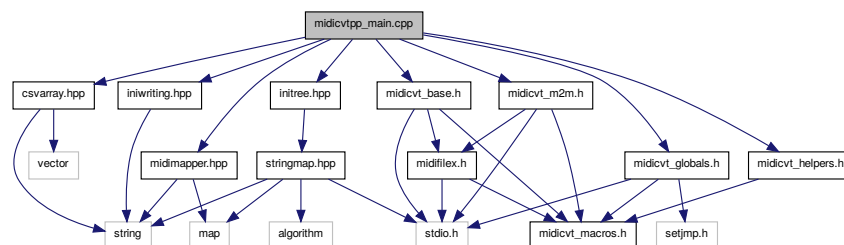
Definition at line 85 of file `midicvt_main.c`.

9.21 midicvtp_main.cpp File Reference

This module creates the `midicvtp` program for translating between MIDI files and text files, and doing some basic remapping operations.

```
#include <csvarray.hpp>
#include <initree.hpp>
#include <iniwritng.hpp>
#include <midicvt_base.h>
#include <midicvt_helpers.h>
#include <midicvt_globals.h>
#include <midicvt_m2m.h>
#include <midimapper.hpp>
```

Include dependency graph for `midicvtp_main.cpp`:

**Functions**

- static void `midicvtp_help()`
Provides the help text for the `midicvt` program.
- static bool `midicvtp_parse(int argc, char *argv[])`
Parses `midicvtp` the command-line for options.
- int `main(int argc, char *argv[])`
Provides the entry-point for the `midicvtp` program.

Variables

- static const char *const `s_help_version` = "midicvtp v 0.1 2014-06-01"
Provides the version string for this program.
- static const char *const `gs_help_usage_1`

Addition help string, which supplements the help for the C program midicvt, contained in libmidifilex/src/midicvt_helpers.h.

- static bool `s_write_csv_drum = false`
Holds the flag needed for the `-csv-drum` options.
- static bool `s_write_csv_patch = false`
Holds the flag needed for the `-csv-patch` options.
- static std::string `s_csv_in_filename`
For the `-csv-xxxx` options, holds the name of the input CSV (comma-separated value) file.
- static std::string `s_ini_out_filename`
For the `-csv-xxxx` options, holds the name of the output INI file.
- static std::string `s_ini_in_filename = ""`
For the `-m2m` option, holds the name of the input INI file that contains the MIDI mapping information for the MIDI-to-MIDI conversion.
- static bool `s_m2m_reversal = false`
Indicates that the `-m2m` mapping will be treated as a reverse mapping, and is specified by the `-reverse` option.
- static std::string `s_mapping_name = "midicvtp"`
Provides the name of the MIDI remapping process.
- static int `s_filter_channel = -1`
Provides the channel number to be affected by channel extraction (`-extract`) or rejection (`-reject`).
- static bool `s_rejection_on = false`
Indicates if the filtered channel is to be rejected (as opposed to extracted).

9.21.1 Detailed Description

This module creates the midicvtp program for translating between MIDI files and text files, and doing some basic remapping operations.

Library:

midicvtp application

Author

Chris Ahlstrom

Date

2014-04-19

Last Edits:

2014-06-01

Version

\$Revision\$

License:

GNU GPL

This C++ program extends the C program midicvt by providing some remapping functionality.

This can save the user from having to create an Awk or Perl script to do some common remapping operations.

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

9.21.2 Function Documentation

9.21.2.1 `int main (int argc, char * argv[])`

Provides the entry-point for the midicvtp program.

Parameters

<i>argc</i>	Provides the standard count of the number of command-line arguments, including the name of the program.
<i>argv</i>	Provides the command-line arguments as an array of pointers.

Returns

Returns a 0 value if the application succeeds, and a non-zero value otherwise.

Definition at line 333 of file midicvtp_main.cpp.

9.21.2.2 static void midicvtp_help () [static]

Provides the help text for the midicvt program.

Parameters

<i>version</i>	Provides the optional version information.
----------------	--

Definition at line 96 of file midicvtp_main.cpp.

9.21.2.3 static bool midicvtp_parse (int argc, char * argv[]) [static]

Parses midicvtp the command-line for options.

First, calls [midicvt_parse\(\)](#) to get the midicvt options, and then parses for the the midicvtp-specific options.

Parameters

<i>argc</i>	The standard argument count for the command-line.
<i>argv</i>	The standard argument list for the command-line.

Returns

Returns 0 if the program succeeded, and a non-zero number if the program fails or help was obtained.

Definition at line 188 of file midicvtp_main.cpp.

9.21.3 Variable Documentation**9.21.3.1 const char* const s_help_version = "midicvtp v 0.1 2014-06-01" [static]**

Provides the version string for this program.

Please leave "midicvtp" as the first characters of this version string. The midicvt command-line parser checks this value if C++-only options are provided.

Definition at line 60 of file midicvtp_main.cpp.

9.21.3.2 std::string s_ini_in_filename = "" [static]

For the `-m2m` option, holds the name of the input INI file that contains the MIDI mapping information for the MIDI-to-MIDI conversion.

The default name used to be `../doc/GM_PSS-790_Drums.ini`, useful mainly for testing. But we want to be able to do `-m2m` without requiring an INI file, in the same way the C code does it.

Definition at line 138 of file midicvtp_main.cpp.

9.21.3.3 `std::string s_mapping_name = "midicvtp"` [static]

Provides the name of the MIDI remapping process.

This name is just a tag name for output, and defaults to "midicvtp". However, the `--testing` option changes it to "testing", which is an internal signal for unknown purposes.

Definition at line 154 of file `midicvtp_main.cpp`.

9.21.3.4 `bool s_write_csv_drum = false` [static]

Holds the flag needed for the `--csv-drum` options.

See the help text for this option.

Definition at line 106 of file `midicvtp_main.cpp`.

9.21.3.5 `bool s_write_csv_patch = false` [static]

Holds the flag needed for the `--csv-patch` options.

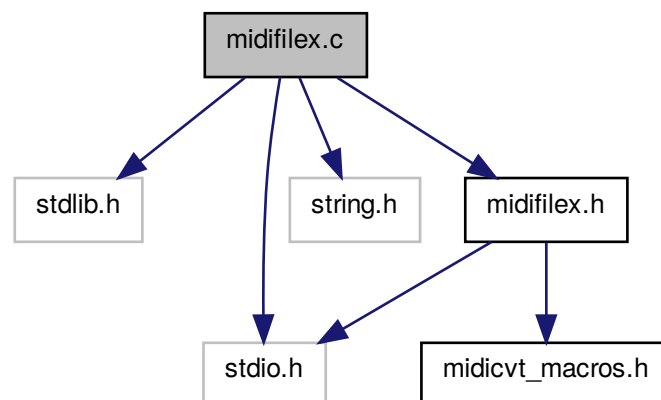
See the help text for this option.

Definition at line 113 of file `midicvtp_main.cpp`.

9.22 `midifilex.c` File Reference

This module provides functions for handling the reading and writing of MIDI files.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <midifilex.h>
Include dependency graph for midifilex.c:
```



Functions

- static void `mferror` (char *s)
Reports an error if the `Mf_error` callback has been assigned, then exits with an error-code of 1.
- static void `mfreport` (char *s)
Reports an information message.
- static `cbool_t` `mfreportable` (void)
- static void `badbyte` (int c)
Provides an error report for a bad byte.
- static int `eputc` (unsigned char c)
Writes a single character.
- static void `biggermsg` (void)
Re-allocates the message buffer by the standard increment of 128 bytes.
- static void `msginit` (void)
Sets `s_message_index` to 0.
- static char * `msg` (void)
Provides a pointer to the message buffer.
- static int `msgleng` (void)
Provides the current index into the message buffer.
- static void `msgadd` (int c)
Adds a character to the message buffer.
- static void `chanmessage` (int status, int c1, int c2)
Handles a channel message.
- static void `sysex` (void)
Handle a system-exclusive message.
- static long `readvarinum` (void)
Read a varying-length number.
- static long `to32bit` (int c1, int c2, int c3, int c4)
Convert a quad of integers into a 32-bit number.
- static short int `to16bit` (int c1, int c2)
Convert a duo of integers into a 16-bit number.
- static long int `read32bit` (void)
Reads four bytes and converts them to a 32-bit number.
- static short int `read16bit` (void)
Reads two bytes and converts them to a 16-bit number.
- void `write32bit` (unsigned long data)
`write32bit()` and `write16bit()` are used to make sure that the byte order of the various data types remains constant between machines.
- static void `write16bit` (int data)
Writes two bytes of data, one byte at a time.
- static void `writevarlen` (unsigned long value)
Write multi-length bytes to MIDI format files.
- static void `metaevent` (int type)
Handles a meta-event.
- static int `readmt` (char *s)
Reads through the "MThd" or "MTrk" header string.
- static void `readheader` (void)
Reads a header chunk.
- static `cbool_t` `readtrack` (void)
Reads a track chunk.
- void `mfread` (void)

- Calls [readheader\(\)](#), then calls [readtrack\(\)](#) while there is data to be read.*

 - void [midifile](#) (void)

Provided for backward compatibility with the original library.
 - void [mf_w_track_chunk](#) (int which_track, FILE *fp, int(*wtrack)(void))

Writes a track chunk.
 - void [mf_w_track_start](#) (int which_track, FILE *fp)

Reads and writes track information.
 - void [mf_w_header_chunk](#) (int format, int ntracks, int division)

Writes a header chunk.
 - void [mfwrite](#) (int format, int ntracks, int division, FILE *fp)

[mfwrite\(\)](#) is the only function you'll need to call to write out a MIDI file.
 - int [mf_w_midi_event](#) (unsigned long delta_time, unsigned int type, unsigned int chan, unsigned char *data, unsigned long size)

Library routine to [mf_write](#) a single MIDI track event in the standard MIDI file format.
 - int [mf_w_meta_event](#) (unsigned long delta_time, unsigned char type, unsigned char *data, unsigned long size)

Library routine to [mf_write](#) a single meta event in the standard MIDI file format.
 - void [mf_w_tempo](#) (unsigned long delta_time, unsigned long tempo)

Writes the tempo data.
 - unsigned long [mf_sec2ticks](#) (float secs, int division, unsigned int tempo)

Converts seconds to ticks.
 - float [mf_ticks2sec](#) (unsigned long ticks, int division, unsigned int tempo)

Provides the inverse of [mf_sec2ticks](#).
 - static [cbool_t](#) [readtrack_m2m](#) (void)

Reads a track chunk for MIDI-to-MIDI conversion.
 - void [mftransform](#) (void)

This function provides an alternative to [mfread\(\)](#) to better handle direct MIDI-to-MIDI conversions.

Variables

- int(* [Mf_getc](#))(void) = [nullptr](#)

Functions to be called while processing and writing the MIDI file.
- int [Mf_nomerge](#) = 0

1 => continue'd system exclusives are not collapsed.
- long [Mf_curtime](#) = 0L

Current time in delta-time units.
- static char * [s_message_buffer](#) = [nullptr](#)

The code below allows collection of a system exclusive message of arbitrary length.
- static int [s_message_size](#) = 0

Holds the size of the currently allocated message buffer.
- static int [s_message_index](#) = 0

Holds the index of next available location in [s_message_buffer](#).
- static int [s_chantype](#) []

Provides a helper array for the [readtrack\(\)](#) and [readtrack_m2m\(\)](#) functions.
- static int [s_laststat](#) = 0

Holds the last status code.
- static int [s_lastmeta](#) = 0

Holds the last meta event type.
- static long [s_track_header_offset](#) = 0

Saves the offset of the header portion of the current track.

9.22.1 Detailed Description

This module provides functions for handling the reading and writing of MIDI files.

Library:

midicvt application

Author

Other authors (see below), with modifications by Chris Ahlstrom,

Date

2014-04-08

Last Edits:

2014-06-02

Version

\$Revision\$

License:

GNU GPL

This file is based on midifile.c of the midifile library written by Tim Thompson (tjt@blink.att.com) and updated by Michael Czeiszperger (mike@pan.com). However, there were some bugs in the write code and Piet (see below) added some features he needed. He also changed some of the names to cope with the 7-character limit for external identifiers in the Sozobon compiler. He made an updated version of the library available, and split the read and write portions. However, he ended up with two midifile.c modules, each having differences, including differences that could cause bugs.

Chris (see below) updated the code to more modern C conventions (there's still more to do, such as adding more "const" items) and tried to improve the readability of the code). Also got a start on adding Doxygen markup to generate a more readable reference manual. Also added a Mf_report function pointer, to be enabled for easier diagnosis of MIDI conversion.

Piet van Oostrum, Dept of Computer Science, Utrecht University,
P.O. Box 80.089, 2508 TB Utrecht, The Netherlands
email: piet@cs.ruu.nl

Chris Ahlstrom, Charleston SC, USA
email: ahlstromcj@gmail.com

Discussion:

For converting a MIDI file to text, the code calls `mfread()`, which reads the MIDI file and then calls the set of callbacks (in the `midicvt_base.c` module) that convert the output to human-readable form.

For converting the human-readable text to MIDI, the code calls `midicvt_compile()`, which reads the header text. Then it calls `mfwrite()`, which calls essentially only the `my_writetrack()` callback. That callback knows how to parse the human-readable data and generate MIDI output for it.

When converting directly from MIDI to MIDI, we have two choices:

- Call `mfread()`, replacing the text-output callbacks in `midicvt_base.c` with the MIDI-output callbacks in `midicvt_m2m.c`. This has some minor issues.

- Call `readheader()` to get the initial information. Then call `mfwrite()` with a new write-track routine. However, this routine must also read MIDI [instead of flex data].

So it seems easier to use the first method, and hack it so it works correctly whether the output is human-readable or MIDI.

Definition in file `midifilex.c`.

9.22.2 Function Documentation

9.22.2.1 `static void badbyte (int c) [static]`

Provides an error report for a bad byte.

Since `mferror()` is called, the application will exit.

Parameters

<code>c</code>	Provides the bad byte, in integer format.
----------------	---

Definition at line 237 of file `midifilex.c`.

9.22.2.2 `static void biggermsg (void) [static]`

Re-allocates the message buffer by the standard increment of 128 bytes.

If it cannot allocate the new buffer, then `mferror()` is called.

Definition at line 324 of file `midifilex.c`.

9.22.2.3 `static void chanmessage (int status, int c1, int c2) [static]`

Handles a channel message.

As noted in the description of the `status` parameter, there are various supported messages. Depending on the type of message, one of the following callback calls will be made:

```
(void) (*Mf_off)(chan, c1, c2);
(void) (*Mf_on)(chan, c1, c2);
(void) (*Mf_pressure)(chan, c1, c2);
(void) (*Mf_parameter)(chan, c1, c2);
(void) (*Mf_program)(chan, c1);
(void) (*Mf_chanpressure)(chan, c1);
(void) (*Mf_pitchbend)(chan, c1, c2);
```

Parameters

<code>status</code>	Provides the type+channel byte. The type nybble ranges from 0x80 to 0xe0, and there are thus 7 values, corresponding to note-off, note-on, pressure, parameter (control), program (patch), channel pressure, and pitch-bend. The channel nybble ranges from 0 to 0xf (i.e. MIDI channels 1 to 16).
<code>c1</code>	Provides the first byte of the message. Depending on the message type, this value can be a MIDI note number, a control number (e.g. the number for a sustain message), a program/patch number, a pitch-wheel change's first value byte, or a system-common message.

<i>c2</i>	Provides the second byte of the message. Depending on the message type, this value can be a velocity value or control level value, the second byte of a pitch-wheel change, or this parameter can be left unused.
-----------	---

Definition at line 452 of file midifilex.c.

9.22.2.4 static int eputc (unsigned char *c*) [static]

Writes a single character.

If an error occurs, then this function calls [mfererror\(\)](#) and aborts. [But [mfererror\(\)](#) will call `exit()`.]

Parameters

<i>c</i>	Provides the character to output with the <code>Mf_putc()</code> callback function.
----------	---

Returns

Returns the return value of `Mf_putc()`. If this value is EOF, then [mfererror\(\)](#) is called.

Definition at line 259 of file midifilex.c.

9.22.2.5 static void metaevent (int *type*) [static]

Handles a meta-event.

The type of event is passed in as a parameter. The message itself is found in [msg\(\)](#), while the length of the message is provided by [msgleng\(\)](#).

Parameters

<i>type</i>	Provides the type of meta event. The following value sets are handled: <ul style="list-style-type: none"> • 0x00. Sequence number. The <code>Mf_seqnum()</code> callback is called. • 0x01 to 0x0f. Text event. 0x01 to 0x07 are standard text events, while the rest are "unrecognized". The <code>Mf_text()</code> callback is called. • 0x2f. End of track. The <code>Mf_eot()</code> callback is called. • 0x51. Set tempo. The <code>Mf_tempo()</code> callback is called. • 0x54. SMPTE. The <code>Mf_smppte()</code> callback is called. • 0x58. Time signature. The <code>Mf_timesig()</code> callback is called. • 0x59. Key signature. The <code>Mf_keysig()</code> callback is called. • 0x7f. Sequencer specific. The <code>Mf_sqspecific()</code> callback is called. • Default. Miscellaneous. The <code>Mf_metamisc()</code> callback is called.
-------------	--

Definition at line 763 of file midifilex.c.

9.22.2.6 unsigned long mf_sec2ticks (float *secs*, int *division*, unsigned int *tempo*)

Converts seconds to ticks.

Calculates the value of

$$\frac{1000 * \text{secs}}{4 * \text{division}} * \frac{1}{\text{tempo}}$$

Parameters

<i>secs</i>	Provides the seconds value to be converted.
<i>division</i>	Provides the division units.
<i>tempo</i>	Provides the tempo value

Returns

Returns the value of the seconds in ticks, as per the formula shown.

Definition at line 1736 of file midifilex.c.

9.22.2.7 float mf_ticks2sec (unsigned long *ticks*, int *division*, unsigned int *tempo*)

Provides the inverse of mf_sec2ticks.

This routine converts delta times in ticks into seconds. The else statement is needed because the formula is different for tracks based on notes and tracks based on SMPTE times.

Parameters

<i>ticks</i>	Provides the ticks value to be converted.
<i>division</i>	Provides the division units.
<i>tempo</i>	Provides the tempo value

Returns

Returns the value of the ticks in seconds. as per the formula shown.

Warning

If 0, then this should throw an exception!!!

Definition at line 1762 of file midifilex.c.

9.22.2.8 void mf_w_header_chunk (int *format*, int *ntracks*, int *division*)

Writes a header chunk.

This involves writing the following values:

```
-# Header identifier "MThd" (using a tricky long integer), 32
  bits.
-# Chunk length, set to 6. 32 bits.
-# Format. 16 bits.
-# Number of tracks. 16 bits.
-# Divisions. 16 bits.
```

Parameters

<i>format</i>	Provides the format byte to describe the SMF type of the file.
<i>ntracks</i>	Provides the number of tracks in the file.
<i>division</i>	Provides the division parameter of the file.

Definition at line 1401 of file midifilex.c.

9.22.2.9 int mf_w_meta_event (unsigned long *delta_time*, unsigned char *type*, unsigned char * *data*, unsigned long *size*)

Library routine to mf_write a single meta event in the standard MIDI file format.

The format of a meta event is:

```
<delta-time><FF><type><length><bytes>
```

Parameters

<i>delta_time</i>	Provides the time in ticks since the last event.
<i>type</i>	Provides the type of meta event.
<i>data</i>	Provides A pointer to a block of chars containing the META EVENT, data.
<i>size</i>	Provides the length of the meta-event data.

Returns

Returns the number of bytes written for this meta-event. This funcio used to return the *size* parameter, but no one ever used the return value, and we need it in [midicvt_m2m.c](#) to adjust the track-size correctly.

Definition at line 1618 of file midifilex.c.

9.22.2.10 `int mf_w_midi_event (unsigned long delta_time, unsigned int type, unsigned int chan, unsigned char * data, unsigned long size)`

Library routine to mf_write a single MIDI track event in the standard MIDI file format.

The format is:

```
<delta-time> <event>
```

In this case, event can be any multi-byte midi message, such as "note on", "note off", etc.

Note

This routine uses an array to pass in variable numbers of parameters. Here's an alternate function signature to consider for the future, where the bytes are passed in a variable-parameter list:

```
int mf_w_midi_event
(
    unsigned long delta_time,
    unsigned int type,
    unsigned int chan,
    unsigned long size,
    ...
)
```

Parameters

<i>delta_time</i>	Provides the time in ticks since the last event.
<i>type</i>	Provides the type of event.
<i>chan</i>	Provides The midi channel.
<i>data</i>	Provides A pointer to a block of chars containing the META EVENT, data.
<i>size</i>	Provides The length of the midi-event data.

Returns

Returns the *size* parameter.

Definition at line 1550 of file midifilex.c.

9.22.2.11 `void mf_w_tempo (unsigned long delta_time, unsigned long tempo)`

Writes the tempo data.

All tempos are written as 120 beats/minute, expressed in microseconds/quarter note.

Parameters

<i>delta_time</i>	Provides the time in ticks since the last event.
<i>tempo</i>	Provides the temp value to write.

Definition at line 1699 of file midifilex.c.

9.22.2.12 void mf_w_track_chunk (int which_track, FILE * fp, int(*) (void) wtrack)

Writes a track chunk.

This involves the following steps:

```
-# Write "MTrk" (as a tricky #define in midifilex.h)
-# Write 0 as the track length.
-# Call the wtrack callback.
-# Write 0, meta-event, end-of-track, and 0.
-# Go back and rewrite the 32-bit track header.
-# Rewrite the 32-bit track length.
```

Note

Why not use the global Mf_wtrack() function instead of passing it as a parameter here?

Parameters

<i>which_track</i>	Indicates the track number of the track to be written. If -1, then the track is a tempo-track.
<i>fp</i>	The output file descriptor.
<i>wtrack</i>	The function to call to do that actual writing. Usually, this function is either Mf_wtempotrack or Mf_wtrack.

Definition at line 1265 of file midifilex.c.

9.22.2.13 void mf_w_track_start (int which_track, FILE * fp)

Reads and writes track information.

Parameters

<i>which_track</i>	Indicates the track number of the track to be written. If -1, then the track is a tempo-track.
<i>fp</i>	The output file descriptor.
<i>wtrack</i>	The function to call to do that actual writing. Usually, this function is either Mf_wtempotrack or Mf_wtrack.

Definition at line 1357 of file midifilex.c.

9.22.2.14 static void mferror (char * s) [static]

Reports an error if the Mf_error callback has been assigned, then exits with an error-code of 1.

Parameters

<i>s</i>	Provides the error message.
----------	-----------------------------

Definition at line 190 of file midifilex.c.

9.22.2.15 void mfred (void)

Calls [readheader\(\)](#), then calls [readtrack\(\)](#) while there is data to be read.

Note

This function and [mfwrite\(\)](#) are the only non-static functions in this file? Not any more!

Definition at line 1204 of file midifilex.c.

9.22.2.16 static void mfreport (char * s) [static]

Reports an information message.

Useful in debugging. To enable it, simply set `Mf_report` equal to your reporting function. In its usage inside this module, it acts very similar to the user's callback functions. This makes it easy to compare the user's output to what was actually encountered in the MIDI file.

Parameters

<code>s</code>	Provides the error message.
----------------	-----------------------------

Definition at line 210 of file midifilex.c.

9.22.2.17 static cbool_t mfreportable (void) [static]

Returns

Returns true if the `Mf_report` function is enabled.

Definition at line 222 of file midifilex.c.

9.22.2.18 void mftransform (void)

This function provides an alternative to [mfread\(\)](#) to better handle direct MIDI-to-MIDI conversions.

The [mfread\(\)](#) function, with suitable callbacks (see the [midicvt_m2m.c](#) module) almost works for MIDI-to-MIDI conversions. However, the existing "write-track" callbacks are a bit difficult to use because they actually require some ability to read input, and don't help keep track of file pointers.

Calls [readheader\(\)](#), which works fine with the [m2m_header\(\)](#) callback. then calls [readtrack_m2m\(\)](#) while there is data to be read.

Definition at line 1949 of file midifilex.c.

9.22.2.19 void mfwrite (int format, int ntracks, int division, FILE * fp)

[mfwrite\(\)](#) is the only function you'll need to call to write out a MIDI file.

First, the `Mf_putc()` and `Mf_wtrack()` callbacks are checked to make sure that they have been assigned to callback functions.

Then `mf_w_header_chunk(format, ntracks, division)` is called.

If the format is SMF 1, then a track chunk is written by passing the `Mf_wtempotrack()` callback to the [mf_w_track_chunk\(\)](#) function.

Finally, the rest of the tracks are written by passing the `Mf_wtrack()` callback to the [mf_w_track_chunk\(\)](#) function.

Parameters

<i>format</i>	Indicates the level of SMF (standard MIDI file) support. <ul style="list-style-type: none"> • 0: Single multi-channel track. • 1: Multiple simultaneous tracks. • 2: One or more sequentially independent single track patterns.
<i>ntracks</i>	Provides the number of tracks in the file.
<i>division</i>	This parameter is kind of tricky, it can represent two things, depending on whether it is positive or negative (bit 15 set or not). If bit 15 of division is zero, bits 14 through 0 represent the number of delta-time "ticks" which make up a quarter note. If bit 15 of division is a one, delta-times in a file correspond to subdivisions of a second similar to SMPTE and MIDI time code. In this format bits 14 through 8 contain one of four values - 24, -25, -29, or -30, corresponding to the four standard SMPTE and MIDI time code frame per second formats, where -29 represents 30 drop frame. The second byte consisting of bits 7 through 0 corresponds the the resolution within a frame. Refer the Standard MIDI Files 1.0 spec for more details.
<i>fp</i>	This should be the open file pointer to the file you want to write. It will have be a global in order to work with <code>Mf_putc</code> .

Definition at line 1462 of file `midifilex.c`.

9.22.2.20 void midifile (void)

Provided for backward compatibility with the original library.

This function simply calls [mfread\(\)](#).

Definition at line 1220 of file `midifilex.c`.

9.22.2.21 static char* msg (void) [static]

Provides a pointer to the message buffer.

Returns

Returns the `s_message_buffer` static variable.

Definition at line 366 of file `midifilex.c`.

9.22.2.22 static void msgadd (int c) [static]

Adds a character to the message buffer.

If necessary, it re-allocates a larger message buffer by calling [biggermsg\(\)](#).

Parameters

<i>c</i>	The character to add to the message buffer.
----------	---

Definition at line 395 of file `midifilex.c`.

9.22.2.23 static int msgleng (void) [static]

Provides the current index into the message buffer.

Returns

Returns the value of `s_message_index`.

Definition at line 379 of file `midifilex.c`.

9.22.2.24 static short int read16bit (void) [static]

Reads two bytes and converts them to a 16-bit number.

Returns

Returns the total value represented by the two characters.

Definition at line 664 of file `midifilex.c`.

9.22.2.25 static long int read32bit (void) [static]

Reads four bytes and converts them to a 32-bit number.

Todo

- We need to use an actual 32-bit return type for 64-bit systems.
- Do we need to make the temp variables volatile; can the compiler reorder them?

Returns

Returns the total value represented by the four characters.

Definition at line 647 of file `midifilex.c`.

9.22.2.26 static void readheader (void) [static]

Reads a header chunk.

First, `readmt()` is called to verify that "MThd" was retrieved from the file. If this succeeds, then the following items are read:

1. Length of the header (32 bits). This value is saved in the global variable `s_Mf_toberead`.
2. Format of the header (16 bits).
3. Number of tracks (16 bits).
4. The division value (16 bits).

The last three values are passed to the `Mf_header()` callback function as parameters. This function should reduce the value of `s_Mf_toberead` as bytes are processed.

If `s_Mf_toberead` is still greater than 0, then the extra characters are flushed by calling `egetc()` `s_Mf_toberead` times.

Definition at line 989 of file `midifilex.c`.

9.22.2.27 static int readmt (char * s) [static]

Reads through the "MThd" or "MTrk" header string.

Characters are read via the `Mf_getc()` callback function. If the characters read do not match the expected string, then a fatal error occurs.

Parameters

s	Provides the string that is expected to be read from the file.
----------	--

Returns

Returns the last character obtained, or EOF if the match was unable to be consumated.

Definition at line 947 of file midifilex.c.

9.22.2.28 static cbool_t readtrack (void) [static]

Reads a track chunk.

First, [readmt\(\)](#) is called to verify that "MTrk" was retrieved from the file. If this succeeds, then this function reads the length of the header (32 bits). This value is saved in the global variable s_Mf_toberead. Then Mf_currttime is set to 0. The Mf_starttrack() callback is called.

While s_Mf_toberead is non-zero, a byte is read and the following events are checked:

- 0xff. Meta event.
- 0xf0. System exclusive message.
- 0xf7. Sysex continuation or arbitrary data.

A lot of other processing is done (see the code), and then the Mf_endtrack() callback is called.

Note

Be sure to compare this function to the new variant, [readtrack_m2m\(\)](#).

Returns

Returns true if the "MTrk" marker was found.

Note

The test file ex1.mid has the Note On byte missing from the notes after the first two! However, the note value, which is in c, lets us reach here, and the code then copies status (which currently holds the Note On byte) into c, effectively restoring the Note On byte.

Definition at line 1065 of file midifilex.c.

9.22.2.29 static cbool_t readtrack_m2m (void) [static]

Reads a track chunk for MIDI-to-MIDI conversion.

This function is a modified version of [readtrack\(\)](#) that works better for direct MIDI-to-MIDI conversion using the callbacks defined in the [midicvt_m2m.c](#) module.

A new function was made from [readtrack\(\)](#) in order to better avoid breaking that function for existing functionality.

Returns

Returns true if the "MTrk" marker was found.

Note

The test file ex1.mid has the Note On byte missing from the notes after the first two! However, the note value, which is in c, lets us reach here, and the code then copies status (which currently holds the Note On byte) into c, effectively restoring the Note On byte.

Definition at line 1802 of file midifilex.c.

9.22.2.30 static long readvarinum (void) [static]

Read a varying-length number.

A variable-length quantity is a MIDI number that is represented by a string of bytes where all bytes but the last have bit 7 set. In each byte, only the 7 least-significant bits provide the numeric value.

- Numbers between 0 and 127 (0x7F) are represented by a single byte.
- 0x80 is represented as "0x81 0x00".
- 0x0FFFFFFF (the largest number) is represented as "0xFF 0xFF 0xFF 0x7F".

This function doesn't return the number of characters it took, it returns the value of the varying-length number.

Definition at line 565 of file midifilex.c.

9.22.2.31 static void sysex (void) [static]

Handle a system-exclusive message.

The `msgleng()` and `msg()` values are passed to the `Mf_sysex()` callback function.

Definition at line 530 of file midifilex.c.

9.22.2.32 static short int to16bit (int c1, int c2) [static]

Convert a duo of integers into a 16-bit number.

Parameters

<i>c1</i>	Provides the most-significant portion of the number. This portion gets shifted leftward by 8 bits once.
<i>c2</i>	Provides the least-significant portion of the number. This portion gets shifted leftward not at all.

Returns

The total value represented by the two parameters is returned.

Definition at line 629 of file midifilex.c.

9.22.2.33 static long to32bit (int c1, int c2, int c3, int c4) [static]

Convert a quad of integers into a 32-bit number.

Parameters

<i>c1</i>	Provides the most-significant portion of the number. This portion gets shifted leftward by 8 bits three times.
<i>c2</i>	Provides the second portion of the number.
<i>c3</i>	Provides the third portion of the number.
<i>c4</i>	Provides the least-significant portion of the number. This portion gets shifted leftward not at all.

Returns

The total value represented by the four parameters is returned.

Definition at line 604 of file midifilex.c.

9.22.2.34 `static void write16bit (int data) [static]`

Writes two bytes of data, one byte at a time.

Parameters

<i>data</i>	Provides the 16 bits of data to be written, one byte at a time.
-------------	---

Definition at line 705 of file midifilex.c.

9.22.2.35 void write32bit (unsigned long *data*)

[write32bit\(\)](#) and [write16bit\(\)](#) are used to make sure that the byte order of the various data types remains constant between machines.

This helps make sure that the code will be portable from one system to the next. It is slightly dangerous that it assumes that longs have at least 32 bits and ints have at least 16 bits, but this has been true at least on PCs, UNIX machines, and Macintosh's.

Todo Provide the proper 32-bit data types needed to do this more portably.

Parameters

<i>data</i>	Provides the 32 bits of data to be written, one byte at a time.
-------------	---

Definition at line 689 of file midifilex.c.

9.22.2.36 static void writevarlen (unsigned long *value*) [static]

Write multi-length bytes to MIDI format files.

Parameters

<i>value</i>	Provides the value to be written.
--------------	-----------------------------------

Definition at line 719 of file midifilex.c.

9.22.3 Variable Documentation

9.22.3.1 int s_chantype[] [static]

Initial value:

```
=
{
    0, 0, 0, 0, 0, 0, 0, 0,
    2, 2, 2, 2, 1, 1, 2, 0
}
```

Provides a helper array for the [readtrack\(\)](#) and [readtrack_m2m\(\)](#) functions.

This static array is indexed by the high half of a status byte. Its value is either the number of bytes needed (1 or 2) for a channel message, or 0 (meaning it's not a channel message).

Definition at line 1031 of file midifilex.c.

9.22.3.2 char* s_message_buffer = nullptr [static]

The code below allows collection of a system exclusive message of arbitrary length.

The message buffer is expanded as necessary. The only visible data/routines are [msginit\(\)](#), [msgadd\(\)](#), [msg\(\)](#), [msgleng\(\)](#).

Definition at line 302 of file midifilex.c.

9.22.3.3 `long s_track_header_offset = 0` `[static]`

Saves the offset of the header portion of the current track.

This value is use only for m2m (MIDI-to-MIDI) processing.

Definition at line 1338 of file `midifilex.c`.

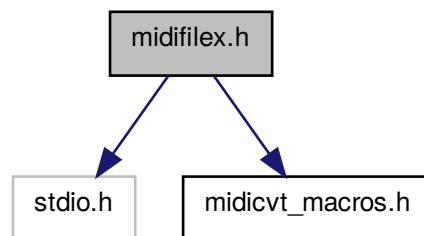
9.23 `midifilex.h` File Reference

This module provides functions for `libmidifilex`.

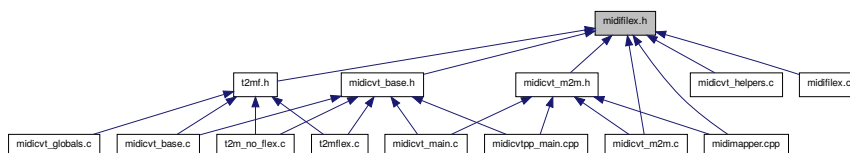
```
#include <stdio.h>
```

```
#include <midicvt_macros.h>
```

Include dependency graph for `midifilex.h`:



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



Macros

- `#define note_off 0x80`
Provides lower-case (!) macros for MIDI status commands.
- `#define damper_pedal 0x40`
Provides lower-case macros for the 7 bit controllers.
- `#define data_inc 0x60`
Provides parameter values.
- `#define non_reg_lsb 0x62`
Provides parameter selection.
- `#define meta_event 0xff`
Provides standard MIDI Files meta event definitions.

- `#define Seq_Circuits (0x01) /* Sequential Circuits Inc. */`

Provides the manufacturer's ID number.

- `#define MThd 0x4d546864L`

Provides miscellaneous definitions and macros.

Functions

- float `mf_ticks2sec` (unsigned long, int, unsigned int)
Provides the inverse of `mf_sec2ticks`.
- unsigned long `mf_sec2ticks` (float, int, unsigned int)
Converts seconds to ticks.
- void `write32bit` (unsigned long data)
`write32bit()` and `write16bit()` are used to make sure that the byte order of the various data types remains constant between machines.
- void `mfread` (void)
Calls `readheader()`, then calls `readtrack()` while there is data to be read.
- void `mftransform` (void)
This function provides an alternative to `mfread()` to better handle direct MIDI-to-MIDI conversions.
- void `mfwrite` (int, int, int, FILE *)
`mfwrite()` is the only function you'll need to call to write out a MIDI file.
- void `midifile` (void)
Provided for backward compatibility with the original library.
- void `mf_w_header_chunk` (int format, int ntracks, int division)
Writes a header chunk.
- void `mf_w_track_chunk` (int which_track, FILE *fp, int(*wtrack)(void))
Writes a track chunk.
- void `mf_w_track_start` (int which_track, FILE *fp)
Reads and writes track information.
- int `mf_w_midi_event` (unsigned long, unsigned int, unsigned int, unsigned char *, unsigned long)
Library routine to `mf_write` a single MIDI track event in the standard MIDI file format.
- void `mf_w_tempo` (unsigned long, unsigned long)
Writes the tempo data.
- int `mf_w_meta_event` (unsigned long, unsigned char, unsigned char *, unsigned long)
Library routine to `mf_write` a single meta event in the standard MIDI file format.

Variables

- int `Mf_nomerge`
1 => continue'd system exclusives are not collapsed.
- long `Mf_currttime`
Current time in delta-time units.
- int(* `Mf_getc`)(void)
Functions to be called while processing and writing the MIDI file.

9.23.1 Detailed Description

This module provides functions for libmidifilex.

Library:

libmidifilex

Author

Chris Ahlstrom and many other authors; see documentation

Date

2014-04-08

Last Edits:

2014-05-20

Version

\$Revision\$

License:

GNU GPL

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

9.23.2 Macro Definition Documentation

9.23.2.1 #define note_off 0x80

Provides lower-case (!) macros for MIDI status commands.

The most significant bit is 1.

Definition at line 118 of file midifilex.h.

9.23.3 Function Documentation

9.23.3.1 unsigned long mf_sec2ticks (float secs, int division, unsigned int tempo)

Converts seconds to ticks.

Calculates the value of

$$\frac{1000 * \text{secs}}{4 * \text{division}} * \frac{1}{\text{tempo}}$$

Parameters

<i>secs</i>	Provides the seconds value to be converted.
<i>division</i>	Provides the division units.
<i>tempo</i>	Provides the tempo value

Returns

Returns the value of the seconds in ticks, as per the formula shown.

Definition at line 1736 of file midifilex.c.

9.23.3.2 float mf_ticks2sec (unsigned long *ticks*, int *division*, unsigned int *tempo*)

Provides the inverse of mf_sec2ticks.

This routine converts delta times in ticks into seconds. The else statement is needed because the formula is different for tracks based on notes and tracks based on SMPTE times.

Parameters

<i>ticks</i>	Provides the ticks value to be converted.
<i>division</i>	Provides the division units.
<i>tempo</i>	Provides the tempo value

Returns

Returns the value of the ticks in seconds. as per the formula shown.

Warning

If 0, then this should throw an exception!!!

Definition at line 1762 of file midifilex.c.

9.23.3.3 void mf_w_header_chunk (int *format*, int *ntracks*, int *division*)

Writes a header chunk.

This involves writing the following values:

```
-# Header identifier "MThd" (using a tricky long integer), 32
  bits.
-# Chunk length, set to 6. 32 bits.
-# Format. 16 bits.
-# Number of tracks. 16 bits.
-# Divisions. 16 bits.
```

Parameters

<i>format</i>	Provides the format byte to describe the SMF type of the file.
<i>ntracks</i>	Provides the number of tracks in the file.
<i>division</i>	Provides the division parameter of the file.

Definition at line 1401 of file midifilex.c.

9.23.3.4 int mf_w_meta_event (unsigned long *delta_time*, unsigned char *type*, unsigned char * *data*, unsigned long *size*)

Library routine to mf_write a single meta event in the standard MIDI file format.

The format of a meta event is:

```
<delta-time><FF><type><length><bytes>
```

Parameters

<i>delta_time</i>	Provides the time in ticks since the last event.
<i>type</i>	Provides the type of meta event.
<i>data</i>	Provides A pointer to a block of chars containing the META EVENT, data.
<i>size</i>	Provides the length of the meta-event data.

Returns

Returns the number of bytes written for this meta-event. This funcio used to return the *size* parameter, but no one ever used the return value, and we need it in [midicvt_m2m.c](#) to adjust the track-size correctly.

Definition at line 1618 of file midifilex.c.

9.23.3.5 `int mf_w_midi_event (unsigned long delta_time, unsigned int type, unsigned int chan, unsigned char * data, unsigned long size)`

Library routine to mf_write a single MIDI track event in the standard MIDI file format.

The format is:

```
<delta-time> <event>
```

In this case, event can be any multi-byte midi message, such as "note on", "note off", etc.

Note

This routine uses an array to pass in variable numbers of parameters. Here's an alternate function signature to consider for the future, where the bytes are passed in a variable-parameter list:

```
int mf_w_midi_event
(
    unsigned long delta_time,
    unsigned int type,
    unsigned int chan,
    unsigned long size,
    ...
)
```

Parameters

<i>delta_time</i>	Provides the time in ticks since the last event.
<i>type</i>	Provides the type of event.
<i>chan</i>	Provides The midi channel.
<i>data</i>	Provides A pointer to a block of chars containing the META EVENT, data.
<i>size</i>	Provides The length of the midi-event data.

Returns

Returns the *size* parameter.

Definition at line 1550 of file midifilex.c.

9.23.3.6 `void mf_w_tempo (unsigned long delta_time, unsigned long tempo)`

Writes the tempo data.

All tempos are written as 120 beats/minute, expressed in microseconds/quarter note.

Parameters

<i>delta_time</i>	Provides the time in ticks since the last event.
<i>tempo</i>	Provides the tempo value to write.

Definition at line 1699 of file midifilex.c.

9.23.3.7 void mf_w_track_chunk (int *which_track*, FILE * *fp*, int(*)*(void) wtrack*)

Writes a track chunk.

This involves the following steps:

```
-# Write "MTrk" (as a tricky #define in midifilex.h)
-# Write 0 as the track length.
-# Call the wtrack callback.
-# Write 0, meta-event, end-of-track, and 0.
-# Go back and rewrite the 32-bit track header.
-# Rewrite the 32-bit track length.
```

Note

Why not use the global `Mf_wtrack()` function instead of passing it as a parameter here?

Parameters

<i>which_track</i>	Indicates the track number of the track to be written. If -1, then the track is a tempo-track.
<i>fp</i>	The output file descriptor.
<i>wtrack</i>	The function to call to do that actual writing. Usually, this function is either <code>Mf_wtempotrack</code> or <code>Mf_wtrack</code> .

Definition at line 1265 of file midifilex.c.

9.23.3.8 void mf_w_track_start (int *which_track*, FILE * *fp*)

Reads and writes track information.

Parameters

<i>which_track</i>	Indicates the track number of the track to be written. If -1, then the track is a tempo-track.
<i>fp</i>	The output file descriptor.
<i>wtrack</i>	The function to call to do that actual writing. Usually, this function is either <code>Mf_wtempotrack</code> or <code>Mf_wtrack</code> .

Definition at line 1357 of file midifilex.c.

9.23.3.9 void mfred (void)

Calls `readheader()`, then calls `readtrack()` while there is data to be read.

Note

This function and `mfwrite()` are the only non-static functions in this file? Not any more!

Definition at line 1204 of file midifilex.c.

9.23.3.10 void mftransform (void)

This function provides an alternative to `mfread()` to better handle direct MIDI-to-MIDI conversions.

The [mfread\(\)](#) function, with suitable callbacks (see the [midicvt_m2m.c](#) module) almost works for MIDI-to-MIDI conversions. However, the existing "write-track" callbacks are a bit difficult to use because they actually require some ability to read input, and don't help keep track of file pointers.

Calls [readheader\(\)](#), which works fine with the [m2m_header\(\)](#) callback. then calls [readtrack_m2m\(\)](#) while there is data to be read.

Definition at line 1949 of file midifilex.c.

9.23.3.11 void mfwrite (int *format*, int *ntracks*, int *division*, FILE * *fp*)

[mfwrite\(\)](#) is the only function you'll need to call to write out a MIDI file.

First, the [Mf_putc\(\)](#) and [Mf_wtrack\(\)](#) callbacks are checked to make sure that they have been assigned to callback functions.

Then [mf_w_header_chunk\(format, ntracks, division\)](#) is called.

If the format is SMF 1, then a track chunk is written by passing the [Mf_wtempotrack\(\)](#) callback to the [mf_w_track_chunk\(\)](#) function.

Finally, the rest of the tracks are written by passing the [Mf_wtrack\(\)](#) callback to the [mf_w_track_chunk\(\)](#) function.

Parameters

<i>format</i>	Indicates the level of SMF (standard MIDI file) support. <ul style="list-style-type: none"> • 0: Single multi-channel track. • 1: Multiple simultaneous tracks. • 2: One or more sequentially independent single track patterns.
<i>ntracks</i>	Provides the number of tracks in the file.
<i>division</i>	This parameter is kind of tricky, it can represent two things, depending on whether it is positive or negative (bit 15 set or not). If bit 15 of division is zero, bits 14 through 0 represent the number of delta-time "ticks" which make up a quarter note. If bit 15 of division is a one, delta-times in a file correspond to subdivisions of a second similar to SMPTE and MIDI time code. In this format bits 14 through 8 contain one of four values - 24, -25, -29, or -30, corresponding to the four standard SMPTE and MIDI time code frame per second formats, where -29 represents 30 drop frame. The second byte consisting of bits 7 through 0 corresponds to the resolution within a frame. Refer the Standard MIDI Files 1.0 spec for more details.
<i>fp</i>	This should be the open file pointer to the file you want to write. It will have to be a global in order to work with Mf_putc .

Definition at line 1462 of file midifilex.c.

9.23.3.12 void midifile (void)

Provided for backward compatibility with the original library.

This function simply calls [mfread\(\)](#).

Definition at line 1220 of file midifilex.c.

9.23.3.13 void write32bit (unsigned long *data*)

[write32bit\(\)](#) and [write16bit\(\)](#) are used to make sure that the byte order of the various data types remains constant between machines.

This helps make sure that the code will be portable from one system to the next. It is slightly dangerous that it assumes that longs have at least 32 bits and ints have at least 16 bits, but this has been true at least on PCs, UNIX machines, and Macintosh's.

Todo Provide the proper 32-bit data types needed to do this more portably.

Parameters

<code>data</code>	Provides the 32 bits of data to be written, one byte at a time.
-------------------	---

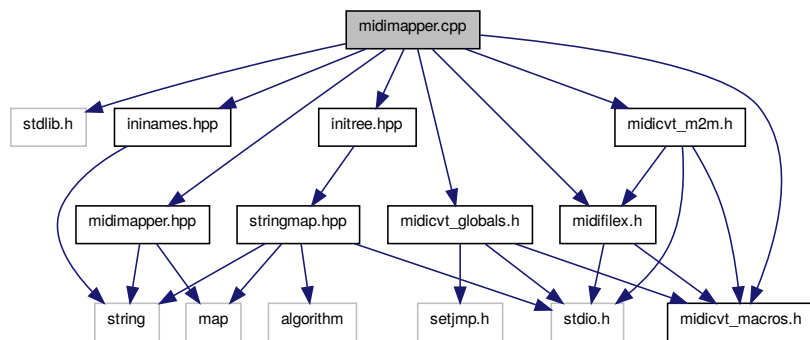
Definition at line 689 of file midifilex.c.

9.24 midimapper.cpp File Reference

This module provides functions for advanced MIDI/text conversions.

```
#include <stdlib.h>
#include <ininames.hpp>
#include <initree.hpp>
#include <midicvt_globals.h>
#include <midicvt_macros.h>
#include <midicvt_m2m.h>
#include <midifilex.h>
#include <midimapper.hpp>
```

Include dependency graph for midimapper.cpp:



Functions

- static int [midimap_non](#) (int chan, int pitch, int vol)

This static C function uses the singleton [midipp::midimapper](#) object to remap the channel and pitch values, and passes them to the corresponding callback function defined in the [midicvt_m2m.c](#) module.

- static int [midimap_noff](#) (int chan, int pitch, int vol)

This static C function uses the singleton [midipp::midimapper](#) object to remap the channel and pitch values, and passes them to the corresponding callback function defined in the [midicvt_m2m.c](#) module.

- static int [midimap_pressure](#) (int chan, int pitch, int pressure)

This static C function uses the singleton [midipp::midimapper](#) object to remap the channel and pitch values, and passes them to the corresponding callback function defined in the [midicvt_m2m.c](#) module.

- static int [midimap_patch](#) (int chan, int patch)

Remaps the channel and the program (patch) number, and passes them to the corresponding callback function defined in the [midicvt_m2m.c](#) module.

- static int [midimap_parameter](#) (int chan, int control, int value)

For filtering parameter/control messages.

- static int [midimap_pitchbend](#) (int chan, int lsb, int msb)

For filtering pitchbend messages.

- static int `midimap_chanpressure` (int chan, int pressure)

For filtering channel-pressure messages.

- void `show_maps` (const std::string &tag, const `midipp::midimapper` &container)

Writes out the contents of the pitch-map container out to stderr.

- void `midimap_init` (`midipp::midimapper` &mm)

Hooks the `midipp::midimapper` object provided to the global pointer for such objects, to allow the C routines to be able to use the features of the MIDI mapper.

Variables

- static `midipp::midimapper` * `gs_singleton_midimapper` = nullptr

Holds the single pointer to the `midipp::midimapper` object the caller created.

9.24.1 Detailed Description

This module provides functions for advanced MIDI/text conversions.

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-24

Last Edits:

2014-05-21

Version

\$Revision\$

License:

GNU GPL

This module uses some of the libmidifilex callbacks of the C modules in `libmidifilex/src/midicvt_base.c`, but for the purposes of writing out another MIDI file, rather than a text file. The new MIDI file has certain keys or patches remapped between General MIDI (GM) settings and the settings an old MIDI device such as the Yamaha PSS-790.

Conversion from MIDI to text starts with the (new) C function `mftransform()`.

Definition in file `midimapper.cpp`.

9.24.2 Function Documentation

9.24.2.1 static int midimap_chanpressure (int chan, int pressure) [static]

For filtering channel-pressure messages.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pressure</i>	Provides the patch/program number to write.

Returns

Returns the result of the [m2m_chanpressure\(\)](#) call. If the channel is not active, then 0 (false) is returned.

Definition at line 1145 of file midimapper.cpp.

9.24.2.2 void midimap_init ([midipp::midimapper & mm](#))

Hooks the [midipp::midimapper](#) object provided to the global pointer for such objects, to allow the C routines to be able to use the features of the MIDI mapper.

This function calls [midicvt_initfuncs_m2m\(\)](#) to set up the MIDI-to-MIDI callbacks, but then overrides some of them with the C call backs in the rest of this module.

Parameters

<i>mm</i>	Provides the MIDI mapper object to be used.
-----------	---

Definition at line 859 of file midimapper.cpp.

9.24.2.3 static int midimap_noff (int *chan*, int *pitch*, int *vol*) [static]

This static C function uses the singleton [midipp::midimapper](#) object to remap the channel and pitch values, and passes them to the corresponding callback function defined in the [midicvt_m2m.c](#) module.

If the [midipp::midimapper](#) object is not hooked in, then the values are passed unchanged.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>vol</i>	Provides the MIDI note velocity to write.

Returns

Returns the return-value of the corresponding [midicvt_m2m](#) callback function.

Definition at line 946 of file midimapper.cpp.

9.24.2.4 static int midimap_non (int *chan*, int *pitch*, int *vol*) [static]

This static C function uses the singleton [midipp::midimapper](#) object to remap the channel and pitch values, and passes them to the corresponding callback function defined in the [midicvt_m2m.c](#) module.

If the [midipp::midimapper](#) object is not hooked in, then the values are passed unchanged. This then yields the same functionality as running the following C program from this library:

```
$ ./midicvt --m2m in.mid out.mid
```

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>vol</i>	Provides the MIDI note velocity to write.

Returns

Returns the return-value of the corresponding `midicvt_m2m` callback function.

Definition at line 900 of file `midimapper.cpp`.

9.24.2.5 `static int midimap_parameter (int chan, int control, int value) [static]`

For filtering parameter/control messages.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>control</i>	Provides the MIDI controller number to write.
<i>value</i>	Provides the MIDI controller parameter value to write.

Returns

Returns the result of the `m2m_parameter()` call. If the channel is not active, then 0 (false) is returned.

Definition at line 1076 of file `midimapper.cpp`.

9.24.2.6 `static int midimap_patch (int chan, int patch) [static]`

Remaps the channel and the program (patch) number, and passes them to the corresponding callback function defined in the `midicvt_m2m.c` module.

If the `midipp::midimapper` object is not hooked in, then the values are passed unchanged.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>patch</i>	Provides the patch/program number to write.

Returns

Returns the return-value of the corresponding `midicvt_m2m` callback function.

Definition at line 1035 of file `midimapper.cpp`.

9.24.2.7 `static int midimap_pitchbend (int chan, int lsb, int msb) [static]`

For filtering pitchbend messages.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>lsb</i>	Provides the least-significant bits of the MIDI pitch-wheel value to write.
<i>msb</i>	Provides the most-significant bits of the MIDI pitch-wheel value to write.

Returns

Returns the result of the `m2m_pitchbend()` call. If the channel is not active, then 0 (false) is returned.

Definition at line 1113 of file `midimapper.cpp`.

9.24.2.8 static int midimap_pressure (int *chan*, int *pitch*, int *pressure*) [static]

This static C function uses the singleton [midipp::midimapper](#) object to remap the channel and pitch values, and passes them to the corresponding callback function defined in the [midicvt_m2m.c](#) module.

If the [midipp::midimapper](#) object is not hooked in, then the values are passed unchanged.

Parameters

<i>chan</i>	Provides the channel number to write.
<i>pitch</i>	Provides the MIDI note number to write.
<i>pressure</i>	Provides the MIDI polyphonic key pressure value to write.

Returns

Returns the return-value of the corresponding [midicvt_m2m](#) callback function.

Definition at line 992 of file [midimapper.cpp](#).

9.24.2.9 void show_maps (const std::string & *tag*, const midipp::midimapper & *container*)

Writes out the contents of the pitch-map container out to stderr.

We can't write to stdout because that is often redirected to a file.

This implementation is a `for_each` style of looping through the container.

Parameters

<i>tag</i>	Identifies the object in the human-readable output.
<i>container</i>	The stringmap through which iteration is done for showing.

Definition at line 693 of file [midimapper.cpp](#).

9.24.3 Variable Documentation

9.24.3.1 midipp::midimapper* gs_singleton_midimapper = nullptr [static]

Holds the single pointer to the [midipp::midimapper](#) object the caller created.

Since we're integrating with C code, we can't really use more than one object at a time anyway.

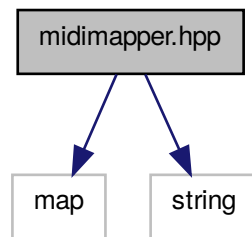
Definition at line 843 of file [midimapper.cpp](#).

9.25 midimapper.hpp File Reference

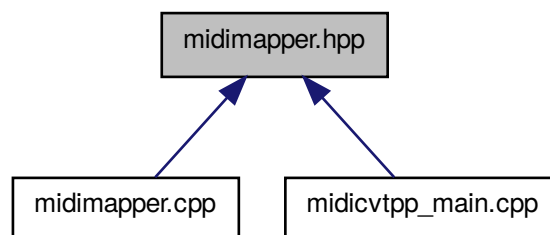
This module provides functions for advanced MIDI/text conversions.

```
#include <map>
#include <string>
```

Include dependency graph for `midimapper.hpp`:



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



Data Structures

- class `midipp::midimapper`

This class provides for some basic remappings to be done to MIDI files, using the old and new facilities of `libmidifilex`.

Functions

- void `show_maps` (const std::string &tag, const `midipp::midimapper` &container)
Writes out the contents of the pitch-map container out to `stderr`.
- void `midimap_init` (`midipp::midimapper` &mm)

Hooks the `midipp::midimapper` object provided to the global pointer for such objects, to allow the C routines to be able to use the features of the MIDI mapper.

9.25.1 Detailed Description

This module provides functions for advanced MIDI/text conversions.

Library:

`libmidipp`

Author

Chris Ahlstrom

Date

2014-04-24

Last Edits:

2014-05-21

Version

\$Revision\$

License:**GNU GPL**

The mapping process works through static functions that reference a global midimapper object.

This object gets its setup from an INI file. This INI file has an unnamed section with the following format:

```
file-style = sectioned
setup-name = midicvtp
record-count = 51
map-type = Drum
gm_hannel = 10
device-channel = 16
```

The "drum" sections are named for the GM note that is to be remapped, and the "patch" sections are name for the GM patch/program number that is to be remapped:

```
[ Drum 35 ]
gm-name = Acoustic Bass Drum
gm-note = 35
dev-name = N/A
dev-note = 35
gm-equiv = Acoustic Base Drum

[ Patch 1 ]
gm-name = Bright Acoustic Piano
gm-patch = 1
dev-name = Jazz Organ 1
dev-patch = 16
gm-equiv = Drawbar Organ
```

The gm-equiv fields are currently just a way to see how good the mapping is. If it matches the gm-name, that mapping is probably pretty good.

Finally, a facility is provided to move channels around. This is needed, for example, when converting drums to channel 10, but the MIDI file already has other music on channel 10. If you don't need this section, don't define it; it saves processing time.

```
[ Channels ]
ch_01 = 1
ch_02 = 2
. . .
ch_16 = 16
```

Definition in file [midimapper.hpp](#).

9.25.2 Function Documentation

9.25.2.1 void midimap_init (midipp::midimapper & *mm*)

Hooks the [midipp::midimapper](#) object provided to the global pointer for such objects, to allow the C routines to be able to use the features of the MIDI mapper.

This function calls [midicvt_initfuncs_m2m\(\)](#) to set up the MIDI-to-MIDI callbacks, but then overrides some of them with the C call backs in the rest of this module.

Parameters

<i>mm</i>	Provides the MIDI mapper object to be used.
-----------	---

Definition at line 859 of file `midimapper.cpp`.

9.25.2.2 void show_maps (const std::string & *tag*, const midipp::midimapper & *container*)

Writes out the contents of the pitch-map container out to stderr.

We can't write to stdout because that is often redirected to a file.

This implementation is a `for_each` style of looping through the container.

Parameters

<i>tag</i>	Identifies the object in the human-readable output.
<i>container</i>	The stringmap through which iteration is done for showing.

Definition at line 693 of file `midimapper.cpp`.

9.26 midipp_functions.dox File Reference

This document describes the functions and parameters of the `libmidipp` module.

9.26.1 Detailed Description

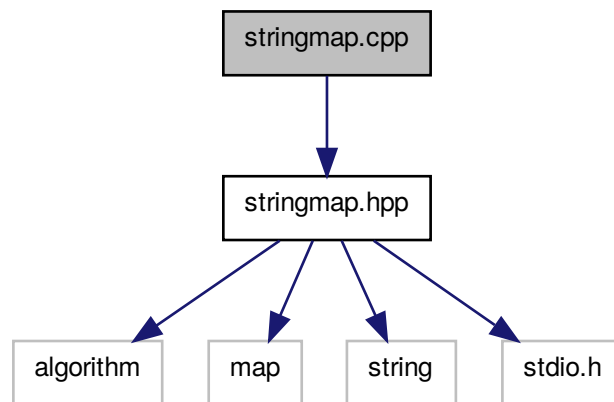
This document describes the functions and parameters of the `libmidipp` module.

Definition in file [midipp_functions.dox](#).

9.27 stringmap.cpp File Reference

```
#include <stringmap.hpp>
```

Include dependency graph for stringmap.cpp:



Functions

- bool **midipp::iequal** (const std::string &s1, const std::string &s2)
A simple case-insensitive comparison for simple string equality.
- void **show** (const std::string &tag, const std::string &s)
Writes out the contents of the string to standard output, in a stylized format.

9.27.1 Detailed Description

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-22

Last Edits:

2014-05-18

Version

\$Revision\$

License:

\$XPC_SUITE_GPL_LICENSE\$

This module defines an std::map that uses only std::string as a key value, for easy lookup.

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

9.27.2 Function Documentation

9.27.2.1 void show (const std::string & tag, const std::string & s)

Writes out the contents of the string to standard output, in a stylized format.

Also see the show_pair() and [show\(\)](#) functions in the [stringmap.hpp](#) module.

Parameters

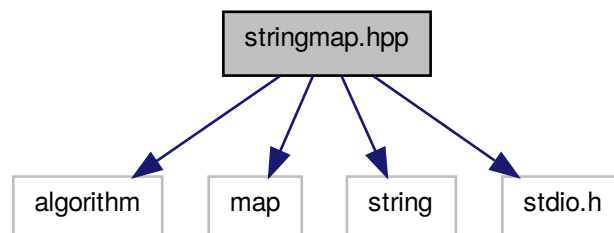
<i>tag</i>	A name for the string, to provide context for the human reader.
<i>s</i>	The string to be shown.

Definition at line 83 of file stringmap.cpp.

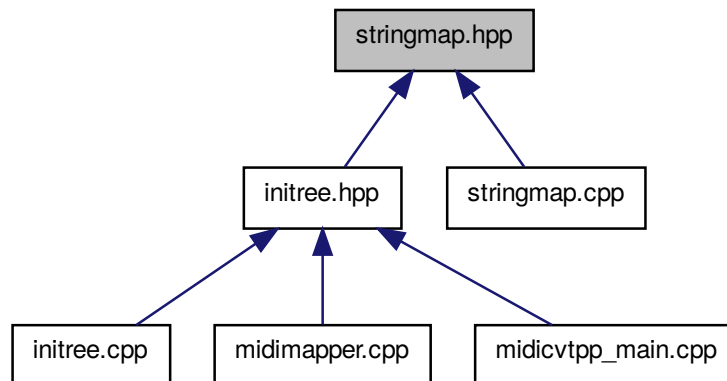
9.28 stringmap.hpp File Reference

```
#include <algorithm>
#include <map>
#include <string>
#include <stdio.h>
```

Include dependency graph for stringmap.hpp:



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



Data Structures

- class `midipp::stringmap< VALUETYPE >`
Provides an `std::map` wrapper geared towards using `std::string` as a key.

Functions

- void `show` (const `std::string` &tag, const `std::string` &s)
Writes out the contents of the string to standard output, in a stylized format.
- bool `midipp::iequal` (const `std::string` &s1, const `std::string` &s2)
A simple case-insensitive comparison for simple string equality.
- template<class VALUETYPE >
 void `midipp::show_pair` (const typename `stringmap< VALUETYPE >::pair` &p)
Writes out the contents of an string/VALUETYPE pair.
- template<class VALUETYPE >
 void `midipp::show` (const `std::string` &tag, const `stringmap< VALUETYPE >` &container)
Writes out the contents of the property container.

9.28.1 Detailed Description

Library:

libmidipp

Author

Chris Ahlstrom

Date

2014-04-22

Last Edits:

2014-05-21

Version

\$Revision\$

License:

\$MIDIPP_SUITE_GPL_LICENSE\$

This module defines an `std::map` template class using `std::string` as a key value.

Definition in file [stringmap.hpp](#).

9.28.2 Function Documentation

9.28.2.1 void show (const std::string & tag, const std::string & s)

Writes out the contents of the string to standard output, in a stylized format.

Also see the `show_pair()` and [show\(\)](#) functions in the [stringmap.hpp](#) module.

Parameters

<i>tag</i>	A name for the string, to provide context for the human reader.
<i>s</i>	The string to be shown.

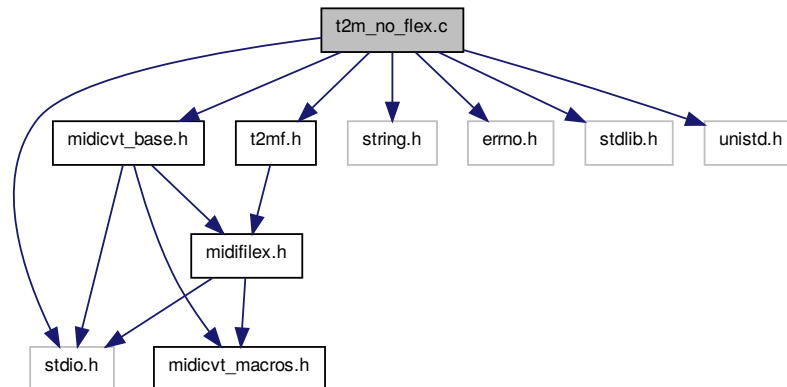
Definition at line 83 of file `stringmap.cpp`.

9.29 t2m_no_flex.c File Reference

This file is to be used instead of the dynamically generated [t2mflex.c](#) if there is no flex program installed or if the installed flex is old [e.g 2.5.35 as opposed to 2.5.39 (!)].

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <midicvt_base.h>
#include <t2mf.h>
#include <unistd.h>
```

Include dependency graph for t2m_no_flex.c:



Functions

- void [yyrestart](#) (FILE *input_file)
Immediately switch to a different input stream.
- void [yy_switch_to_buffer](#) (YY_BUFFER_STATE new_buffer)
Switch to a different input buffer.
- YY_BUFFER_STATE [yy_create_buffer](#) (FILE *file, int size)
Allocate and initialize an input buffer state.
- void [yy_delete_buffer](#) (YY_BUFFER_STATE b)
Destroy the buffer.
- void [yy_flush_buffer](#) (YY_BUFFER_STATE b)
Discard all buffered characters.
- void [yypush_buffer_state](#) (YY_BUFFER_STATE new_buffer)
Pushes the new state onto the stack.
- void [yypop_buffer_state](#) (void)
Removes and deletes the top of the stack, if present.
- YY_BUFFER_STATE [yy_scan_buffer](#) (char *base, yy_size_t size)
Setup the input buffer state to scan directly from a user-specified character buffer.
- YY_BUFFER_STATE [yy_scan_string](#) (yyconst char *yystr)
Setup the input buffer state to scan a string.
- YY_BUFFER_STATE [yy_scan_bytes](#) (yyconst char *yybytes, yy_size_t yybytes_len)
Setup the input buffer state to scan the given bytes.
- FILE * [yyget_in](#) (void)
Get the input stream.
- void [yyset_in](#) (FILE *in_str)
Set the input stream.
- FILE * [yyget_out](#) (void)
Get the output stream.
- yy_size_t [yyget_leng](#) (void)
Get the length of the current token.
- char * [yyget_text](#) (void)
Get the current token.

- int [yyget_lineno](#) (void)
Get the current line number.
- void [yyset_lineno](#) (int line_number)
Set the current line number.

Variables

- static size_t [yy_buffer_stack_top](#) = 0
index of top of stack.
- static size_t [yy_buffer_stack_max](#) = 0
capacity of stack.
- static YY_BUFFER_STATE * [yy_buffer_stack](#) = 0
Stack as an array.
- int [do_hex](#) = 0
Created and declared by flex.
- long [yyval](#)
Use externs from an include file! The flex-generated code is a bit messy.
- [YY_DECL](#)
The main scanner function which does all the work.

9.29.1 Detailed Description

This file is to be used instead of the dynamically generated [t2mflex.c](#) if there is no flex program installed or if the installed flex is old [e.g 2.5.35 as opposed to 2.5.39 (!)].

Library:

midicvt application

Author

Chris Ahlstrom and many other authors

Date

2014-04-09

Last Edits:

2014-06-21

Version

\$Revision\$

License:

GNU GPL

By using this file, we don't have to use flex again to generate [t2mflex.c](#) from t2mf.fl and put up with warning messages.

Of course, you will have to manually reconstruct the current file if you decided to change the parsing dictated by the t2mf.fl file, or edit Makefile.am to re-enable the use of flex..

Definition in file [t2m_no_flex.c](#).

9.29.2 Function Documentation

9.29.2.1 YY_BUFFER_STATE yy_create_buffer (FILE * *file*, int *size*)

Allocate and initialize an input buffer state.

Parameters

<i>file</i>	A readable stream.
<i>size</i>	The character buffer size in bytes. When in doubt, use <code>YY_BUF_SIZE</code> .

Returns

the allocated buffer state.

Definition at line 1971 of file `t2m_no_flex.c`.

9.29.2.2 void yy_delete_buffer (YY_BUFFER_STATE *b*)

Destroy the buffer.

Parameters

<i>b</i>	a buffer created with yy_create_buffer()
----------	--

Definition at line 1999 of file `t2m_no_flex.c`.

9.29.2.3 void yy_flush_buffer (YY_BUFFER_STATE *b*)

Discard all buffered characters.

On the next scan, `YY_INPUT` will be called.

Parameters

<i>b</i>	the buffer state to be flushed, usually <code>YY_CURRENT_BUFFER</code> .
----------	--

Definition at line 2046 of file `t2m_no_flex.c`.

9.29.2.4 YY_BUFFER_STATE yy_scan_buffer (char * *base*, yy_size_t *size*)

Setup the input buffer state to scan directly from a user-specified character buffer.

Parameters

<i>base</i>	the character buffer
<i>size</i>	the size in bytes of the character buffer

Returns

the newly allocated buffer state object.

Definition at line 2173 of file `t2m_no_flex.c`.

9.29.2.5 YY_BUFFER_STATE yy_scan_bytes (yyconst char * *yybytes*, yy_size_t *yybytes_len*)

Setup the input buffer state to scan the given bytes.

The next call to `yylex()` will scan from a *copy* of *bytes*.

Parameters

<i>yybytes</i>	the byte buffer to scan
<i>_yybytes_len</i>	the number of bytes in the buffer pointed to by <i>bytes</i> .

Returns

the newly allocated buffer state object.

Definition at line 2223 of file t2m_no_flex.c.

9.29.2.6 YY_BUFFER_STATE yy_scan_string (*yyconst* char * *yystr*)

Setup the input buffer state to scan a string.

The next call to `yylex()` will scan from a *copy* of *str*.

Parameters

<i>yystr</i>	a NUL-terminated string to scan
--------------	---------------------------------

Returns

the newly allocated buffer state object.

Note

If you want to scan bytes that may contain NUL values, then use [yy_scan_bytes\(\)](#) instead.

Definition at line 2210 of file t2m_no_flex.c.

9.29.2.7 void yy_switch_to_buffer (YY_BUFFER_STATE *new_buffer*)

Switch to a different input buffer.

Parameters

<i>new_buffer</i>	The new input buffer.
-------------------	-----------------------

Definition at line 1926 of file t2m_no_flex.c.

9.29.2.8 void yypop_buffer_state (void)

Removes and deletes the top of the stack, if present.

The next element becomes the new top.

Definition at line 2105 of file t2m_no_flex.c.

9.29.2.9 void yypush_buffer_state (YY_BUFFER_STATE *new_buffer*)

Pushes the new state onto the stack.

The new state becomes the current state. This function will allocate the stack if necessary.

Parameters

<i>new_buffer</i>	The new state.
-------------------	----------------

Definition at line 2075 of file t2m_no_flex.c.

9.29.2.10 void yyrestart (FILE * *input_file*)

Immediately switch to a different input stream.

Parameters

<i>input_file</i>	A readable stream.
-------------------	--------------------

Note

This function does not reset the start condition to `INITIAL`.

Definition at line 1907 of file t2m_no_flex.c.

9.29.2.11 void yyset_in (FILE * *in_str*)

Set the input stream.

This does not discard the current input buffer.

Parameters

<i>in_str</i>	A readable stream.
---------------	--------------------

See also

[yy_switch_to_buffer](#)

Definition at line 2340 of file t2m_no_flex.c.

9.29.2.12 void yyset_lineno (int *line_number*)

Set the current line number.

Parameters

<i>line_number</i>	
--------------------	--

Definition at line 2328 of file t2m_no_flex.c.

9.29.3 Variable Documentation

9.29.3.1 int do_hex = 0

Created and declared by flex.

We added these declarations here to avoid warnings and errors. Find these variables in the flex-generated file [t2mflex.c](#).

Definition at line 989 of file t2m_no_flex.c.

9.29.3.2 YY_BUFFER_STATE* yy_buffer_stack = 0 [static]

Stack as an array.

Definition at line 309 of file t2m_no_flex.c.

9.29.3.3 `size_t yy_buffer_stack_max = 0` `[static]`

capacity of stack.

Definition at line 308 of file `t2m_no_flex.c`.

9.29.3.4 `size_t yy_buffer_stack_top = 0` `[static]`

index of top of stack.

Definition at line 307 of file `t2m_no_flex.c`.

9.29.3.5 `long yyval`

Use externs from an include file! The flex-generated code is a bit messy.

The generated `t2mflex.c` file defines these variables, we declare them in `t2fm.h`, and have to define `yyval` here. Bleh.

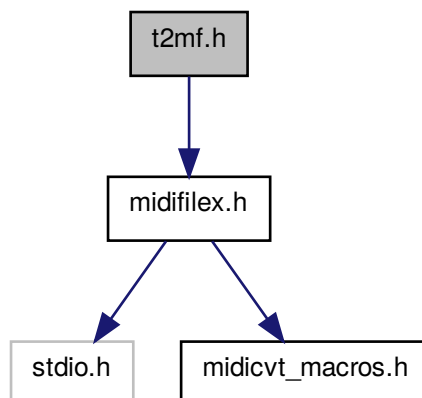
Definition at line 80 of file `midicvt_globals.c`.

9.30 t2mf.h File Reference

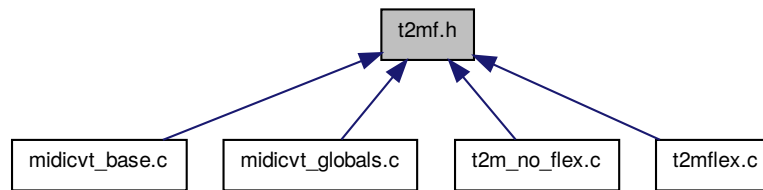
This module provides global variables for the text-to-MIDI conversion portion of `libmidiflex`.

```
#include <midiflex.h>
```

Include dependency graph for `t2mf.h`:



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



Macros

- `#define YY_TYPEDEF_YY_SIZE_T`
Good old CentOS's version of flex declared `yylen` as an integer.
- `#define MTHD 256`
Macros galore.

Variables

- `int do_hex`
Created and declared by flex.
- `long yyval`
Use externs from an include file! The flex-generated code is a bit messy.

9.30.1 Detailed Description

This module provides global variables for the text-to-MIDI conversion portion of libmidifilex.

Library:

libmidifilex

Author

Chris Ahlstrom and many other authors

Date

2014-04-09

Last Edits:

2014-04-20

Version

\$Revision\$

License:

GNU GPL

Definition in file [t2mf.h](#).

9.30.2 Macro Definition Documentation

9.30.2.1 #define YY_TYPEDEF_YY_SIZE_T

Good old CentOS's version of flex declared `yylen` as an integer.

Debian unstable's flex declares it as `yy_size_t`. Unfortunately, there's no header-file that defines it! We found one typedef via a web search, but the actual type is a generated typedef in the generated C file, so we have to match that, and hold off the compiler from complaining about a duplicate typedef.

Definition at line 79 of file `t2mf.h`.

9.30.3 Variable Documentation

9.30.3.1 int do_hex

Created and declared by flex.

We added these declarations here to avoid warnings and errors. Find these variables in the flex-generated file [t2mflex.c](#).

Definition at line 989 of file `t2m_no_flex.c`.

9.30.3.2 long yyval

Use externs from an include file! The flex-generated code is a bit messy.

The generated [t2mflex.c](#) file defines these variables, we declare them in `t2fm.h`, and have to define `yyval` here. Bleh.

Definition at line 80 of file `midicvt_globals.c`.

9.31 VERSION.h File Reference

This module provides simpler version information for the libmidifilex library.

9.31.1 Detailed Description

This module provides simpler version information for the libmidifilex library.

Library:

libmidifilex

Author

Chris Ahlstrom

Date

2007-02-28

Last Edits:

2014-06-02

Version

\$Revision\$

License:

GNU GPL

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