MIDIcvt Developer's Reference Manual

0.4.0.0

Generated by Doxygen 1.8.11

Contents

1 midicvt

Author(s) Chris Ahlstrom 2015-08-14

1.1 Introduction

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.

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.

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 midicvtpp 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_filex* 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.

3 The libmidipp Library

3.1 Introduction to the MIDIcvt 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.

4 Licenses, MIDIcvt Projects.

Author(s) Chris Ahlstrom 2015-08-14

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**.

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**.

4.4 XPC Documentation License

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

4.5 XPC Affero License

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

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*).

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.

6 Deprecated List

Global midipp::write_simple_drum_file (const std::string &filespec, const csvarray &csv)
The input fields of a Drum map grabbed from a spreadsheet are in the following order:

7 Data Structure Index

7.1 Data Structures

Here are the data structures with brief descriptions:

8 File Index

8.1 File List

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

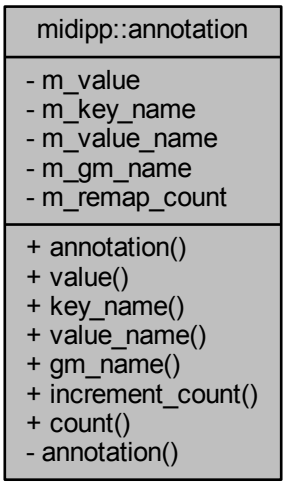
9 Data Structure Documentation

9.1 midipp::annotation Class Reference

This class is meant to extend the map of values with additional data that can be written out to summarize some information about the MIDI remapping that was done.

`#include <midimapper.hpp>`

Collaboration diagram for midipp::annotation:



Public Member Functions

- annotation (int value, const std::string &keyname, const std::string &valuename, const std::string &gmname)

  Principal constructor for the annotation class.
- int value () const

  'Getter' function for member m_value
- const std::string & key_name () const

  'Getter' function for member m_key_name
- const std::string & value_name () const

'Getter' function for member m_value_name

- const std::string & gm_name () const

  'Getter' function for member m_gm_name

- void increment_count ()

  'Setter' function for member m_remap_count

- int count () const

  'Getter' function for member m_remap_count

Private Attributes

- const int m_value

  The integer value to which the incoming (key) value is to be mapped.

- const std::string m_key_name

  The name of the drum note or patch represented by the key value.

- const std::string m_value_name

  The name of the drum note or patch represented by the integer value.

- const std::string m_gm_name

  The name of the GM drum note or patch that is replacing the device's drum note of patch.

- int m_remap_count

  The number of times this particular mapping was performed in the MIDI remapping operation.

9.1.1 Detailed Description

This class is meant to extend the map of values with additional data that can be written out to summarize some information about the MIDI remapping that was done.

Instead of just the integer value to use, this class holds the names of the items on both ends of the mapping, plus a usage count. We also added the "GM equivalent" name to this class as well.

Definition at line 106 of file midimapper.hpp.

9.1.2 Constructor & Destructor Documentation

9.1.2.1 midipp::annotation::annotation (int *value*, const std::string & *keyname*, const std::string & *valuename*, const std::string & *gmname*)

Principal constructor for the annotation class.

Parameters

value	The integer value to which the incoming (key) value is to be mapped.
keyname	The name of the drum note or patch represented by the key value.
valuename	The name of the drum note or patch represented by the integer value.

Definition at line 93 of file midimapper.cpp.

9.1.3 Field Documentation

9.1.3.1 const std::string midipp::annotation::m_gm_name [private]

The name of the GM drum note or patch that is replacing the device's drum note of patch.

Sometimes there is no exact replacement, so it is good to know what GM sound is replacing the device's sound.

Definition at line 138 of file midimapper.hpp.

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

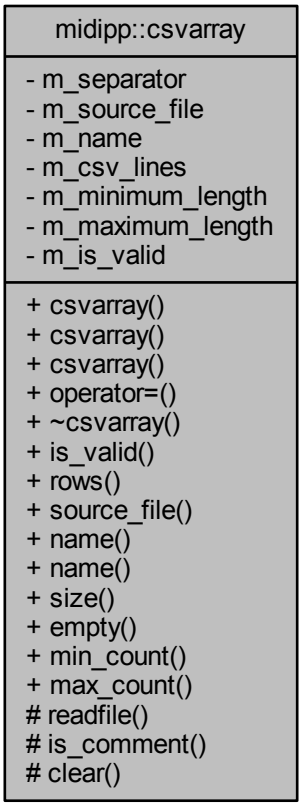
- midimapper.hpp
- midimapper.cpp

9.2 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:



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 ()

  Destructor
- 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

  Accessor *m_csv_lines.size()*
- bool empty () const

  Accessor *m_csv_lines.empty()*
- 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.

9.2.1 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.

9.2.2 Member Typedef Documentation

9.2.2.1 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.

9.2.3 Constructor & Destructor Documentation

9.2.3.1 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.

9.2.3.2 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

name	Provides the name of the csvarray object.
filespec	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.
separator	Provides the separator character that demarcates fields. The default value of this parameter is the comma.

Definition at line 78 of file csvarray.cpp.

9.2.3.3 midipp::csvarray::csvarray (const csvarray & *source*)

Copy constructor for csvarray.

Parameters

source	Provides the original object to be copied.

Definition at line 102 of file csvarray.cpp.

9.2.3.4 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.

9.2.4 Member Function Documentation

9.2.4.1 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.

9.2.4.2 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.

9.2.4.3 csvarray & midipp::csvarray::operator= (const csvarray & *source*)

Principal assignment operator for csvarray.

Parameters

source	Provides the original object to be assigned.

Definition at line 122 of file csvarray.cpp.

9.2.4.4 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

filespec	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.

9.2.4.5 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.

9.2.5 Field Documentation

9.2.5.1 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.

9.2.5.2 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.

9.2.5.3 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:

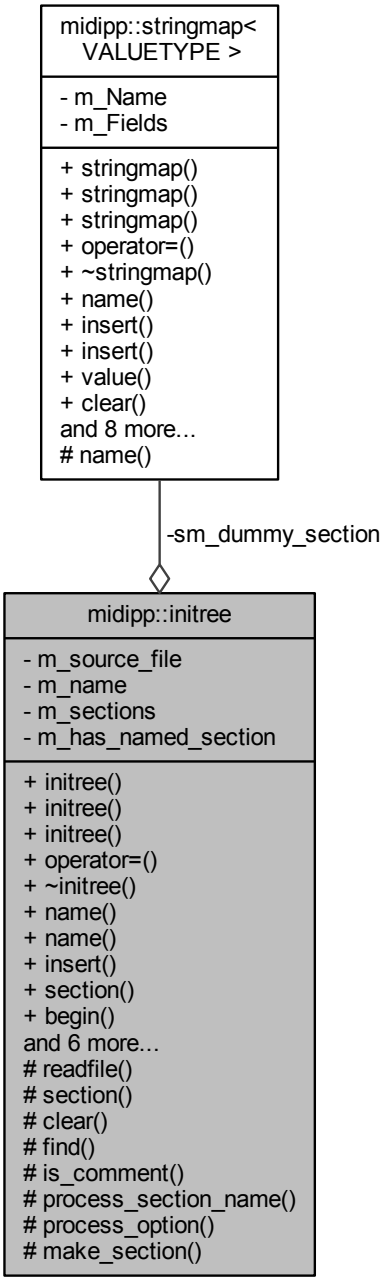
- csvarray.hpp
- csvarray.cpp

9.3 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 ()

  Destructor

- 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 &sectionname, const Section &section)

  Allows the insertion of an stringmap object into the container.

- const Section & section (const std::string &sectionname) const

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

- const_iterator begin () const

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

- const_iterator end () const

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

- iterator begin ()

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

- iterator end ()

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

- size_t size () const

  Accessor *m_sections.size()*

- bool empty () const

  Accessor *m_sections.empty()*

- const_iterator find (const std::string &sectionname) const

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

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 &sectionname)

  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 &sectionname)

  Accessor *m_sections.find() This function makes this class look more like an STL container.*
- 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 &sectionname)

  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 &sectionname)

  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.

9.3.1 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.

9.3.2 Member Typedef Documentation

9.3.2.1 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.

9.3.2.2 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.

9.3.2.3 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.

9.3.2.4 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.

9.3.3 Constructor & Destructor Documentation

9.3.3.1 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.

9.3.3.2 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

name	Provides the name of the initree object.
filespec	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.

9.3.3.3 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.

9.3.3.4 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.

9.3.4 Member Function Documentation

9.3.4.1 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.

9.3.4.2 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.

9.3.4.3 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.

9.3.4.4 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.

9.3.4.5 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.

9.3.4.6 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.

9.3.4.7 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.

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

Allows the insertion of an stringmap object into the container.

Parameters

section	The string that is to serve as the lookup value for the inserted object.
value	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.

9.3.4.9 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

c	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.

9.3.4.10 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

sectionname	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.

9.3.4.11 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.

9.3.4.12 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 quites if spaces and special
characters are part of the string.

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

Parameters

s	Provides the current line of text, which starts with a letter.
p	Provides the position of the first letter.
currentsection	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 created 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.

9.3.4.13 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

s	Provides the current line of text, which starts with '['.
p	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.

9.3.4.14 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

filespec	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.

9.3.4.15 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

sectionname	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.

9.3.4.16 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

sectionname	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.

9.3.4.17 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.

9.3.5 Field Documentation

9.3.5.1 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.

9.3.5.2 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:

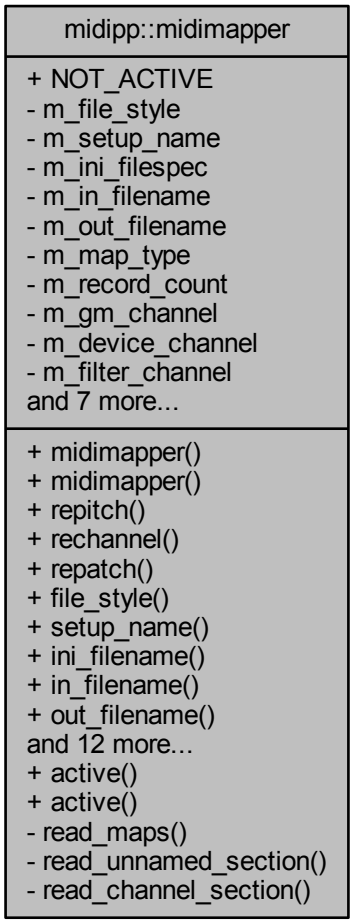
- initree.hpp
- initree.cpp

9.4 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, const std::string &infile="", const std::string &outfile="")

  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 & ini_filename () const

  'Getter' function for member m_ini_filespec
- const std::string & in_filename () const

  'Getter' function for member m_in_filename
- const std::string & out_filename () const

  'Getter' function for member m_out_filename
- 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 midimap & drum_map () const

  'Getter' function for member m_drum_map Returns a reference to the note map.
- const midimap & 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 Types

- typedef std::map< int, annotation > midimap

  Provides the type of the map between one set of values and another set of values.

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_ini_filespec

  Saves the name of the INI file, which can be useful in the show_maps() function.
- std::string m_in_filename

  Saves the name of the input MIDI file, for information only.
- std::string m_out_filename

  Saves the name of the output MIDI file, for information only.
- 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.
- midimap m_drum_map

  Provides the mapping between pitches.
- midimap 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.

Friends

- void show_maps (const std::string &tag, const midipp::midimapper &container, bool full_output)

  Writes out the contents of the pitch-map container out to stderr.

9.4.1 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 midicvtpp 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 234 of file midimapper.hpp.

9.4.2 Constructor & Destructor Documentation

9.4.2.1 midipp::midimapper::midimapper ( )

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

This object can be used for testing.

Definition at line 114 of file midimapper.cpp.

9.4.2.2 midipp::midimapper::midimapper (const std::string & *name*, const std::string & *filespec* = " ", bool *reverse_it* = **false**, int *filter_channel* = **NOT_ACTIVE**, bool *reject_it* = **false**, const std::string & *infile* = " ", const std::string & *outfile* = " ")

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

This object can be used for testing.

Parameters

name	Provides a handy name to refer to when reading the output of this object.
filespec	Provides the full file-path specification of an INI-style file to be read. See the example files in the `tests/inifiles` directory.
reverse_it	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.
filter_channel	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.
reject_it	If the *filter_channel* 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.
infile	The name of the input MIDI file, for informational purposes only.
outile	The name of the outut MIDI file, for informational purposes only.

Definition at line 178 of file midimapper.cpp.

9.4.3 Member Function Documentation

9.4.3.1 static bool midipp::midimapper::active (int *value*) [inline],[static]

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

Parameters

value	The integer value to be checked.

Returns

Returns true if the value is not NOT_ACTIVE.

Definition at line 484 of file midimapper.hpp.

9.4.3.2 static bool midipp::midimapper::active (int *v1,* int *v2*) [inline],[static]

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

Parameters

v1	The first integer value to be checked.
v2	The second integer value to be checked.

Returns

Returns true if both of the values are not NOT_ACTIVE.

Definition at line 502 of file midimapper.hpp.

9.4.3.3 const midimap& 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 640 of file midimapper.hpp.

9.4.3.4 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

it	Provides the tree of INI name/value pairs to analyze.

Returns

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

Definition at line 565 of file midimapper.cpp.

9.4.3.5 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, gm_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 gm_device_note value that best matches the device's original sound. 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.

std::map::insert<> returns a pair of values: an iterator into the map, and a boolean value for success/failure. If the insertion fails, the pair is already in the map. We tell the user about this, but do not treat it as a fatal error.

Parameters

filename	Provides the full path specification of the file to be read.

Returns

Returns true if the operation succeeded.

Definition at line 254 of file midimapper.cpp.

9.4.3.6 bool midipp::midimapper::read_unnamed_section (const initree & *it*) [private]

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

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

Change Note ca 2016-04-17 We no longer error and break if an unnamed-section value tag is not present. It should merely disable the functionality of that tag, not the whole remapping file. Note that this functionality was never part of the unit test, and we ought to fix that lack at some point.

Parameters

it	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 445 of file midimapper.cpp.

9.4.3.7 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

channel	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 707 of file midimapper.cpp.

9.4.3.8 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

program	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 760 of file midimapper.cpp.

9.4.3.9 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

channel	Provides the channel for the input note. Only if the channel matches the one channel to be remapped, will repitching occur.
input	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 653 of file midimapper.cpp.

9.4.4 Friends And Related Function Documentation

9.4.4.1 void show_maps (const std::string & *tag,* const **midipp::midimapper** & *container,* bool *full_output*) [friend]

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 each container.

Parameters

tag	Identifies the object in the human-readable output.
container	The midimapper through which iteration is done for showing.
full_output	Defaults to true, which means to show everything. If false, map entries not used in the conversion are not shown.

Definition at line 790 of file midimapper.cpp.

9.4.5 Field Documentation

9.4.5.1 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. There's no need for channel names with this one.

Definition at line 448 of file midimapper.hpp.

9.4.5.2 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 369 of file midimapper.hpp.

9.4.5.3 **midimap** 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 427 of file midimapper.hpp.

9.4.5.4 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 285 of file midimapper.hpp.

9.4.5.5 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 388 of file midimapper.hpp.

9.4.5.6 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 359 of file midimapper.hpp.

9.4.5.7 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 415 of file midimapper.hpp.

9.4.5.8 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 333 of file midimapper.hpp.

9.4.5.9 midimap 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 440 of file midimapper.hpp.

9.4.5.10 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 346 of file midimapper.hpp.

9.4.5.11 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 296 of file midimapper.hpp.

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

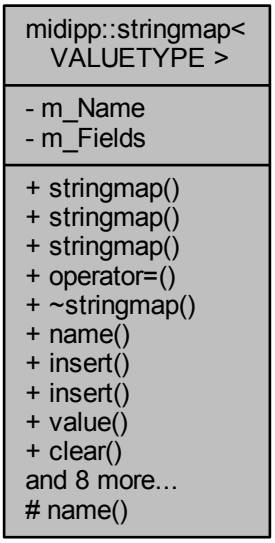
- midimapper.hpp
- midimapper.cpp

9.5 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 >:



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)

Principal Assignment Operator

- virtual ~stringmap ()

  Destructor

- 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)

  Manufacturers 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 ()

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

- const_iterator begin () const

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

- iterator end ()

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

- const_iterator end () const

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

- size_t size () const

  ***Accessor** m_Fields.size()*

- bool empty () const

  ***Accessor** m_Fields.empty()*

- iterator find (const std::string &key)

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

- const_iterator find (const std::string &key) const

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

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.

9.5.1 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.

9.5.2 Member Typedef Documentation

9.5.2.1 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.

9.5.3 Constructor & Destructor Documentation

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

Copy constructor Copies the Container.

Parameters

source	The stringmap to be copied.

Definition at line 162 of file stringmap.hpp.

9.5.3.2 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.

9.5.4 Member Function Documentation

9.5.4.1 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.

9.5.4.2 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.

9.5.4.3 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.

9.5.4.4 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.

9.5.4.5 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.

9.5.4.6 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.

9.5.4.7 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.

9.5.4.8 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

key	The string that is to serve as the lookup value for the inserted object.
value	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.

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

Manufacturers 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

value	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.

9.5.4.10 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

source	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.

9.5.4.11 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.

9.5.4.12 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

key	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.

9.5.5 Field Documentation

9.5.5.1 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:

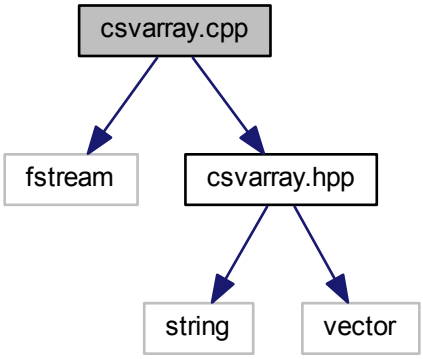
- stringmap.hpp

10 File Documentation

10.1 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.

10.1.1 Detailed Description

Primitive, but useful, CSV-file input parser.

Library libmidipp

Author(s) 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.

10.1.2 Macro Definition Documentation

10.1.2.1 #define TOKEN_SPACES " \t\0"

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.

10.1.3 Function Documentation

10.1.3.1 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

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

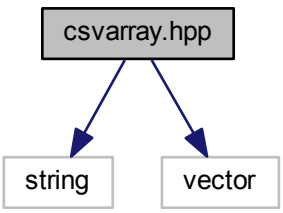
Definition at line 240 of file csvarray.cpp.

10.2 csvarray.hpp File Reference

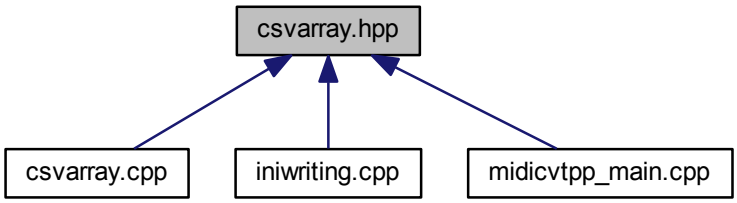
Library libmidipp

```
#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.

10.2.1 Detailed Description

Library libmidipp

Author(s) 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.

10.2.2 Function Documentation

10.2.2.1 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

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

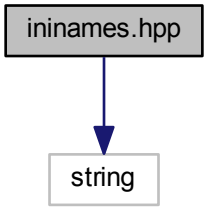
Definition at line 240 of file csvarray.cpp.

10.3 ininames.hpp File Reference

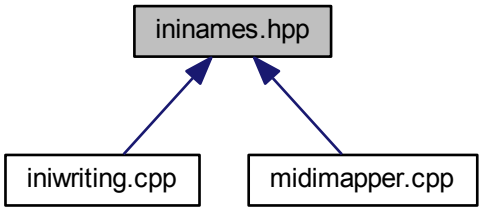
Provides some free functions to write INI files, given a midipp::csvarray object.

```
#include <string>
```

Include dependency graph for ininames.hpp:



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



Enumerations

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.

10.3.1 Detailed Description

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

Library libmidipp

Author(s) 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.

10.3.2 Enumeration Type Documentation

10.3.2.1 enum midipp::gm_drum_field_index_t

Enumerator

DRUM_INDEX_GM_NAME Provides mnemonics for the expected indices into the vector of fields that occurs on each line of our Drum-map files. Provides the index needed to access the column containing the General MIDI drum name for the given MIDI note. Note that this name does not have to correspond to General MIDI; it can correspond to another consumer-level MIDI device. But the most common case will be mapping between one device and Genera MIDI.

DRUM_INDEX_GM_NOTE Provides the index needed to access the column containing the MIDI note value for the given General MIDI drum name in the previous column. This note value is the input value of the note that we want to remap.

DRUM_INDEX_DEV_NAME Provides the index needed to access the column containing the consumer-level device's name for the drum that we want to map to General MIDI or another device.

DRUM_INDEX_DEV_NOTE Provides the index needed to access the column containing the General MIDI (or other device) note number to which we want to remap. This note value is the output value of the note, given the input value at DRUM_INDEX_GM_NOTE.

DRUM_INDEX_GM_EQUIV This item can be optionally placed in the fifth column of the CSV to indicate the GM note name to which the device is being mapped.

Definition at line 82 of file ininames.hpp.

10.3.2.2 enum midipp::gm_ini_section_t

Provides a selection switch for the supported INI section types.

Currently that includes "Drum" and "Patch" sections.

Enumerator

INI_SECTION_DRUM Marks a "[Drum]" section.

INI_SECTION_PATCH Marks a "[Patch]" section.

INI_SECTION_CHANNEL Marks a "[Channel]" section.

INI_SECTION_UNKNOWN Marks an unknown section name.

Definition at line 32 of file ininames.hpp.

10.3.2.3 enum midipp::gm_patch_field_index_t

Enumerator

PATCH_INDEX_GM_NAME Provides mnemonics for the expected indices into the vector of fields that occurs on each line of our Patch-map files. Provides the index needed to access the column containing the General MIDI drum name for the given MIDI note. Note that this name does not have to correspond to General MIDI; it can correspond to another consumer-level MIDI device. But the most common case will be mapping between one device and Genera MIDI.

PATCH_INDEX_GM_PATCH Provides the index needed to access the column containing the MIDI note value for the given General MIDI drum name in the previous column. This note value is the input value of the note that we want to remap.

PATCH_INDEX_DEV_NAME Provides the index needed to access the column containing the consumer-level device's name for the drum that we want to map to General MIDI or another device.

PATCH_INDEX_DEV_PATCH Provides the index needed to access the column containing the General M↩ IDI (or other device) note number to which we want to remap. This note value is the output value of the note, given the input value at PATCH_INDEX_GM_PATCH.

PATCH_INDEX_GM_EQUIV This item can be optionally placed in the fifth column of the CSV to show the GM patch name to which the device is being mapped.

Definition at line 168 of file ininames.hpp.

10.3.3 Variable Documentation

10.3.3.1 const std::string midipp::DRUM_SECTION = "Drum"

Provides the prefix for the initial part of the drum-section name.

These names are of the form "Drum nnn" where nnn is the MIDI note number, ranging from 0 to 127.

Definition at line 47 of file ininames.hpp.

10.3.3.2 const std::string midipp::GM_INI_REVERSE = "reverse"

Provides a consistent string for the reverse attribute.

This is a boolean attribute.

Definition at line 281 of file ininames.hpp.

10.3.3.3 const std::string midipp::GM_INI_TESTING = "testing"

Provides a consistent string for the testing attribute.

This is a boolean attribute.

Definition at line 288 of file ininames.hpp.

10.3.3.4 const std::string midipp::PATCH_SECTION = "Patch"

Provides the prefix for the initial part of the patch-section name.

These names are of the form "Patch nnn" where nnn is the MIDI program number, ranging from 0 to 127.

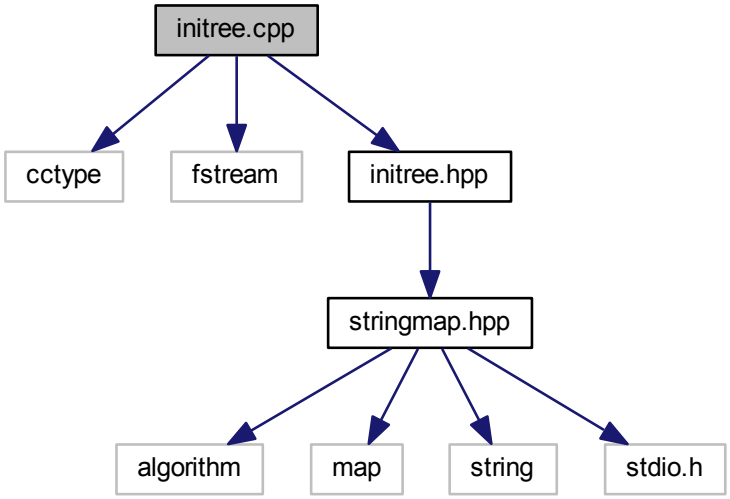
Definition at line 133 of file ininames.hpp.

10.4 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.

10.4.1 Detailed Description

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

Library libmidipp

Author(s) 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.* `optioname = 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.*

10.4.2 Function Documentation

10.4.2.1 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

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

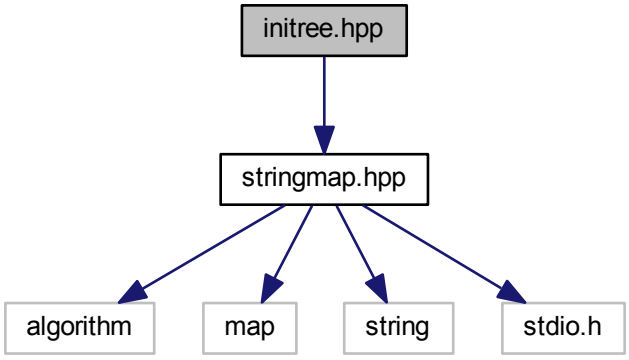
Definition at line 533 of file initree.cpp.

10.5 initree.hpp File Reference

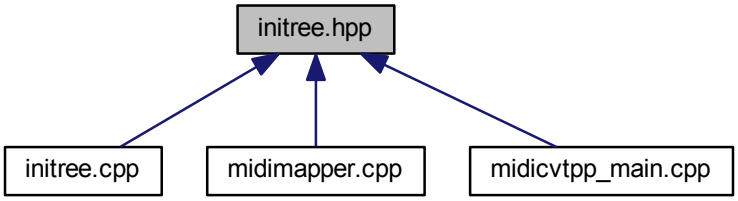
Provides a "tree" that represents an INI file with named and unnamed sections, and name/value pairs.

```
#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.

10.5.1 Detailed Description

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

Library libmidipp

Author(s) 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.

10.5.2 Function Documentation

10.5.2.1 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

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

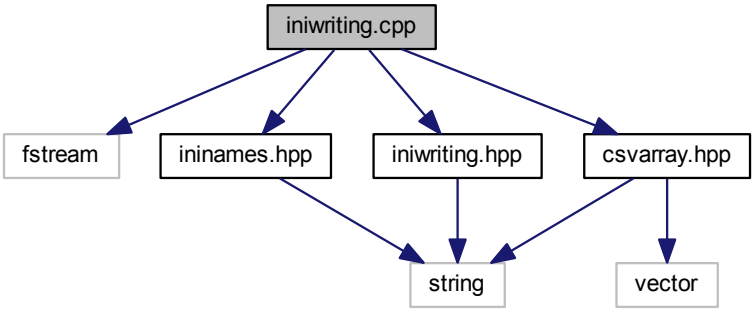
Definition at line 533 of file initree.cpp.

10.6 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 writeheader)
  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.

10.6.1 Detailed Description

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

Library libmidipp

Author(s) Chris Ahlstrom

Date 2014-04-23

Last Edits 2016-04-23

Version

$Revision$ **License** $XPC_SUITE_GPL_LICENSE$

Provides a way to create INI files from the contents of a midipp::csvarray object.

10.6.2 Function Documentation

10.6.2.1 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.

Parameters

filespec	Provides the full path to the CSV file to be processed.
csv	Provides the csvarray object to be dumped as a sectioned INI file.
writefooter	Provides an optional parameter, defaulting to true, to determine if the footer at the bottom of the file should be written. This value will be set to false if writing both drums and patch to the file.

Returns

Returns 'true' if all operations succeeded.

Definition at line 190 of file iniwriting.cpp.

10.6.2.2 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.

Parameters

filespec	Provides the full path to the CSV file to be processed.
csv	Provides the csvarray object to be dumped as a sectioned INI file.
writeheader	Provides an optional parameter, defaulting to true, to determine if the header at the top of the file should be written. This value will be set to false if writing both drums and patch to the file.

Returns

Returns 'true' if all operations succeeded.

Definition at line 278 of file iniwriting.cpp.

10.6.2.3 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.

Deprecated The input fields of a Drum map grabbed from a spreadsheet are in the following order:

```
-# GM drum name
-# GM drum note number
-# Device drum name
-# Device note number
```

Parameters

filespec	Provides the full path to the file to be processed.
csv	Provides the csvarray object to be dumped as a simple INI file.

Returns

Returns 'true' if all operations succeeded.

Definition at line 116 of file iniwriting.cpp.

10.6.3 Variable Documentation

10.6.3.1 const std::string midipp::s_explanatory_header [static]

Provides explanatory text for the top of the INI file.

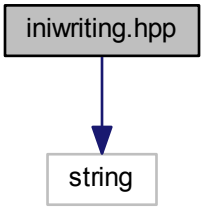
This is C++, so we don't have to worry about the length of character strings.

Definition at line 40 of file iniwriting.cpp.

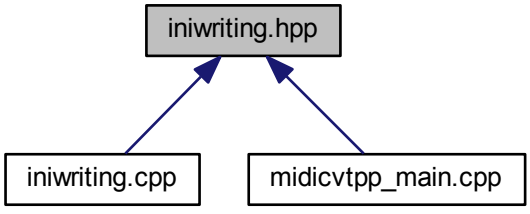
10.7 iniwriting.hpp File Reference

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.

10.7.1 Detailed Description

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

Library libmidipp

Author(s) 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.

10.7.2 Function Documentation

10.7.2.1 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.

Parameters

filespec	Provides the full path to the CSV file to be processed.
csv	Provides the csvarray object to be dumped as a sectioned INI file.
writefooter	Provides an optional parameter, defaulting to true, to determine if the footer at the bottom of the file should be written. This value will be set to false if writing both drums and patch to the file.

Returns

Returns 'true' if all operations succeeded.

Definition at line 190 of file iniwriting.cpp.

10.7.2.2 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.

Parameters

filespec	Provides the full path to the CSV file to be processed.
csv	Provides the csvarray object to be dumped as a sectioned INI file.
writeheader	Provides an optional parameter, defaulting to true, to determine if the header at the top of the file should be written. This value will be set to false if writing both drums and patch to the file.

Returns

Returns 'true' if all operations succeeded.

Definition at line 278 of file iniwriting.cpp.

10.7.2.3 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.

Deprecated The input fields of a Drum map grabbed from a spreadsheet are in the following order:

```
-# GM drum name
-# GM drum note number
-# Device drum name
-# Device note number
```

Parameters

filespec	Provides the full path to the file to be processed.
csv	Provides the csvarray object to be dumped as a simple INI file.

Returns

Returns 'true' if all operations succeeded.

Definition at line 116 of file iniwriting.cpp.

10.8 mainpage-reference.dox File Reference

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

10.8.1 Detailed Description

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

10.9 midi_functions.dox File Reference

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

10.9.1 Detailed Description

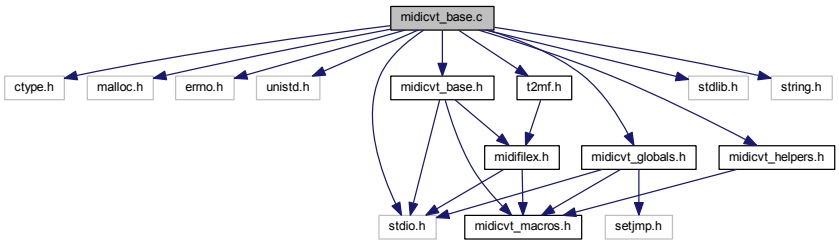
This document describes the functions and parameters of the libmidifilex modules.

10.10 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:



Macros

- #define MFILE_FORMAT_4 "MFile %d %d %d %d\n"

  We now write out "MThd" instead of "MFile" for our generated results.

Functions

- static void checkchan (void)

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

  Check for the end-of-line.
- static void checknote (void)

  Checks the incoming value from flex.

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_smpte (int hr, int mn, int se, int fr, int ff)

  Callback function implementing Mf_smpte().
- 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" or (new) "MThd" token is found.
- static cbool_t redirect_stdout (const char ∗filename, const char ∗mode)

  Redirects stdout to a file.
- static cbool_t revert_stdout ()

  Handles the file descriptor for stdout that was saved 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 normal 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.

10.10.1 Detailed Description

This module provides functions for basic MIDI/text conversions.

Library midicvt application

Author(s) Chris Ahlstrom and many other authors

Date 2014-04-09

Last Edits 2016-04-17

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

System Real-Time Messages:

```
- <code> F8 </code>.      Timing Clock.
- <code> F9 </code>.      Undefined.
- <code> FA </code>.      Start.
- <code> FB </code>.      Continue.
- <code> FC </code>.      Stop.
- <code> FD </code>.      Undefined.
- <code> FE </code>.      Active Sensing.
- <code> FF </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.

System Exclusive Messages:

The raw format of a SysEx event to be sent to a MIDI device is roughly as follows:

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

As encoded in a MIDI file, the SysEx message is in the following format:

1. Delta-time byte. A typical value is 0x00.
2. System-exclusive start byte.
3. A varinum length value that covers all of the following bytes, including the SysEx termination marker.
4. Manufacturer's code.
5. Data bytes.
6. EOSysEx byte.

Some MIDI devices send a SysEx message as a series of small packets with a time-delay between each packet:

1. F0 followed by a number of data bytes, but no F0
2. One or more packets of data bytes that have no F0 or F7.
3. One last packet that ends with an F7.

The MIDI file must encode this single, multi-packet SysEx message as a series of smaller SysE essages, with F7 serving as a "SysEx Continuation" marker. So the above multi-packet message becomes:

1. Initial packet:
   (a) Delta-time byte.
   (b) F0 SysEx start byte.
   (c) A varinum length value.
   (d) Manufacturer's code.
   (e) Data bytes.
2. Continuation packet, one or more:
   (a) Delta-time byte.
   (b) F0 SysEx start byte.
   (c) F7 SysEx continuation byte.
   (d) A varinum length value.
   (e) Data bytes.
3. Final packet.
   (a) Delta-time byte.
   (b) F0 SysEx start byte.
   (c) A varinum length value.
   (d) Data bytes.
   (e) F7 SysEx stop byte.

F7 can also serve as an "Escaped" event. More on that later.

Devices will vary somewhat on the format of the information encoded in the SysEx message. Here's one device and its SysEx description:

http://www.midi-hardware.com/instrukcje/mpot32sysex20.pdf

1. Sys-Ex header: F0

2. Manufacturer ID for MIDI-hardware.com: 00 20 7A
3. Product ID (device ID) for MPOT32: 03
4. Input ID: one byte in range 00..63
5. The command: one byte in range 01..11
6. Command's parameters, dependent on what command was used.
7. Sys-Ex footer: F7

An example Sys-Ex string might look like this:

F0 00 20 7A 03 02 02 01 01 03 F7

Here's another device:

```
-# Sys-Ex header: F0
-# Manufacturer ID for Roland: 41
-# Device ID: 10 by default, could be other values to support
   multiple devices in the same MIDI daisy-chain.
-# Model ID: 42 for a Roland GS synth.
-# Mode: 12 for sending, 11 for a request for information
-# Start address for message: 40007F
-# Data size: Amount of data to send or receive.
-# Infamous Roland checksum.
-# Sys-Ex footer: F7

F0 41 10 42 12 40007F 00 41 F7
```

That's all for now.

10.10.2 Macro Definition Documentation

10.10.2.1 #define MFILE_FORMAT_4 "MFile %d %d %d %d\n"

We now write out "MThd" instead of "MFile" for our generated results.

The reason? We were puzzled why we didn't see the header, after not having run this program for a long time.

Now, files that were written by the old (0.2) version of midicvt can still be read by the new version. One can write out the old tag using the –mfile option.

Definition at line 224 of file midicvt_base.c.

10.10.3 Function Documentation

10.10.3.1 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 1798 of file midicvt_base.c.

10.10.3.2 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 262 of file midicvt_base.c.

10.10.3.3 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 2143 of file midicvt_base.c.

10.10.3.4 static void get16val (void) [static]

Question

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

Definition at line 1904 of file midicvt_base.c.

10.10.3.5 static int getbyte (char * *mess*) [static]

Gets the current byte value as contained in yyval.

Parameters

mess	The message to print if yyval is greater than 127.

Returns

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

Definition at line 1744 of file midicvt_base.c.

10.10.3.6 static int getint (char * *mess*) [static]

Returns the current byte value as contained in yyval.

Parameters

mess	The message to print if yylex() is not equal to INT.

Returns

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

Definition at line 1777 of file midicvt_base.c.

10.10.3.7 void midicvt_compile (void)

This function makes sure the "MFile" or (new) "MThd" 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 1706 of file midicvt_base.c.

10.10.3.8 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 2268 of file midicvt_base.c.

10.10.3.9 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 2348 of file midicvt_base.c.

10.10.3.10 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 exlusive 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

leng	Provides the number of bytes in the message.
mess	Points to the bytes in the message.

Returns

Returns true, always.

Definition at line 1275 of file midicvt_base.c.

10.10.3.11 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

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

Returns

Returns true, always.

Definition at line 851 of file midicvt_base.c.

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

Callback function implementing Mf_error().

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

Parameters

s	Provides the error string to report.

Returns

Returns true, always.

Definition at line 1297 of file midicvt_base.c.

10.10.3.13 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

format	Provides the byte describing the format of the MIDI file. This value will be 0, 1, or 2.
ntrks	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.
division	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 525 of file midicvt_base.c.

10.10.3.14 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

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

Returns

Returns true, always.

Definition at line 1116 of file midicvt_base.c.

10.10.3.15 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 1074 of file midicvt_base.c.

10.10.3.16 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

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

Returns

Returns true, always.

Definition at line 918 of file midicvt_base.c.

10.10.3.17 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

num	Provides the number to be converted to a sequence-number specification.

Returns

Returns true, always.

Definition at line 1046 of file midicvt_base.c.

10.10.3.18 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

leng	Provides the number of bytes in the event.
mess	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 953 of file midicvt_base.c.

10.10.3.19 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

type	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().
leng	Provides the number of bytes in the event.
mess	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 993 of file midicvt_base.c.

10.10.3.20 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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
vol	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 686 of file midicvt_base.c.

10.10.3.21 static int my_non (int *chan,* int *pitch,* int *vol*) [static]

Callback function implementing Mf_on().

Command: `0x9n` .

This function outputs the time using prtime(), 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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
vol	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 655 of file midicvt_base.c.

10.10.3.22 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

chan	Provides the channel number to write.
control	Provides the MIDI controller number to write.
value	Provides the MIDI controller parameter value to write.

Returns

Returns true, always.

Definition at line 755 of file midicvt_base.c.

10.10.3.23 static int my_pitchbend (int *chan,* int *lsb,* int *msb*) [static]

Callback function implementing Mf_pitchbend().

Command: `0xEn` .

- `Ec ll 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

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

Returns

Returns true, always.

Definition at line 793 of file midicvt_base.c.

10.10.3.24 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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
pressure	Provides the MIDI polyphonic key pressure value to write.

Returns

Returns true, always.

Definition at line 714 of file midicvt_base.c.

10.10.3.25 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

chan	Provides the channel number to write.
program	Provides the patch/program number to write.

Returns

Returns true, always.

Definition at line 822 of file midicvt_base.c.

10.10.3.26 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

hr	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 transmitable MIDI events.
mn	Provides the minutes of the SMPTE time.
se	Provides the seconds of the SMPTE time.
fr	Provides the frames of the SMPTE time.
ff	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 1246 of file midicvt_base.c.

10.10.3.27 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 .`

Parameters

leng	Provides the number of bytes in the message.
mess	Provides a pointer to the message bytes.

Returns

Returns true, always.

Definition at line 877 of file midicvt_base.c.

10.10.3.28 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

tempo	Provides the tempo value as a single integer.

Returns

Returns true, always.

Definition at line 1155 of file midicvt_base.c.

10.10.3.29 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

nn	Provides the numerator of the time signature as it would be notated.
dd	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....
cc	Provides the number of MIDI clock in a metronome click.
bb	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 1200 of file midicvt_base.c.

10.10.3.30 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

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

Returns

Returns true, always.

Definition at line 593 of file midicvt_base.c.

10.10.3.31 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 564 of file midicvt_base.c.

10.10.3.32 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 1339 of file midicvt_base.c.

10.10.3.33 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

p	Provides the characters to be written.
leng	Provides the number of characters to be written.

Definition at line 424 of file midicvt_base.c.

10.10.3.34 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

pitch	Provides the MIDI note value to be written.

Returns

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

Definition at line 472 of file midicvt_base.c.

10.10.3.35 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

s	Provides the error string. Note that yyleng and yytext are global inputs to this function, as well.

Returns

Returns true, always.

Definition at line 1662 of file midicvt_base.c.

10.10.3.36 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

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

Definition at line 350 of file midicvt_base.c.

10.10.3.37 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 300 of file midicvt_base.c.

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

Redirects stdout to a file.

Parameters

filename	Provides the filename to be use for the out that would have gone to stdout.
mode	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 2167 of file midicvt_base.c.

10.10.3.39 static cbool_t revert_stdout ( ) [static]

Handles the file descriptor for stdout that was saved for later restoration.

To avoid a memory leak at exit(), this function closes g_redirect_file if it was assigned a value [i.e. fileno(stdio)].

Warning

Currently, the dup2() call is returning error 9, EBADF, a bad file descriptor. We're guessing it is not needed anyway, and commenting it out.

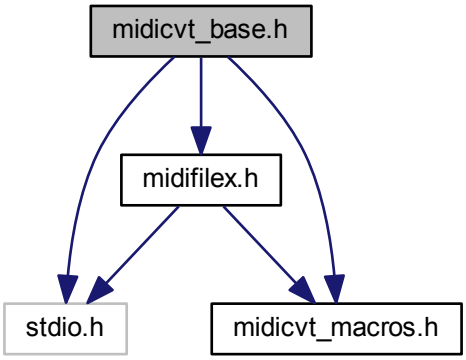
Definition at line 2215 of file midicvt_base.c.

10.11 midicvt_base.h File Reference

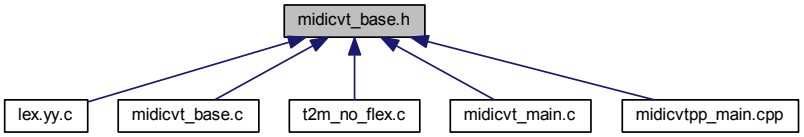
This module provides functions for a basic MIDI/text conversion application.

```
#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" or (new) "MThd" 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 normal if necessary.

10.11.1 Detailed Description

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

Library midicvt application portion of libmidifilex

Author(s) Chris Ahlstrom and many others; see documentation

Date 2014-04-09

Last Edits 2014-05-20

Version

$Revision$ **License** GNU GPL

10.11.2 Function Documentation

10.11.2.1 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 262 of file midicvt_base.c.

10.11.2.2 void midicvt_compile (void)

This function makes sure the "MFile" or (new) "MThd" 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 1706 of file midicvt_base.c.

10.11.2.3 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 2268 of file midicvt_base.c.

10.11.2.4 cbool_t midicvt_setup_mfread (void)

Sets up the file handles for both reading and writing.

Returns

Returns true if all steps succeeded.

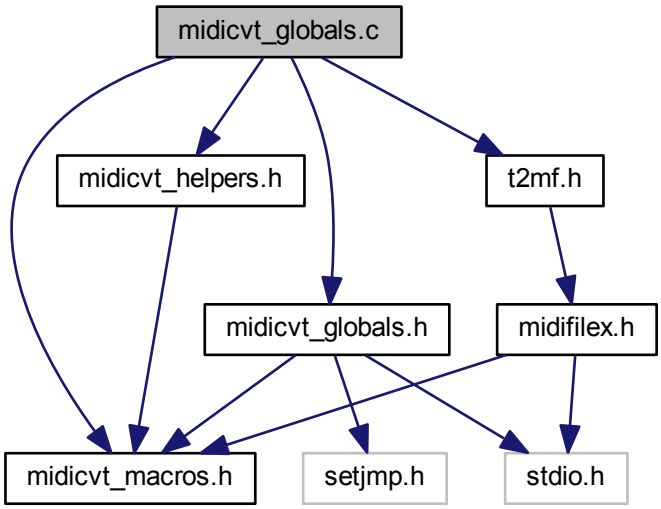
Definition at line 2348 of file midicvt_base.c.

10.12 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_mfile (cbool_t f)

  'Setter' function for member g_option_mfile_tag
- cbool_t midicvt_option_mfile (void)

  'Getter' function for member g_option_mfile_tag
- void midicvt_set_option_strict (cbool_t f)

  'Setter' function for member g_option_strict_track
- cbool_t midicvt_option_strict (void)

  'Getter' function for member g_option_strict_track
- void midicvt_set_option_ignore (cbool_t f)

  'Setter' function for member g_option_ignore_track
- cbool_t midicvt_option_ignore (void)

  'Getter' function for member g_option_ignore_track
- 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

- FILE * g_io_file = nullptr

  Global variables!
- long yyval = 0UL

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

10.12.1 Detailed Description

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

Library midicvt application

Author(s) Chris Ahlstrom

Date 2014-04-09

Last Edits 2015-08-19

Version

$Revision$ **License** GNU GPL

10.12.2 Function Documentation

10.12.2.1 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 308 of file midicvt_globals.c.

10.12.2.2 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 330 of file midicvt_globals.c.

10.12.3 Variable Documentation

10.12.3.1 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.

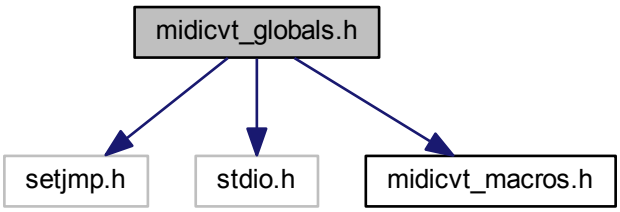
Definition at line 89 of file midicvt_globals.c.

10.13 midicvt_globals.h File Reference

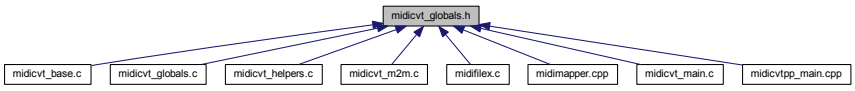
This module provides global variables and accessor functions for the midicvt portion of libmidifilex.

```
#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:



Macros

- #define READMT_EOF EOF

  Provides a return value for readmt() that indicates that EOF was encountered.
- #define READMT_IGNORE_NON_MTRK (-9)

  Provides a return value for readmt() that indicates that "MTrk" was not matched, but we're simply ignoring such results.

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_mfile (cbool_t f)

  'Setter' function for member g_option_mfile_tag
- cbool_t midicvt_option_mfile (void)

  'Getter' function for member g_option_mfile_tag

- void midicvt_set_option_strict (cbool_t f)
  'Setter' function for member g_option_strict_track
- cbool_t midicvt_option_strict (void)
  'Getter' function for member g_option_strict_track
- void midicvt_set_option_ignore (cbool_t f)
  'Setter' function for member g_option_ignore_track
- cbool_t midicvt_option_ignore (void)
  'Getter' function for member g_option_ignore_track
- 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

- FILE * g_io_file
  Global variables!
- long yyval
  Use externs from an include file! The flex-generated code is a bit messy.

10.13.1 Detailed Description

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

Library midicvt application portion of libmidifilex

Author(s) Chris Ahlstrom and others; see documentation

Date 2014-04-09

Last Edits 2015-08-19

Version

$Revision$ **License** GNU GPL

10.13.2 Macro Definition Documentation

10.13.2.1 #define READMT_EOF EOF

Provides a return value for readmt() that indicates that EOF was encountered.

This is a more searchable macro for EOF.

Definition at line 49 of file midicvt_globals.h.

10.13.3 Function Documentation

10.13.3.1 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 308 of file midicvt_globals.c.

10.13.3.2 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 330 of file midicvt_globals.c.

10.13.4 Variable Documentation

10.13.4.1 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.

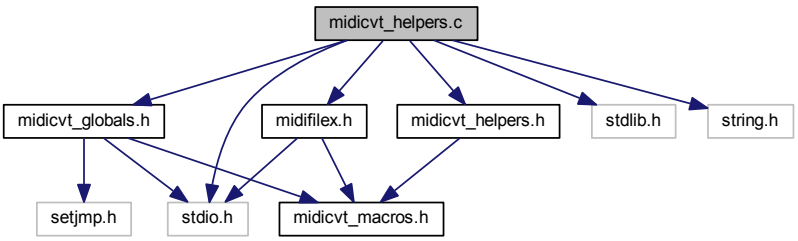
Definition at line 89 of file midicvt_globals.c.

10.14 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.
- cbool_t midi_help_or_version (void)

  Provides the gs_help_or_version flag for use in main().
- static int report (const char *msg)

  Provides an optional and simple reporting function to be used by libmidifileex.
- 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 0.4.0.0"

  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.
- static cbool_t gs_help_or_version = false

  Indicates if –help or –version appeared on the command line.

10.14.1 Detailed Description

This module provides the help text for midicvt.

Library midicvt application

Author(s) Chris Ahlstrom

Date 2014-04-19

Last Edits 2016-04-23

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.

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

10.14.2 Macro Definition Documentation

10.14.2.1 #define MIDICVT_PATH_MAX 1024

Maximum path-name length.

(We don't really support Windows yet.)

Definition at line 128 of file midicvt_helpers.c.

10.14.3 Function Documentation

10.14.3.1 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

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

Definition at line 414 of file midicvt_helpers.c.

10.14.3.2 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 330 of file midicvt_helpers.c.

10.14.3.3 void midicvt_help (const char * *version*)

Provides the help text for the midicvt program.

Parameters

version	Provides the optional version information.

Definition at line 199 of file midicvt_helpers.c.

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

Parses the midicvt (C) command line.

Parameters

argc	Provides the command-line argument count, including the name of the program.
argv	Provides the command-line argument list, including the name of the program.
version	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 452 of file midicvt_helpers.c.

10.14.3.5 cbool_t midicvt_set_input_file (const char * *inputfile*)

Sets the input file-name.

Warning

There is an issue here. If we do not provide a "-i" option to specifiy the input MIDI file, the file specified by the "--m2m" option will be treated as the input file, even though it is an "ini" file. We will consider putting a check here to allow only MIDI file extensions (".mid", ".midi", ".smf*"). Better is to find out what is happening here.

Parameters

inputfile	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 252 of file midicvt_helpers.c.

10.14.3.6 cbool_t midicvt_set_output_file (const char * *outputfile*)

Sets the output file-name.

Parameters

outputfile	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 304 of file midicvt_helpers.c.

10.14.3.7 void midicvt_version (const char * *version*)

Provides the version text for the midicvt-related programs.

Parameters

version	Provides the optional version information.

Definition at line 183 of file midicvt_helpers.c.

10.14.3.8 static int report (const char * *msg*) [static]

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

Parameters

msg	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 378 of file midicvt_helpers.c.

10.14.4 Variable Documentation

10.14.4.1 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 138 of file midicvt_helpers.c.

10.14.4.2 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 153 of file midicvt_helpers.c.

10.14.4.3 cbool_t gs_help_or_version = false [static]

Indicates if –help or –version appeared on the command line.

Returning false from midicvt_parse() is not enough.

Definition at line 173 of file midicvt_helpers.c.

10.14.4.4 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.

10.14.4.5 const char* const gs_help_version = "midicvt v 0.4.0.0" [static]

Default version string.

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

Definition at line 54 of file midicvt_helpers.c.

10.14.4.6 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 145 of file midicvt_helpers.c.

10.14.4.7 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.

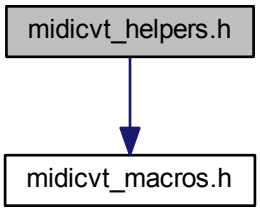
Definition at line 160 of file midicvt_helpers.c.

10.15 midicvt_helpers.h File Reference

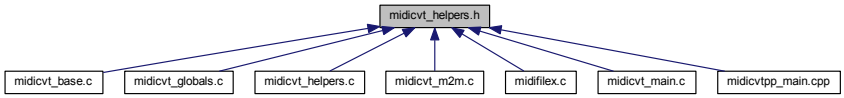
This module declares the help and version functions for midicvt.

```
#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 ∗shortopt, 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.
- cbool_t midi_help_or_version (void)

  Provides the gs_help_or_version flag for use in main().

10.15.1 Detailed Description

This module declares the help and version functions for midicvt.

Library midicvt application

Author(s) 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 the C++ version of midicvt.

10.15.2 Function Documentation

10.15.2.1 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

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

Definition at line 414 of file midicvt_helpers.c.

10.15.2.2 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 330 of file midicvt_helpers.c.

10.15.2.3 void midicvt_help (const char * *version*)

Provides the help text for the midicvt program.

Parameters

version	Provides the optional version information.

Definition at line 199 of file midicvt_helpers.c.

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

Parses the midicvt (C) command line.

Parameters

argc	Provides the command-line argument count, including the name of the program.
argv	Provides the command-line argument list, including the name of the program.
version	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 452 of file midicvt_helpers.c.

10.15.2.5 cbool_t midicvt_set_input_file (const char * *inputfile*)

Sets the input file-name.

Warning

There is an issue here. If we do not provide a "-i" option to specifiy the input MIDI file, the file specified by the "--m2m" option will be treated as the input file, even though it is an "ini" file. We will consider putting a check here to allow only MIDI file extensions (".mid", ".midi", ".smf∗"). Better is to find out what is happening here.

Parameters

inputfile	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 252 of file midicvt_helpers.c.

10.15.2.6 cbool_t midicvt_set_output_file (const char ∗ *outputfile*)

Sets the output file-name.

Parameters

outputfile	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 304 of file midicvt_helpers.c.

10.15.2.7 void midicvt_version (const char ∗ *version*)

Provides the version text for the midicvt-related programs.

Parameters

version	Provides the optional version information.

Definition at line 183 of file midicvt_helpers.c.

10.16 midicvt_license.dox File Reference

This file is a copy of the main license file.

10.16.1 Detailed Description

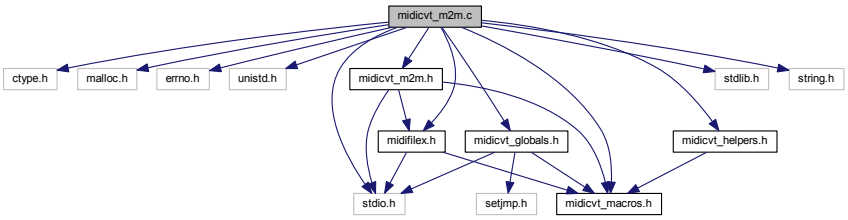
This file is a copy of the main license file.

10.17 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_smpte (int hr, int mn, int se, int fr, int ff)

  Callback function implementing Mf_smpte().
- 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.

10.17.1 Detailed Description

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

Library midicvt application

Author(s) Chris Ahlstrom and many other authors

Date 2014-04-27

Last Edits 2016-04-17

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.

10.17.2 Function Documentation

10.17.2.1 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.

10.17.2.2 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

c	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.

10.17.2.3 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 exlusive continuation.

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

Parameters

leng	Provides the number of bytes in the message.
mess	Points to the bytes in the message.

Returns

Returns true, always.

Definition at line 991 of file midicvt_m2m.c.

10.17.2.4 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 midicvtpp.

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

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

Returns

Returns true, always.

Definition at line 561 of file midicvt_m2m.c.

10.17.2.5 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

s	Provides the error string to report.

Returns

Returns true, always.

Definition at line 128 of file midicvt_m2m.c.

10.17.2.6 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

format	Provides the byte describing the format (0, 1, or 2) of the MIDI file.
ntrks	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.
division	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.

10.17.2.7 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

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

Returns

Returns true, always.

Definition at line 835 of file midicvt_m2m.c.

10.17.2.8 int m2m_meot (void)

Callback function implementing Mf_eot().

This function writes "Mata 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.

10.17.2.9 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

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

Returns

Returns true, always.

Definition at line 635 of file midicvt_m2m.c.

10.17.2.10 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.

10.17.2.11 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

leng	Provides the number of bytes in the event.
mess	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 671 of file midicvt_m2m.c.

10.17.2.12 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_currtime, text_event, mess, (long) leng);

Parameters

type	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().
leng	Provides the number of bytes in the event.
mess	Provides a pointer to the event bytes.

Returns

Returns true, always.

Definition at line 714 of file midicvt_m2m.c.

10.17.2.13 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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
vol	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 371 of file midicvt_m2m.c.

10.17.2.14 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 midicvtpp.

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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
vol	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 335 of file midicvt_m2m.c.

10.17.2.15 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 midicvtpp.

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

chan	Provides the channel number to write.
control	Provides the MIDI controller number to write.
value	Provides the MIDI controller parameter value to write.

Returns

Returns true, always.

Definition at line 450 of file midicvt_m2m.c.

10.17.2.16 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 midicvtpp.

Command: `0xEn` .

- `Ec ll 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

chan	Provides the channel number to write.
lsb	Provides the least-significant bits of the MIDI pitch-wheel value to write.
msb	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.

10.17.2.17 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 midicvtpp.

Command: `0xAn` .

Very similar to Note-On messages [see m2m_non()].

Parameters

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
pressure	Provides the MIDI polyphonic key pressure value to write.

Returns

Returns true, always.

Definition at line 404 of file midicvt_m2m.c.

10.17.2.18 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 midicvtpp.

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

chan	Provides the channel number to write.
program	Provides the patch/program number to write.

Returns

Returns true, always.

Definition at line 528 of file midicvt_m2m.c.

10.17.2.19 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

hr	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 transmitable MIDI events.
mn	Provides the minutes of the SMPTE time.
se	Provides the seconds of the SMPTE time.
fr	Provides the frames of the SMPTE time.
ff	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.

10.17.2.20 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

leng	Provides the number of bytes in the message.
mess	Provides a pointer to the message bytes.

Returns

Returns true, always.

Definition at line 596 of file midicvt_m2m.c.

10.17.2.21 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

tempo	Provides the tempo value as a single integer.

Returns

Returns true, always.

Definition at line 874 of file midicvt_m2m.c.

10.17.2.22 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

nn	Provides the numerator of the time signature as it would be notated.
dd	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....
cc	Provides the number of MIDI clock in a metronome click.
bb	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.

10.17.2.23 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 manufacturer 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

header_offset	Provides the offset in the output file where the "MTrk" and track length were tentatively written.
track_size	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.

10.17.2.24 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.

10.17.2.25 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;

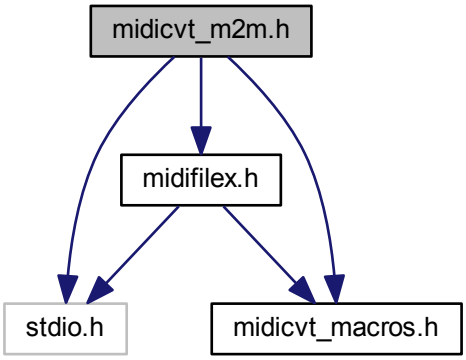
Definition at line 1011 of file midicvt_m2m.c.

10.18 midicvt_m2m.h File Reference

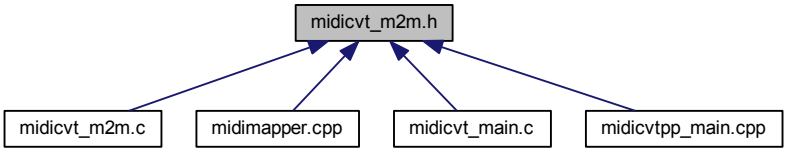
This module provides functions for a basic MIDI-to-MIDI conversion application.

```
#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().

10.18.1 Detailed Description

This module provides functions for a basic MIDI-to-MIDI conversion application.

Library midicvt application portion of libmidifilex

Author(s) Chris Ahlstrom and many others; see documentation

Date 2014-04-29

Last Edits 2014-05-13

Version

$Revision$ **License** GNU GPL

10.18.2 Function Documentation

10.18.2.1 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 midicvtpp.

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

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

Returns

Returns true, always.

Definition at line 561 of file midicvt_m2m.c.

10.18.2.2 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 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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
vol	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 371 of file midicvt_m2m.c.

10.18.2.3 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 midicvtpp.

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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
vol	Provides the MIDI note velocity to write.

Returns

Returns true, always.

Definition at line 335 of file midicvt_m2m.c.

10.18.2.4 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 midicvtpp.

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

chan	Provides the channel number to write.
control	Provides the MIDI controller number to write.
value	Provides the MIDI controller parameter value to write.

Returns

Returns true, always.

Definition at line 450 of file midicvt_m2m.c.

10.18.2.5 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 midicvtpp.

Command: `0xEn` .

- `Ec ll 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

chan	Provides the channel number to write.
lsb	Provides the least-significant bits of the MIDI pitch-wheel value to write.
msb	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.

10.18.2.6 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 midicvtpp.

Command: `0xAn` .

Very similar to Note-On messages [see m2m_non()].

Parameters

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
pressure	Provides the MIDI polyphonic key pressure value to write.

Returns

Returns true, always.

Definition at line 404 of file midicvt_m2m.c.

10.18.2.7 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 midicvtpp.

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

chan	Provides the channel number to write.
program	Provides the patch/program number to write.

Returns

Returns true, always.

Definition at line 528 of file midicvt_m2m.c.

10.18.2.8 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.

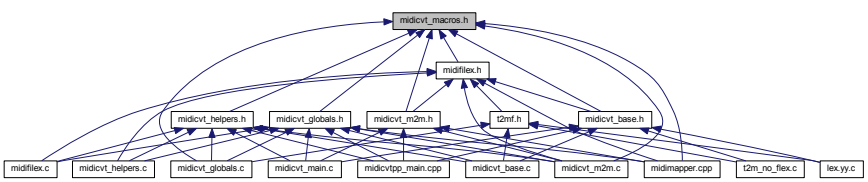
Mf_eot = m2m_meot;

Definition at line 1011 of file midicvt_m2m.c.

10.19 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 bool_to_cstr(x) ((x) ? "true" : "false")

  Provides an easy way to convert a boolean to a string.
- #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.

10.19.1 Detailed Description

This module provides macros for generating simple messages, MIDI parameters, and more.

Library libmidifilex

Author(s) Chris Ahlstrom and other authors; see documentation

Date 2013-11-17

Last Edits 2016-04-19

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.

10.19.2 Macro Definition Documentation

10.19.2.1 #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.

10.19.2.2 #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.

10.19.2.3 #define nullptr 0

Language macros:

Provides an alternative to NULL.

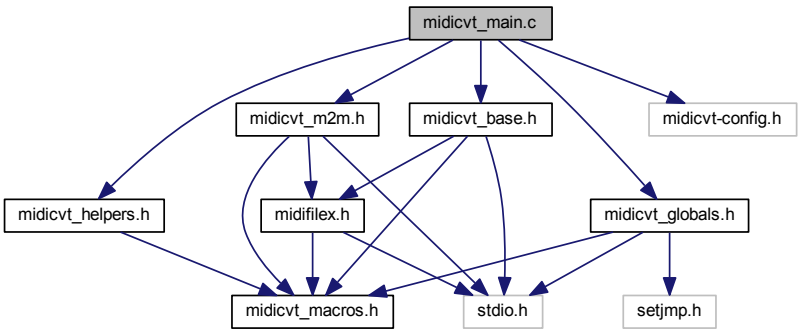
Definition at line 110 of file midicvt_macros.h.

10.20 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 "midicvt-config.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

  Version string.

10.20.1 Detailed Description

This module creates the midicvt program for translating between MIDI files and text files.

Library midicvt application

Author(s) Major modifications by Chris Ahlstrom

Date 2014-04-09

Last Edits 2016-04-15

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.

10.20.2 Function Documentation

10.20.2.1 int main (int *argc,* char * *argv[ ]*)

Provides the entry-point for the midicvt program.

Parameters

argc	Provides the standard count of the number of command-line arguments, including the name of the program.
argv	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.

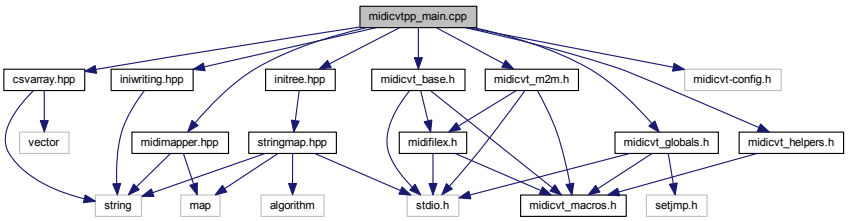
Definition at line 88 of file midicvt_main.c.

10.21 midicvtpp_main.cpp File Reference

This module creates the midicvtpp program for translating between MIDI files and text files, and doing some basic remapping operations.

```
#include <csvarray.hpp>
#include <initree.hpp>
#include <iniwriting.hpp>
#include <midicvt_base.h>
#include <midicvt_helpers.h>
#include <midicvt_globals.h>
#include <midicvt_m2m.h>
#include <midimapper.hpp>
#include "midicvt-config.h"
```

Include dependency graph for midicvtpp_main.cpp:



Functions

- static void midicvtpp_help ()

  Provides the help text for the midicvt program.
- static bool midicvtpp_parse (int argc, char ∗argv[ ])

  Parses midicvtpp the command-line for options.
- int main (int argc, char ∗argv[ ])

  Provides the entry-point for the midicvtpp program.

Variables

- static const char ∗const s_help_version

  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 = "midicvtpp"

  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.

- static bool s_summarize_conversion = false

  Indicates that we want summary output, which is set using the –summarize option.

10.21.1 Detailed Description

This module creates the midicvtpp program for translating between MIDI files and text files, and doing some basic remapping operations.

Library midicvtpp application

Author(s) Chris Ahlstrom

Date 2014-04-19

Last Edits 2016-04-23

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.

10.21.2 Function Documentation

10.21.2.1 int main (int *argc,* char * *argv[ ]*)

Provides the entry-point for the midicvtpp program.

Parameters

argc	Provides the standard count of the number of command-line arguments, including the name of the program.
argv	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 364 of file midicvtpp_main.cpp.

10.21.2.2 static void midicvtpp_help ( ) [static]

Provides the help text for the midicvt program.

Parameters

version	Provides the optional version information.

Definition at line 101 of file midicvtpp_main.cpp.

10.21.2.3 static bool midicvtpp_parse (int *argc,* char * *argv[ ]*) [static]

Parses midicvtpp the command-line for options.

First, calls midicvt_parse() to get the midicvt options, and then parses for the the midicvtpp-specific options.

Parameters

argc	The standard argument count for the command-line.
argv	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 202 of file midicvtpp_main.cpp.

10.21.3 Variable Documentation

10.21.3.1 const char* const s_help_version [static]

Initial value:

```
=
   "midicvtpp v " MIDICVT_VERSION " " MIDICVT_VERSION_DATE_SHORT
```

Provides the version string for this program.

Please leave "midicvtpp" 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 61 of file midicvtpp_main.cpp.

10.21.3.2 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 143 of file midicvtpp_main.cpp.

10.21.3.3 std::string s_mapping_name = "midicvtpp" [static]

Provides the name of the MIDI remapping process.

This name is just a tag name for output, and defaults to "midicvtpp". However, the –testing option changes it to "testing", which is an internal signal for unknown purposes.

Definition at line 159 of file midicvtpp_main.cpp.

10.21.3.4 bool s_summarize_conversion = false [static]

Indicates that we want summary output, which is set using the –summarize option.

This output is similar to that produced by the –debug option, but only shows values that were actually converted during the MIDI mapping.

Definition at line 182 of file midicvtpp_main.cpp.

10.21.3.5 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 111 of file midicvtpp_main.cpp.

10.21.3.6 bool s_write_csv_patch = false [static]

Holds the flag needed for the –csv-patch options.

See the help text for this option.

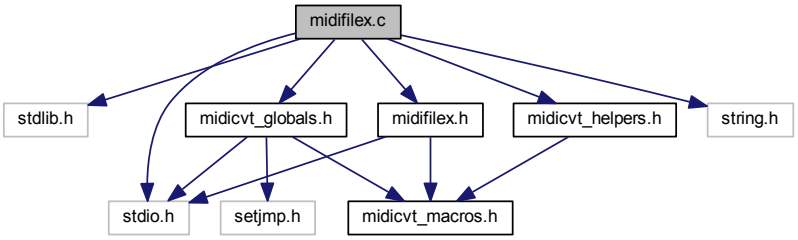
Definition at line 118 of file midicvtpp_main.cpp.

10.22 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 "midicvt_globals.h"
#include "midicvt_helpers.h"
#include "midifilex.h"
```

Include dependency graph for midifilex.c:



Functions

- static void mferror (char *s)

  Reports an error, then calls Mf_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 int egetc (void)

  Reads a single character using the Mf_getc() callback.
- 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, decrementing s_Mf_toberead with every character obtained.
- 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 writevarinum (unsigned long value)

  Write multi-length bytes to MIDI format files.
- static void metaevent (int type)

  Handles a meta-event.
- static int readmt (const char *s)

  Reads through the "MThd" or "MTrk" header string.
- static int readheader (void)

  Reads a header chunk.
- static long get_lookfor ()

Replaces the following line of code, trying to get easier debugging without introducing a nasty side-effect on s_Mf↩ _toberead.

- static void continuation_error (int c)

  Helper code for SysEx continuation errors.
- static void delta_time_report (long dtime)

  Helper code for delta-time reporting.
- static void chunk_size_report (long toberead)

  Helper code for chunk-size reporting.
- static cbool_t readtrack (cbool_t is_m2m)

  Reads a track chunk for MIDI-to-ASCII or for MIDI-to-MIDI conversion.
- 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.
- 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'ed system exclusives are not collapsed.
- long Mf_currtime = 0L

  Current time in delta-time units.
- static long s_Mf_toberead = 0L

  Private book-keeping global (static) variables for the number of bytes expected in an event, and the number of bytes written.
- 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 long s_track_header_offset = 0

  Saves the offset of the header portion of the current track.

- static int s_laststat = 0

  Holds the last status code.

- static int s_lastmeta = 0

  Holds the last meta event type.

10.22.1 Detailed Description

This module provides functions for handling the reading and writing of MIDI files.

Library midicvt application

Author(s) Other authors (see below), with modifications by Chris Ahlstrom,

Date 2014-04-08

Last Edits 2016-04-19

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 improved 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.

Running Status:

Running status speeds up the sending of MIDI bytes to a synthesizer/sequencer by using redundancy where possible. For example, if sending a consecutive group of Note On and Note Off messages to a particular channel, some time can be saved by not sending the channel status byte after the first time. Here's an example with Note On on channel 1:

```
    0x90 3C 7F
    0x90 40 7F
    0x90 43 F3


Since no change in status occurs after the first of these three
events, we can drop the subsequent status bytes:


    0x90 3C 7F
    40 7F
    43 F3


The 0x90 byte is saved in a "running status buffer" (RSB), and is
filled in by the receiving device.  Here is the sequence of events
for operating with running status.

-# Clear the RSB buffer (RSB = 0) to start.
-# If a <b>Voice Category Status</b> (VCS) byte is received, then set
   RSB = VCS.  VCS bytes range from 0x80 to 0xEF.  This is binary
   1000000 to 11100000.
-# If a data byte is received (data bytes range from 0x00 to 0x7F,
   binary 0000000 to 0111111; that is, bit 7 is always 0 in a data byte):
   -# If RSB != 0, first insert the RSB into the incoming data
      stream, then insert the data byte.
   -# If RSB == 0, then just insert the data byte into the incoming data
      stream.
-# Clear the RSB buffer (RSB = 0) when a System Common Message (SCM)
   status byte is received.  SCM bytes range from 0xF0 to 0xF7.
-# The message after an SCM <b>must</b> begin with a status byte.
   That is a byte with bit 7 set.
-# Do no special action when a Realtime Category Message (RCM) byte is
   received.  RCM bytes range from 0xF8 to 0xFF.
```

10.22.2 Function Documentation

10.22.2.1 static void badbyte (int *c*) [static]

Provides an error report for a bad byte.

Since mferror() is called, the application will exit.

Parameters

c	Provides the bad byte, in integer format.

Definition at line 285 of file midifilex.c.

10.22.2.2 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 382 of file midifilex.c.

10.22.2.3 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

status	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).
c1	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.
c2	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 bye of a pitch-wheel change, or this parameter can be left unused.

Definition at line 527 of file midifilex.c.

10.22.2.4 static int egetc (void) [static]

Reads a single character using the Mf_getc() callback.

This function also decrements s_Mf_toberead, as a side-effect. This function will call mferror() to abort on EOF.

Returns

Returns the character read by the Mf_getc() callback.

Definition at line 337 of file midifilex.c.

10.22.2.5 static int eputc (unsigned char *c*) [static]

Writes a single character.

If an error occurs, then this functon calls mferror() and aborts. [But mferror() will call exit().]

Parameters

c	Provides the character to output with the Mf_putc() callback function.

Returns

Returns the return value of Mf_putc(). If this value is EOF, then mferror() is called.

Definition at line 311 of file midifilex.c.

10.22.2.6 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().

Change Note ca 2015-10-11 The current version of the test file b4uacuse-GM-format.midi has a missing value near the beginning. Actually, the value isn't missing. It's just that it is a sequence number of 0, which is written in the allowed alternate format, "FF 00 00", instead of the normal format "FF 00 02 ss ss", where "ss ss" would be "00 00". Anyway, this cause a null msg() return, which we must ignore, to avoid a crash.

Parameters

type	Provides the type of meta event. The following value sets are handled: • 0x00. Sequence number. The Mf_seqnum() callback is called. • 0x01 to 0x0f. Text event. 0x01 to 0x07 are standard text events, while the rest are "unrecognized". The Mf_text() callback is called. • 0x2f. End of track. The Mf_eot() callback is called. • 0x51. Set tempo. The Mf_tempo() callback is called. • 0x54. SMPTE. The Mf_smpte() callback is called. • 0x58. Time signature. The Mf_timesig() callback is called. • 0x59. Key signature. The Mf_keysig() callback is called. • 0x7f. Sequencer specific. The Mf_sqspecific() callback is called. • Default. Miscellaneous. The Mf_metamisc() callback is called.

Definition at line 849 of file midifilex.c.

10.22.2.7 unsigned long mf_sec2ticks (float *secs*, int *division*, unsigned int *tempo*)

Converts seconds to ticks.

Calculates the value of

```
1000 * secs      1
------------ * -----
4 * division   tempo
```

Parameters

secs	Provides the seconds value to be converted.
division	Provides the division units.
tempo	Provides the tempo value

Returns

Returns the value of the seconds in ticks, as per the formula shown.

Definition at line 2013 of file midifilex.c.

10.22.2.8 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

ticks	Provides the ticks value to be converted.
division	Provides the division units.
tempo	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 2039 of file midifilex.c.

10.22.2.9 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

format	Provides the format byte to describe the SMF type of the file.
ntracks	Provides the number of tracks in the file.
division	Provides the division parameter of the file.

Definition at line 1678 of file midifilex.c.

10.22.2.10 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

delta_time	Provides the time in ticks since the last event.
type	Provides the type of meta event.
data	Provides A pointer to a block of chars containing the META EVENT, data.
size	Provides the length of the meta-event data.

Returns

Returns the number of bytes written for this meta-event. This functio 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 1895 of file midifilex.c.

10.22.2.11 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

delta_time	Provides the time in ticks since the last event.
type	Provides the type of event.
chan	Provides The midi channel.
data	Provides A pointer to a block of chars containing the META EVENT, data.
size	Provides The length of the midi-event data.

Returns

Returns the *size* parameter.

Definition at line 1827 of file midifilex.c.

10.22.2.12 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

delta_time	Provides the time in ticks since the last event.
tempo	Provides the temp value to write.

Definition at line 1976 of file midifilex.c.

10.22.2.13 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

which_track	Indicates the track number of the track to be written. If -1, then the track is a tempo-track.
fp	The output file descriptor.
wtrack	The function to call to do that actual writing. Usually, this function is either Mf_wtempotrack or Mf_wtrack.

Definition at line 1547 of file midifilex.c.

10.22.2.14 void mf_w_track_start (int *which_track,* FILE * *fp*)

Reads and writes track information.

Parameters

which_track	Indicates the track number of the track to be written. If -1, then the track is a tempo-track.
fp	The output file descriptor.

Definition at line 1634 of file midifilex.c.

10.22.2.15 static void mferror (char * *s*) [static]

Reports an error, then calls Mf_error if the Mf_error callback has been assigned, then exits with an error-code of 1.

Parameters

s	Provides the error message.

Definition at line 233 of file midifilex.c.

10.22.2.16 void mfread (void)

Calls readheader(), then calls readtrack() while there is data to be read.

Once done, we delete the message buffer to avoid a valgrind leakage indication at exit.

Note

This function and mfwrite() are the only non-static functions in this file? Not any more!

Definition at line 1479 of file midifilex.c.

10.22.2.17 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

s	Provides the error message.

Definition at line 258 of file midifilex.c.

10.22.2.18 static cbool_t mfreportable (void) [inline],[static]

Returns

Returns true if the Mf_report function is enabled.

Definition at line 270 of file midifilex.c.

10.22.2.19 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.

Once done, we delete the message buffer to avoid a valgrind leakage indication at exit.

Definition at line 2082 of file midifilex.c.

10.22.2.20 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

format	Indicates the level of SMF (standard MIDI file) support. • 0: Single multi-channel track. • 1: Multiple simultaneous tracks. • 2: One or more sequentially independent single track patterns.
ntracks	Provides the number of tracks in the file.
division	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 similiar 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.
fp	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 Mf_putc.

Definition at line 1739 of file midifilex.c.

10.22.2.21 void midifile (void)

Provided for backward compatibility with the original library.

This function simply calls mfread().

Definition at line 1502 of file midifilex.c.

10.22.2.22 static char* msg (void) [static]

Provides a pointer to the message buffer.

Returns

Returns the s_message_buffer static variable.

Definition at line 424 of file midifilex.c.

10.22.2.23 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

c	The character to add to the message buffer.

Definition at line 453 of file midifilex.c.

10.22.2.24 static int msgleng (void) [inline], [static]

Provides the current index into the message buffer.

Returns

Returns the value of s_message_index.

Definition at line 437 of file midifilex.c.

10.22.2.25 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 740 of file midifilex.c.

10.22.2.26 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 723 of file midifilex.c.

10.22.2.27 static int 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 1097 of file midifilex.c.

10.22.2.28 static int readmt (const 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, if midicvt_option_strict() is true. If it is false, tracks with other chunk names can be processed.

If midicvt_option_ignore() is true, non-MTrk chunks are allowed, but ignored.

Parameters

s	Provides the string that is expected to be read from the file.

Returns

Returns the last character obtained, or READMT_EOF if not characters could be read (and the –strict option is in force). Also returns READMT_EOF if the match was unable to be detected. Returns READMT_IGNO↩ RE_NON_MTRK if there is no match, but the –ignore option is active.

Definition at line 1039 of file midifilex.c.

10.22.2.29 static cbool_t readtrack (cbool_t *is_m2m*) [static]

Reads a track chunk for MIDI-to-ASCII or for MIDI-to-MIDI conversion.

Legacy:

First, readmt() is called to verify that "MTrk" (or an unknown chunk) was retrieved from the file. If this succeeds, then this function reads the length of the track (32 bits). This value is saved in the global variable s_Mf_toberead. Then Mf_currtime 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. An SCM.
- 0xf1 to 0xf6. Various SCM messages, ignored at present.
- 0xf7. SysEx continuation or arbitrary data, an SCM.

The receipt of an SCM should clear the RSB. (See the top of the module for the meaning of the abbreviations.)

A lot of other processing is done (see the code), and then the Mf_endtrack() calllback is called.

M2M:

This call 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.

There is one big puzzle to figure out... Why does this function have to *set* the current time in M2M mode, rather than add to it the way readtrack() does?

```
Mf_currtime += readvarinum();    // legacy
Mf_currtime  = readvarinum();    // m2m
```

Running Status:

At the code tagged below as "Running Status", we see that 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. This is what running status does. Also see the documentation on running status for this whole module.

Note the "running" boolean. If false, this indicates that we just got a status byte and are saving it for a possible usage as running status. If true, we have an RSB already, and now have a data byte.

Parameters

is_m2m	Provides a way to do things slightly differently for the M2M mode. u

Returns

Returns true if the "MTrk" marker was found. Actually, if any marker is found, and there is no EOF returned.

Definition at line 1301 of file midifilex.c.

10.22.2.30 static long readvarinum (void) [static]

Read a varying-length number, decrementing s_Mf_toberead with every character obtained.

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 641 of file midifilex.c.

10.22.2.31 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 605 of file midifilex.c.

10.22.2.32 static short int to16bit (int *c1*, int *c2*) [inline], [static]

Convert a duo of integers into a 16-bit number.

Parameters

c1	Provides the most-significant portion of the number. This portion gets shifted leftward by 8 bits once.
c2	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 705 of file midifilex.c.

10.22.2.33 static long to32bit (int *c1*, int *c2*, int *c3*, int *c4*) [static]

Convert a quad of integers into a 32-bit number.

Parameters

c1	Provides the most-significant portion of the number. This portion gets shifted leftward by 8 bits three times.
c2	Provides the second portion of the number.
c3	Provides the third portion of the number.
c4	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 680 of file midifilex.c.

10.22.2.34 static void write16bit (int *data*) [static]

Writes two bytes of data, one byte at a time.

Parameters

data	Provides the 16 bits of data to be written, one byte at a time.

Definition at line 780 of file midifilex.c.

10.22.2.35 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

data	Provides the 32 bits of data to be written, one byte at a time.

Definition at line 764 of file midifilex.c.

10.22.2.36 static void writevarinum (unsigned long *value*) [static]

Write multi-length bytes to MIDI format files.

We changed the name of this function to "writevarinum()" to match "readvarinum()" and cut down on some confusion.

Parameters

value	Provides the value to be written.

Definition at line 796 of file midifilex.c.

10.22.3 Variable Documentation

10.22.3.1 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 1144 of file midifilex.c.

10.22.3.2 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 360 of file midifilex.c.

10.22.3.3 long s_Mf_toberead = 0L [static]

Private book-keeping global (static) variables for the number of bytes expected in an event, and the number of bytes written.

Definition at line 221 of file midifilex.c.

10.22.3.4 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.

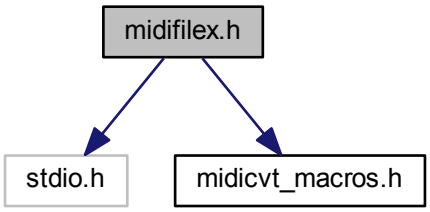
Definition at line 1235 of file midifilex.c.

10.23 midifilex.h File Reference

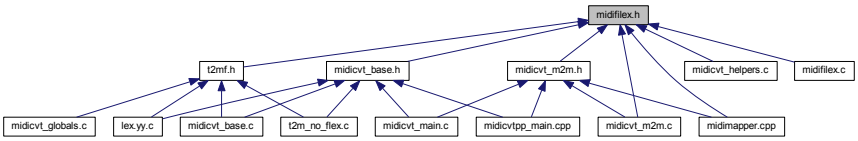
This module provides functions for libmidifilex.

```
#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'ed system exclusives are not collapsed.
- long Mf_currtime

  Current time in delta-time units.
- int(∗ Mf_getc)(void)

  Functions to be called while processing and writing the MIDI file.

10.23.1 Detailed Description

This module provides functions for libmidifilex.

Library libmidifilex

Author(s) Chris Ahlstrom and many other authors; see documentation

Date 2014-04-08

Last Edits 2014-05-20

Version

$Revision$ **License** GNU GPL

10.23.2 Macro Definition Documentation

10.23.2.1 #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.

10.23.3 Function Documentation

10.23.3.1 unsigned long mf_sec2ticks (float *secs,* int *division,* unsigned int *tempo*)

Converts seconds to ticks.

Calculates the value of

```
1000 * secs      1
------------ * -----
4 * division   tempo
```

Parameters

secs	Provides the seconds value to be converted.
division	Provides the division units.
tempo	Provides the tempo value

Returns

Returns the value of the seconds in ticks, as per the formula shown.

Definition at line 2013 of file midifilex.c.

10.23.3.2 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

ticks	Provides the ticks value to be converted.
division	Provides the division units.
tempo	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 2039 of file midifilex.c.

10.23.3.3 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

format	Provides the format byte to describe the SMF type of the file.
ntracks	Provides the number of tracks in the file.
division	Provides the division parameter of the file.

Definition at line 1678 of file midifilex.c.

10.23.3.4 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

delta_time	Provides the time in ticks since the last event.
type	Provides the type of meta event.
data	Provides A pointer to a block of chars containing the META EVENT, data.
size	Provides the length of the meta-event data.

Returns

Returns the number of bytes written for this meta-event. This functio 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 1895 of file midifilex.c.

10.23.3.5 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

delta_time	Provides the time in ticks since the last event.
type	Provides the type of event.
chan	Provides The midi channel.
data	Provides A pointer to a block of chars containing the META EVENT, data.
size	Provides The length of the midi-event data.

Returns

Returns the *size* parameter.

Definition at line 1827 of file midifilex.c.

10.23.3.6 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

delta_time	Provides the time in ticks since the last event.
tempo	Provides the temp value to write.

Definition at line 1976 of file midifilex.c.

10.23.3.7 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

which_track	Indicates the track number of the track to be written. If -1, then the track is a tempo-track.
fp	The output file descriptor.
wtrack	The function to call to do that actual writing. Usually, this function is either Mf_wtempotrack or Mf_wtrack.

Definition at line 1547 of file midifilex.c.

10.23.3.8 void mf_w_track_start (int *which_track,* FILE * *fp*)

Reads and writes track information.

Parameters

which_track	Indicates the track number of the track to be written. If -1, then the track is a tempo-track.
fp	The output file descriptor.

Definition at line 1634 of file midifilex.c.

10.23.3.9 void mfread (void)

Calls readheader(), then calls readtrack() while there is data to be read.

Once done, we delete the message buffer to avoid a valgrind leakage indication at exit.

Note

This function and mfwrite() are the only non-static functions in this file? Not any more!

Definition at line 1479 of file midifilex.c.

10.23.3.10 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.

Once done, we delete the message buffer to avoid a valgrind leakage indication at exit.

Definition at line 2082 of file midifilex.c.

10.23.3.11 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

format	Indicates the level of SMF (standard MIDI file) support. • 0: Single multi-channel track. • 1: Multiple simultaneous tracks. • 2: One or more sequentially independent single track patterns.
ntracks	Provides the number of tracks in the file.
division	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 similiar 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.
fp	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 Mf_putc.

Definition at line 1739 of file midifilex.c.

10.23.3.12 void midifile (void)

Provided for backward compatibility with the original library.

This function simply calls mfread().

Definition at line 1502 of file midifilex.c.

10.23.3.13 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

data	Provides the 32 bits of data to be written, one byte at a time.

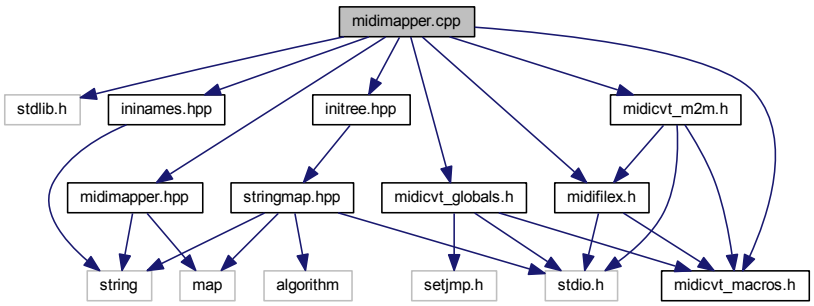
Definition at line 764 of file midifilex.c.

10.24 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 midipp::show_maps (const std::string &tag, const midipp::midimapper &container, bool full_output)

  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.

10.24.1 Detailed Description

This module provides functions for advanced MIDI/text conversions.

Library libmidipp

Author(s) Chris Ahlstrom

Date 2014-04-24

Last Edits 2016-04-23

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().

10.24.2 Function Documentation

10.24.2.1 static int midimap_chanpressure (int *chan*, int *pressure*) [static]

For filtering channel-pressure messages.

Parameters

chan	Provides the channel number to write.
pressure	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 1262 of file midimapper.cpp.

10.24.2.2 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

mm	Provides the MIDI mapper object to be used.

Definition at line 976 of file midimapper.cpp.

10.24.2.3 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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
vol	Provides the MIDI note velocity to write.

Returns

Returns the return-value of the corresponding midicvt_m2m callback function.

Definition at line 1063 of file midimapper.cpp.

10.24.2.4 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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
vol	Provides the MIDI note velocity to write.

Returns

Returns the return-value of the corresponding midicvt_m2m callback function.

Definition at line 1017 of file midimapper.cpp.

10.24.2.5 static int midimap_parameter (int *chan,* int *control,* int *value*) [static]

For filtering parameter/control messages.

Parameters

chan	Provides the channel number to write.
control	Provides the MIDI controller number to write.
value	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 1193 of file midimapper.cpp.

10.24.2.6 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

chan	Provides the channel number to write.
patch	Provides the patch/program number to write.

Returns

Returns the return-value of the corresponding midicvt_m2m callback function.

Definition at line 1152 of file midimapper.cpp.

10.24.2.7 static int midimap_pitchbend (int *chan,* int *lsb,* int *msb*) [static]

For filtering pitchbend messages.

Parameters

chan	Provides the channel number to write.
lsb	Provides the least-significant bits of the MIDI pitch-wheel value to write.
msb	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 1230 of file midimapper.cpp.

10.24.2.8 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

chan	Provides the channel number to write.
pitch	Provides the MIDI note number to write.
pressure	Provides the MIDI polyphonic key pressure value to write.

Returns

Returns the return-value of the corresponding midicvt_m2m callback function.

Definition at line 1109 of file midimapper.cpp.

10.24.2.9 void midipp::show_maps (const std::string & *tag*, const midipp::midimapper & *container*, bool *full_output*)

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 each container.

Parameters

tag	Identifies the object in the human-readable output.
container	The midimapper through which iteration is done for showing.
full_output	Defaults to true, which means to show everything. If false, map entries not used in the conversion are not shown.

Definition at line 790 of file midimapper.cpp.

10.24.3 Variable Documentation

10.24.3.1 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.

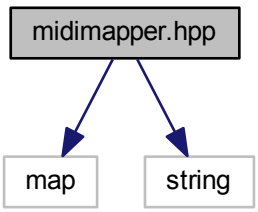
Definition at line 960 of file midimapper.cpp.

10.25 midimapper.hpp File Reference

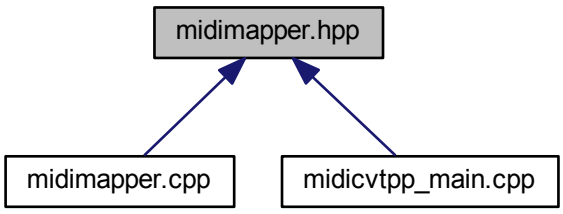
This module provides functions for advanced MIDI/text conversions.

```
#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::annotation

  This class is meant to extend the map of values with additional data that can be written out to summarize some information about the MIDI remapping that was done.
- 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 midipp::show_maps (const std::string &tag, const midipp::midimapper &container, bool full_output)

  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.

10.25.1 Detailed Description

This module provides functions for advanced MIDI/text conversions.

Library libmidipp

Author(s) Chris Ahlstrom

Date 2014-04-24

Last Edits 2016-04-23

Version

$Revision$ **License** GNU GPL

The mapping process works though 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 = midicvtpp
        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
```

10.25.2 Function Documentation

10.25.2.1 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

mm	Provides the MIDI mapper object to be used.

Definition at line 976 of file midimapper.cpp.

10.25.2.2 void midipp::show_maps (const std::string & *tag,* const midipp::midimapper & *container,* bool *full_output*)

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 each container.

Parameters

tag	Identifies the object in the human-readable output.
container	The midimapper through which iteration is done for showing.
full_output	Defaults to true, which means to show everything. If false, map entries not used in the conversion are not shown.

Definition at line 790 of file midimapper.cpp.

10.26 midipp_functions.dox File Reference

This document describes the functions and parameters of the libmidipp module.

10.26.1 Detailed Description

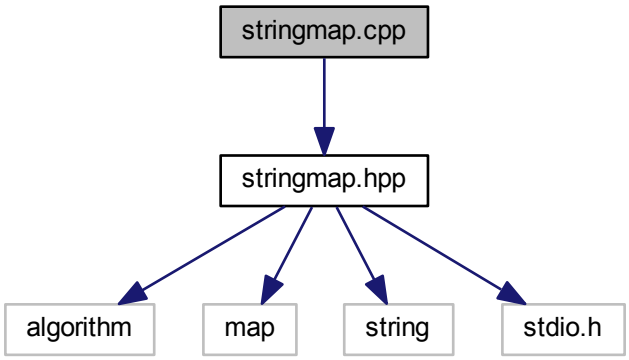
This document describes the functions and parameters of the libmidipp module.

10.27 stringmap.cpp File Reference

Library libmidipp

```
#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.

10.27.1 Detailed Description

Library libmidipp

Author(s) 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.

10.27.2 Function Documentation

10.27.2.1 bool midipp::iequal (const std::string & *s1*, const std::string & *s2*)

A simple case-insensitive comparison for simple string equality.

It is a bit similar to the cmp_nocase() function in section 20.3.8 "Comparisons" of Stroustrop's C++ Programming Language book.

We just stick it in this existing module for convenience.

Note

If your application is already using Boost, you can use the boost::iequal() function instead.

Parameters

s1	Provides the first string.
s2	Provides the second string.

Returns

Returns true of the strings are identical except for case.

Definition at line 42 of file stringmap.cpp.

10.27.2.2 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

tag	A name for the string, to provide context for the human reader.
s	The string to be shown.

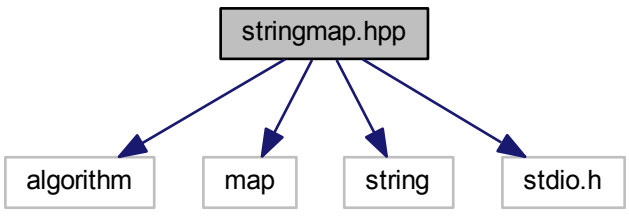
Definition at line 83 of file stringmap.cpp.

10.28 stringmap.hpp File Reference

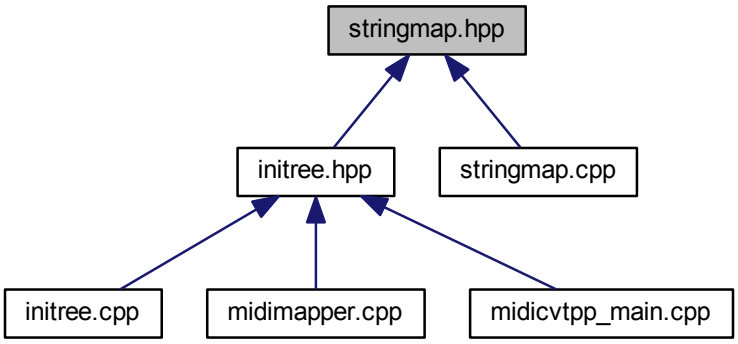
Library libmidipp

```
#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.

10.28.1 Detailed Description

Library libmidipp

Author(s) 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.

10.28.2 Function Documentation

10.28.2.1 bool midipp::iequal (const std::string & *s1*, const std::string & *s2*)

A simple case-insensitive comparison for simple string equality.

It is a bit similar to the cmp_nocase() function in section 20.3.8 "Comparisons" of Stroustrop's C++ Programming Language book.

We just stick it in this existing module for convenience.

Note

If your application is already using Boost, you can use the boost::iequal() function instead.

Parameters

s1	Provides the first string.
s2	Provides the second string.

Returns

Returns true of the strings are identical except for case.

Definition at line 42 of file stringmap.cpp.

10.28.2.2 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

tag	A name for the string, to provide context for the human reader.
s	The string to be shown.

Definition at line 83 of file stringmap.cpp.

10.28.2.3 template<class VALUETYPE > void midipp::show (const std::string & *tag*, const stringmap< VALUETYPE > & *container*)

Writes out the contents of the property container.

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

Usage

```
resultrow r;
(void) r.insert("name", "value");
show(std::string("Row"), r);
```

Parameters

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

Definition at line 471 of file stringmap.hpp.

10.28.2.4 template<class VALUETYPE > void midipp::show_pair (const typename stringmap< VALUETYPE >::pair & *p*)

Writes out the contents of an string/VALUETYPE pair.

This function exists in order to be used in an std::for_each() call.

Warning

The VALUETYPE object must have its own overload of the global show() function.

Parameters

p	The pair value to be shown.

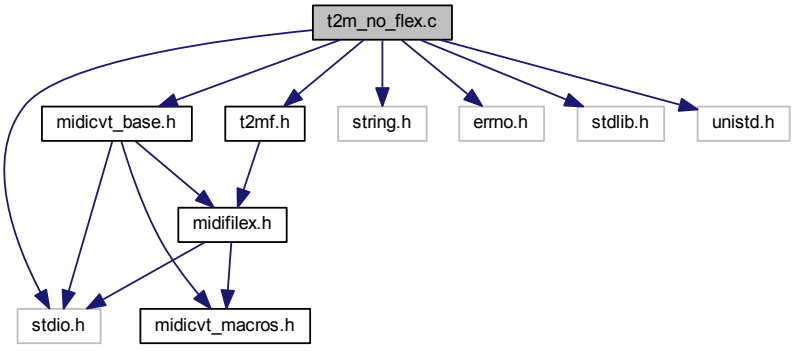
Definition at line 439 of file stringmap.hpp.

10.29 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.

10.29.1 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(s) Chris Ahlstrom and many other authors

Date 2014-04-09

Last Edits 2015-11-18

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.

Change Note ca 2015-08-14 We want to support "MThd" as an alternative to "MFile" in parsing an ASCII file, so we modified t2mf.fl to also return "MTHD" when encounterding "MThd". Then we rebuilt this module, which was quite a task, even with the help of gvimdiff! Compare the test files in the "results" directory: ex1-mthd.asc versus ex1.asc.

10.29.2 Function Documentation

10.29.2.1 YY_BUFFER_STATE yy_create_buffer (FILE * *file*, int *size*)

Allocate and initialize an input buffer state.

Parameters

file	A readable stream.
size	The character buffer size in bytes. When in doubt, use `YY_BUF_SIZE`.

Returns

the allocated buffer state.

Definition at line 2051 of file t2m_no_flex.c.

10.29.2.2 void yy_delete_buffer (YY_BUFFER_STATE *b*)

Destroy the buffer.

Parameters

b	a buffer created with yy_create_buffer()

Definition at line 2080 of file t2m_no_flex.c.

10.29.2.3 void yy_flush_buffer (YY_BUFFER_STATE *b*)

Discard all buffered characters.

On the next scan, YY_INPUT will be called.

Parameters

b	the buffer state to be flushed, usually `YY_CURRENT_BUFFER`.

Definition at line 2127 of file t2m_no_flex.c.

10.29.2.4 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

base	the character buffer
size	the size in bytes of the character buffer

Returns

the newly allocated buffer state object.

Definition at line 2254 of file t2m_no_flex.c.

10.29.2.5 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

yybytes	the byte buffer to scan
_yybytes_len	the number of bytes in the buffer pointed to by *bytes*.

Returns

the newly allocated buffer state object.

Definition at line 2304 of file t2m_no_flex.c.

10.29.2.6 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

yystr	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 2291 of file t2m_no_flex.c.

10.29.2.7 void yy_switch_to_buffer (YY_BUFFER_STATE *new_buffer*)

Switch to a different input buffer.

Parameters

new_buffer	The new input buffer.

Definition at line 2006 of file t2m_no_flex.c.

10.29.2.8 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 2186 of file t2m_no_flex.c.

10.29.2.9 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

new_buffer	The new state.

Definition at line 2156 of file t2m_no_flex.c.

10.29.2.10 void yyrestart (FILE * *input_file*)

Immediately switch to a different input stream.

Parameters

input_file	A readable stream.

Note

This function does not reset the start condition to `INITIAL`.

Definition at line 1987 of file t2m_no_flex.c.

10.29.2.11 void yyset_in (FILE * *in_str*)

Set the input stream.

This does not discard the current input buffer.

Parameters

in_str	A readable stream.

See also

yy_switch_to_buffer

Definition at line 2421 of file t2m_no_flex.c.

10.29.2.12 void yyset_lineno (int *line_number*)

Set the current line number.

Parameters

line_number	

Definition at line 2409 of file t2m_no_flex.c.

10.29.3 Variable Documentation

10.29.3.1 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 1008 of file t2m_no_flex.c.

10.29.3.2 YY_BUFFER_STATE* yy_buffer_stack = 0 [static]

Stack as an array.

Definition at line 316 of file t2m_no_flex.c.

10.29.3.3 size_t yy_buffer_stack_max = 0 [static]

capacity of stack.

Definition at line 315 of file t2m_no_flex.c.

10.29.3.4 size_t yy_buffer_stack_top = 0 [static]

index of top of stack.

Definition at line 314 of file t2m_no_flex.c.

10.29.3.5 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.

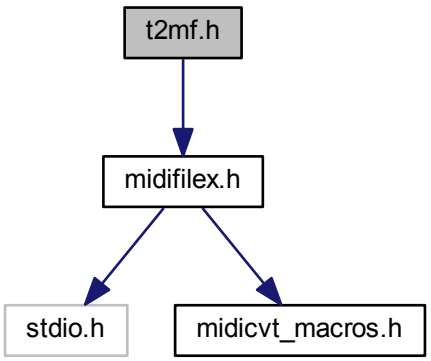
Definition at line 89 of file midicvt_globals.c.

10.30 t2mf.h File Reference

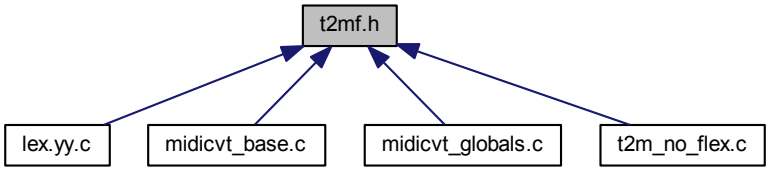
This module provides global variables for the text-to-MIDI conversion portion of libmidifilex.

```
#include <midifilex.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 yyleng 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.

10.30.1 Detailed Description

This module provides global variables for the text-to-MIDI conversion portion of libmidifilex.

Library libmidifilex

Author(s) Chris Ahlstrom and many other authors

Date 2014-04-09

Last Edits 2015-08-22

Version

$Revision$ **License** GNU GPL

10.30.2 Macro Definition Documentation

10.30.2.1 #define YY_TYPEDEF_YY_SIZE_T

Good old CentOS's version of flex declared yyleng 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 80 of file t2mf.h.

10.30.3 Variable Documentation

10.30.3.1 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 950 of file lex.yy.c.

10.30.3.2 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.

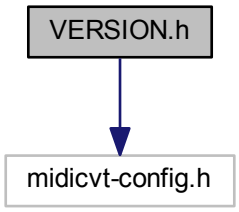
Definition at line 89 of file midicvt_globals.c.

10.31 VERSION.h File Reference

This module provides simpler version information for the libmidifilex library.

```
#include "midicvt-config.h"
```

Include dependency graph for VERSION.h:



10.31.1 Detailed Description

This module provides simpler version information for the libmidifilex library.

Library libmidifilex

Author(s) Chris Ahlstrom

Date 2007-02-28

Last Edits 2016-04-23

Version

$Revision$ **License** GNU GPL

Index