MIDI Model Description Specification 2.0

Contents

History

Version 1.0

Initial release.

Version 2.0

- Added this “History” section.
- Section “MMD Function Requirements”: added decode() function which supersedes and replaces the decode_program() and decode_globals() functions;
- Section “Model Information Attributes”: added documentation of the new **writable_slots** and **notes** attributes;
- Added new section “Tagged Record Format”.

Introduction

A MIDI Model Description (MMD) defines the characteristics of a model of MIDI device, including such as:

- the designer of the device, and the code used in MIDI System Exclusive (SysEx) messages to identify this particular model;
- the method by which a device of this model can be automatically detected, if possible;
- SysEx commands supported by this model;
- procedures for encoding and decoding SysEx data for this model.

Programming Language

MMDs are written in Lua 5.3 programming language. Previous experience with Lua is an asset, but readers experienced in other programming languages should have little difficulty getting acquainted, and will easily acquire proficiency to develop their own MMDs.

Hexadecimal Notation

Lua recognizes integer numbers written in hexadecimal (base 16) using the prefix "0x". For example, the value 19 can be written as "0x13". We use this notation in this document when the number of bits used to represent an integer value is fixed. For example, "0x0C" is the number 12 encoded using no more than 8 bits; "0x05A2" is the number 1,442 encoded on no more than 16 bits.

Strings

In this document we use the expression *octet string* to refer generally to a Lua string whose content is a sequence of 8-bit values that may or may not represent printable characters. Thus, octet strings are generally not printable. We use the expression *character string* for a Lua string that is meant to be printable, that is, consists exclusively of printable ASCII or UNICODE characters.

Tables

Many MMD functions use Lua tables as parameters and returned values. Lua tables exist in two flavors and this document uses the following terminology to differentiate between them when it is important:

- associative array: a type of Lua table that works like a dictionary or phone book: each item in the table consists of a key and a value. The value of an item is obtained by looking up its key in the table;
- list: a type of Lua table whose items are arranged in a sequence and are accessed by giving their relative position in the sequence. The first item is at position #1, the second at #2, and so on.

Learning Lua

A complete tutorial on the Lua programming language is beyond the scope of this document. However, many excellent resources are available:

- http://www.lua.org: the official web site of the Lua language, maintained by its creators at PUC-Rio.
- LERUSALIMSCHY Roberto, "Programming in Lua", Fourth edition: covers version 5.3 of the language.

The reader is also encouraged to examine existing MMD files and using them as templates for creating new ones.

Contents and Structure of an MMD

An MMD is a Lua module that implements one or more of the following functions: **info()**, **globals()**, **dump_program_command()**, **dump_globals_command()**, **decode()**, **load_program_command()** and **load_globals_command()**.

All MMDs are required to implement at least the **info()** function. An MMD may or may not provide implementations for the other functions depending on the capabilities of the model of MIDI device that it describes. For example, the **dump_globals_command()** is required only for a model that implements SysEx commands for remotely initiating SysEx dumps of its global parameters. Devices that do not implement such commands may still be able to transmit their globals, but the corresponding SysEx dumps must then be initiated manually by the user.

The following template outlines the contents of an MMD:

```
local model = {}

function model.info() -- -> model_info
   -- … implementation of the info() function …
end

function model.globals() -- -> category_list
   -- … implementation of the globals() function …
end

function model.dump_program_command( config, slot ) -- -> msg_list, header, max_rsps, slot_list
   -- … implementation of the dump_program_command() function …
end

function model.dump_globals_command( config, globals ) -- -> msg_list, header, max_rsps
   -- … implementation of the dump_globals_command() function …
end

function model.decode( msg_list ) -- -> tagged_record_list
   -- … implementation of the decode () function …
end -- model.decode_program()

function model.load_program_command( config, record_list, slot ) -- -> msg_list
   -- … implementation of the load_program_command() function …
end

function model.load_globals_command( config, globals, record_list ) -- -> msg_list
   -- … implementation of the load_globals_command() function …
end -- model.load_globals_command()

return model
```

The requirements for each function are detailed in section “MMD Function Requirements”.

Definitions

This document uses the following terminology:

SysEx Data

Any information that is stored within a MIDI device and can be exported in its own proprietary format out of the device via SysEx messaging. SysEx data can be restored only to the same device or another device of the same model.

SysEx Command

A SysEx command is a SysEx message that can be sent to a MIDI device and which will cause the device to perform a model-specific action, usually transmitting SysEx data that is stored within the device.

SysEx Dump

A sequence of SysEx messages that contain SysEx data from a MIDI device.

Device Inquiry

Refers to the MIDI device discovery protocol specified in the MIDI v1.0 specification. This mechanism consists of two standard, non-real time system messages denoted as IDENTITY REQUEST and IDENTITY REPLY. A computer or controller may transmit an IDENTITY REQUEST message to connected devices, and each device is then expected to respond with an IDENTITY REPLY message. The IDENTITY REPLY message identifies the model of the device, and may contain other information that can then be used to uniquely address the device in subsequent communications. Refer to the MIDI v1.0 specification for more information.

Unit Number

Sometimes referred to as “global MIDI channel” or “system identifier”. Unit numbers are used to differentiate between multiple devices of the same model when they are connected to the same MIDI output port by way of a MIDI splitter, or daisy-chaining through a 5-pin DIN “MIDI thru” port.

For models that support the MIDI device inquiry protocol, this is the value that appears in the third byte of the device's IDENTITY REPLY message. The unit number is encoded as a 7-bit integer value in range 0-127 (representing unit numbers #1 to #128). A model typically supports only a subset of the available range (#1 to #32 for example).

Program

For the purpose of this specification, SysEx data from a MIDI device that describes a configuration of the device for a particular performance. MIDI device manufacturers often use the terms “patch”, “preset” or “performance” to refer to the same concept.

A program might contain the settings for a synthesizer to produce a particular sound. For a drum machine, a program might be a single beat pattern or a sequence of beat patterns for a complete song. For an effects processor, a program might describe a particular effect such a reverb or chorus.

Active Program

The program that is currently selected in a MIDI device.

Edit Buffer

The edit buffer is the memory in the MIDI device that holds the active program. For most devices, this memory is volatile: its content is erased when the device is powered off.

Stored Program

Many MIDI devices include an amount of persistent memory that is used to store programs that remain in memory when the device is powered off. The user may create a program in the edit buffer and then save it to a persistent memory slot so that the program can be recalled into the edit buffer later. The persistent memory is divided into a number of slots each capable of holding a complete program. To help organize the programs, the slots are also often grouped into banks. For example, a device might comprise four banks of 64 slots each.

An MMD assigns a unique number to each stored program slot of its model, starting at 1. If the slots are grouped into multiple banks, the first slot of each successive bank just takes the next available slot number following the previous bank. For example, for a device that has 4 banks of 64 programs, the slots of the second bank are numbered 65 to 128 inclusive, the slots of the third bank 129 to 192, and so on.

Globals

Globals are settings and other data stored in a MIDI device that the device keeps separate from the programs, and that apply at all times regardless of the active program. Examples of such settings are:

- the MIDI channel that the device responds to when receiving voice channel messages (e.g., NOTE ON/OFF, CONTROL CHANGE, etc.);
- for a keyboard, whether the aftertouch function is enabled;
- user-defined tones and sound samples that are used in programs to make layered sounds;
- sequencer or chord patterns;
- etc.

Globals are sometimes organized into separate categories that can be dumped and restored individually from/to the device. In MMDs, each category is given a unique name that is then used in MMD functions to indicate which is the category to dump or restore.

The MIDI Functions Library

The MIDI functions library is an extension to the Lua language which is available in the context of MMDs. The functions are used to construct, display and manipulate MIDI messages.

midi.octets_to_hex()

Synopsis

midi.octets_to_hex(octets) -> hex

Description

Given an octet string, this function returns a character string with a textual representation of the numerical value of each byte in the octet string. In the returned character string, each byte is represented by 2 hexadecimal digits, and adjacent byte values are separated by a single whitespace character, for example "04 12 5A B4".

Parameters:

octets: the octet string with the sequence of bytes to convert to hexadecimal representation.

Returns:

hex: hexadecimal representation of the **octets** parameter.

midi.hex_to_octets(hex)

Synopsis

midi.hex_to_octets(hex) -> octets

Description

Given a character string containing hexadecimal digits, this function converts each adjacent pair of digits into an 8-bit value, and returns the resulting byte sequence in a new octet string. Whitespaces are ignored. This function performs the opposite operation from **midi.octets_to_hex()**.

Parameters:

hex: hexadecimal representation of an octet string.

Returns:

octets: an octet string with the sequence of bytes represented by the **hex** parameter.

midi.pack()

Synopsis

midi.pack(unpacked) -> packed

Description

This function packs the bytes in the given octet string into a sequence of 7-bit values suitable for transmission in a MIDI message. This type of encoding is used by many MIDI devices to carry arbitrary binary data in SysEx messages. The function returns a new octet string.

This conversion to packed MSB format consists of stripping the most significant bit of each byte in the given octet string, and storing it into an additional 7-bit MSB word. The MSB word thus contains the most significant bits of the next group of 7 bytes as illustrated below:

Unpacked	Packed
`Byte#          bits`	`Word#          bits`
`1:     A7 A6 A5 A4 A3 A2 A1 A0`	`1:      0 G7 F7 E7 D7 C7 B7 A7 (MSB word)`
`2:     B7 B6 B5 B4 B3 B2 B1 B0`	`2:      0 A6 A5 A4 A3 A2 A1 A0`
`3:     C7 C6 C5 C4 C3 C2 C1 C0`	`3:      0 B6 B5 B4 B3 B2 B1 B0`
`4:     D7 D6 D5 D4 D3 D2 D1 D0`	`4:      0 C6 C5 C4 C3 C2 C1 C0`
`5:     E7 E6 E5 E4 E3 E2 E1 E0`	`5:      0 D6 D5 D4 D3 D2 D1 D0`
`6:     F7 F6 F5 F4 F3 F2 F1 F0`	`6:      0 E6 E5 E4 E3 E2 E1 E0`
`7:     G7 G6 G5 G4 G3 G2 G1 G0`	`7:      0 F6 F5 F4 F3 F2 F1 F0`
	`8:      0 G6 G5 G4 G3 G2 G1 G0`

If the length of the given octet string to encode is not an exact multiple of 7, the extra bytes are encoded similarly, leaving unused bits in the MSB word, for example:

Unpacked	Packed
`Byte#          bits`	`Word#            bits`
`1:     A7 A6 A5 A4 A3 A2 A1 A0`	`1:      0  0  0  0  0 C7 B7 A7 (MSB word)`
`2:     B7 B6 B5 B4 B3 B2 B1 B0`	`2:      0 A6 A5 A4 A3 A2 A1 A0`
`3:     C7 C6 C5 C4 C3 C2 C1 C0`	`3:      0 B6 B5 B4 B3 B2 B1 B0`
	`4:      0 C6 C5 C4 C3 C2 C1 C0`

Parameters:

unpacked: the octet string to pack.

Returns:

packed: the contents of the **unpacked** parameter in packed MSB word format.

midi.unpack()

Synopsis

midi.unpack(packed) -> unpacked

Description

Given an octet string in packed MSB word format, unpacks it and returns a new octet string containing the result. This function performs the opposite conversion from **midi.pack()**.

Parameters

packed: octet string in packed MSB word format.

Returns

unpacked: the unpacked contents of the **packed** parameter.

midi.note_off()

Synopsis

midi.note_off(channel, note, velocity) -> octets

Description

Construct a MIDI NOTE OFF message.

Parameters

channel: voice channel number, integer in range 0-15.

note: note to turn off, integer in range 0-127.

velocity: note off velocity, integer in range 0-127.

Returns

octets: an octet string containing the encoded message.

midi.note_on()

Synopsis

midi.note_on(channel, note, velocity) -> octets

Description

Construct a MIDI NOTE ON message.

Parameters

channel: voice channel number, integer in range 0-15.

note: note to turn on, integer in range 0-127.

velocity: note on velocity, integer in range 0-127.

Returns

octets: an octet string containing the encoded message.

midi.poly_key_pressure()

Synopsis

midi.poly_key_pressure(channel, note, amount) -> octets

Description

Construct a MIDI POLYPHONIC KEY PRESSURE message.

Parameters

channel: voice channel number, integer in range 0-15.

note: note to set, integer in range 0-127.

amount: new key pressure amount, integer in range 0-127.

Returns

octets: an octet string containing the encoded message.

midi.control_change()

Synopsis

midi.control_change(channel, control, value) -> octets

Description

Construct a MIDI CONTROL CHANGE message.

Parameters

channel: voice channel number, integer in range 0-15.

control: number of the control to set, integer in range 0-127.

value: new control value, integer in range 0-127.

Returns

octets: an octet string containing the encoded message.

midi.program_change()

Synopsis

midi.program_change(channel, program) -> octets

Description

Construct a MIDI PROGRAM CHANGE message.

Parameters

channel: voice channel number, integer in range 0-15.

program: number of the program to select, integer in range 0-127.

Returns

octets: an octet string containing the encoded message.

midi.channel_pressure()

Synopsis

midi.channel_pressure(channel, amount) -> octets

Description

Construct a MIDI CHANNEL PRESSURE message.

Parameters

channel: voice channel number, integer in range 0-15.

amount: new pressure amount, integer in range 0-127.

Returns

octets: an octet string containing the encoded message.

midi.pitch_bend_change()

Synopsis

midi.pitch_bend_change(channel, amount) -> octets

Description

Construct a MIDI PITCH BEND CHANGE message.

Parameters

channel: voice channel number, integer in range 0-15.

amount: bend amount, integer in range 0-16,383. The value is offset by 8,192: values in range 0-8,191 bend down; values in range 8,193-16,383 bend up.

Returns

octets: an octet string containing the encoded message.

midi.song_position_pointer()

Synopsis

midi.song_position_pointer(beats) -> octets

Description

Construct a MIDI SONG POSITION POINTER message.

Parameters

beats: position to move the pointer to, in number of MIDI beats from the start of the song. Integer in range 0-16383.

Returns

octets: an octet string containing the encoded message.

midi.song_select()

Synopsis

midi.song_select(song) -> octets

Description

Construct a MIDI SONG SELECT message.

Parameters

song: index of the song to select, integer in range 0-127

Returns

octets: an octet string containing the encoded message.

midi.tune_request()

Synopsis

midi.tune_request() -> octets

Description

Construct a MIDI TUNE REQUEST message.

Parameters

None.

Returns

octets: an octet string containing the encoded message.

midi.timing_clock()

Synopsis

midi.timing_clock() -> octets

Description

Construct a MIDI TIMING CLOCK message.

Parameters

None.

Returns

octets: an octet string containing the encoded message.

midi.start()

Synopsis

midi.start() -> octets

Description

Construct a MIDI START message.

Parameters

None.

Returns

octets: an octet string containing the encoded message.

midi.continue()

Synopsis

midi.continue() -> octets

Description

Construct a MIDI CONTINUE message.

Parameters

None.

Returns

octets: an octet string containing the encoded message.

midi.stop()

Synopsis

midi.stop() -> octets

Description

Construct a MIDI STOP message.

Parameters

None.

Returns

octets: an octet string containing the encoded message.

midi.active_sensing()

Synopsis

midi.active_sensing() -> octets

Description

Construct a MIDI ACTIVE SENSING message.

Parameters

None.

Returns

octets: an octet string containing the encoded message.

midi.system_reset()

Synopsis

midi.system_reset() -> octets

Description

Construct a MIDI SYSTEM RESET message.

Parameters

None.

Returns

octets: an octet string containing the encoded message.

midi.data_entry()

Synopsis

midi.data_entry(channel, value) -> octets

Description

Construct the sequence of MIDI CONTROL CHANGE messages to set the currently-selected Registered or Non-Registered Parameter Number (RPN or NRPN) to the given 14-bit value. The most significant 7 bits of the value are transmitted to control number 6; the least significant 7 bits to control number 38.

This function is normally used in combination with “midi.select_rpn()” and “midi.select_nrpn()” to first select the parameter to set. Data entry sequences may then be transmitted to a device repeatedly to set the value of the selected parameter without having to reselect the parameter each time.

Parameters

channel: voice channel number, integer in range 0-15.

value: new value of the selected parameter, integer in range 0-16383.

Returns

octets: an octet string containing the encoded messages.

midi.select_nrpn()

Synopsis

midi.select_nrpn(channel, number) -> octets

Description

Construct the sequence of MIDI CONTROL CHANGE messages to select the given Non-Registered Parameter Number (NRPN). The most significant 7 bits of the parameter number are transmitted to control number 99; the least significant 7 bits to control number 98.

Parameters

channel: voice channel number, integer in range 0-15.

number: NRPN to select, integer in range 0-16383.

Returns

octets: an octet string containing the encoded messages.

midi.set_nrpn()

Synopsis

midi.set_nrpn(channel, number, value) -> octets

Description

Construct the sequence of MIDI CONTROL CHANGE messages to set the given Non-Registered Parameter Number (NRPN) to the given value. This function returns a sequence of four CONTROL CHANGE messages equivalent to the concatenation of the messages constructed by “midi.select_nrpn()” and “midi.data_entry()”.

Parameters

channel: voice channel number, integer in range 0-15.

number: NRPN to set, integer in range 0-16383.

value: new value of the selected parameter, integer in range 0-16383.

Returns

octets: an octet string containing the encoded messages.

midi.select_rpn()

Synopsis

midi.select_rpn(channel, number) -> octets

Description

Construct the sequence of MIDI CONTROL CHANGE messages to select the given Registered Parameter Number (RPN). The most significant 7 bits of the parameter number are transmitted to control number 101; the least significant 7 bits to control number 100.

Parameters

channel: voice channel number, integer in range 0-15.

number: RPN to select, integer in range 0-16383.

Returns

octets: an octet string containing the encoded messages.

midi.set_rpn()

Synopsis

midi.set_rpn(channel, number, value) -> octets

Description

Construct the sequence of MIDI CONTROL CHANGE messages to set the given Registered Parameter Number (RPN) to the given value. This function returns a sequence of four CONTROL CHANGE messages equivalent to the concatenation of the messages constructed by “midi.select_rpn()” and “midi.data_entry()”.

Parameters

channel: voice channel number, integer in range 0-15.

number: RPN to set, integer in range 0-16383.

value: new value of the selected parameter, integer in range 0-16383.

Returns

octets: an octet string containing the encoded message.

midi.select_program_bank()

Synopsis

midi.select_program_bank(channel, bank) -> octets

Description

Construct the sequence of MIDI CONTROL CHANGE messages to select the given program bank number. The most significant 7 bits of the parameter number are transmitted to control number 0; the least significant 7 bits to control number 32.

Parameters

channel: voice channel number, integer in range 0-15.

bank: number of the program bank to select, integer in range 0-16383.

Returns

octets: an octet string containing the encoded message.

midi.change_program_bank()

Synopsis

midi.change_program_bank(channel, bank, program) -> octets

Description

Construct the sequence of MIDI messages to select a program from a given bank. This function returns a sequence of four messages equivalent to the concatenation of the messages constructed by “midi.select_program_bank()” and “midi.program_change()”.

Parameters

channel: voice channel number, integer in range 0-15.

bank: number of the program bank to select, integer in range 0-16383.

program: number of the program to select, integer in range 0-127.

Returns

octets: an octet string containing the encoded message.

MMD Function Requirements

This section states the functional requirements for MMD functions. For interoperability, an MMD implementation must satisfy these requirements.

info()

Synopsis

info() -> model_info

Description

This function shall return an associative array **model_info** with the attributes of the model of MIDI device supported by this MMD.

The key for each element in the returned array is the name of a model attribute, and the value is a number or string in accordance with the type of the attribute. See "Model Information Attributes" for the list of model attributes.

The following is an example of an implementation of the info() function:

```
function model.info() -- -> model_info
   return {
      version = 2,
      name = "Roland JV-50",
      icon = "*.png",
      manufacturer = "41",
      family = "4D",
      probe = "scan",
      unit_first = 0,
      unit_last = 31,
      unit_factory = 16 }
end -- model.info()
```

Parameters

None.

Returns

model_info: an associative array with the list of attributes for this model. See "Model Information Attributes" for more information.

globals()

Synopsis

globals() -> category_list

Description

Returns the names of the globals categories that are available to retrieve from this model. Also see "**dump_globals_command()**" for more information.

Parameters

None.

Returns

- **category_list**: a list of character strings, each the name of a globals category supported by this MMD.

dump_program_command()

Synopsis

dump_program_command(config[, slot]) -> msg_list, header[, max_rsps[, slots]]

Description

Constructs and returns a list of SysEx messages intended to command a device of this model and with the given configuration to transmit the requested program.

In many models, the program is divided into many sections each of which must be queried separately using a different SysEx command in order to retrieve the entire program. This function shall construct and return the list of all SysEx commands needed to retrieve the entire program from the device.

Parameters

- **config**: an associative array with the configuration of the destination device for the command. See the "Device Configuration" section.
- **slot**: optional positive integer identifying the persistent memory slot that contains the program to retrieve from the device. If omitted, **nil** or 0, the function shall return the list of SysEx commands to retrieve the active program from the device's edit buffer. See “Stored Program” for more information about program slot numbers.

Returns

This function shall return **nil** if the supplied device configuration is invalid or the given slot number is outside the range supported by this MMD. Otherwise:

- **msg_list**: list of octet strings, each a SysEx message that encodes a command intended to be transmitted to the device;
- **header**: an octet string containing the SysEx header that is expected to prefix SysEx messages received from the device in response to each command. Messages received from the device that do not start with this header will be taken as invalid and rejected;
- **max_rsps**: an optional positive integer that indicates the expected maximum number of messages that will be sent by the device in response to **each** SysEx command in **msg_list**. If this value is not returned, **nil** or zero ('0'), each response from the device will be deemed complete when the device transmits no additional message for a duration that exceeds the

command timeout for this model. See the description of the “**timeout**” model attribute for more information;

- **slots**: an optional character string with the list of program slots that the device will transmit in response to the command. This value shall be returned when devices of this model are unable to transmit just the SysEx data of the individual requested program, but only a group that includes the requested program (in particular, many devices are unable to dump individual stored programs, only a single dump that includes every stored program). The list is a comma-separated sequence of slot numbers, for example **"64,65,66,67,68,80,81,82"**. Sequences of consecutive numbers can be abbreviated as a range using a dash (minus sign), for example **"64-68,80-82"**.

dump_globals_command()

Synopsis

dump_globals_command(config, globals) -> msg_list, header[, max_rsps]

Description

Constructs and returns a list of SysEx messages that will command a device of this model with the given configuration to transmit a SysEx dump of the globals named in argument.

Parameters

- **config**: an associative array with the configuration of the device that is the target of the command. See the "Device Configuration" section;
- **globals**: a character string with the name of the globals to retrieve. This should be one of the names in the list returned by the globals() function. See “globals()” for more information.

Returns

This function shall return **nil** if the supplied device configuration is invalid or the **globals** parameter does not name a globals category supported by this MMD. Otherwise:

- **header**: an octet string with the header that is expected to prefix SysEx data messages from the device. Messages received from the device that do not start with this header will be taken as invalid and rejected;
- **max_rsps**: an optional positive integer that indicates the expected maximum number of SysEx data messages that will be sent by the device following <u>**each**</u> SysEx command in **msg_list**. If this value is not returned, **nil** or zero ('0'), each response from the device will be deemed complete when the device transmits no additional message for a duration that exceeds the command timeout for this model. See the description of the “**timeout**” model attribute for more information.

decode()

Synopsis

decode(msg_list) -> tagged_record_list

Description

Decode SysEx data messages and return a list of tagged records with the data items extracted from the messages. Ignore unrecognized SysEx messages intended for a different model, or that contain no data that can be restored to a device of this model.

Parameters

- **msg_list**: a list of octet strings, each a SysEx data message to decode.

Returns

This function shall return **nil** if supplied with an invalid **msg_list** argument. Otherwise:

- **tagged_record_list**: a list of tagged records describing the data items extracted from the given message list. See "Tagged Record Format" for more information.

load_program_command()

Synopsis

load_program_command(config, records[, slot]) -> msg_list

Description

Construct the list of MIDI messages (SysEx or otherwise) necessary to reload a program onto a MIDI device of this model with the given configuration.

Parameters

- **config**: an associative array with the configuration of the device that is the target of the command to construct. See the "Device Configuration" section;
- **records**: a list of octet strings, each the value of a **data** record previously returned by the "**decode()**" MMD function, and altogether representing the program data item to encode. The format and number of records is arbitrary and specific to the MMD implementation. Also see "Tagged Record Format";
- **slot**: the number of the destination slot for the program. If omitted or zero ('0'), the function shall construct the command to load the program into the device's edit buffer.

Returns

This function shall return **nil** if the supplied device configuration is invalid, the given data records are incompatible with devices supported by this MMD, or the given slot number is outside the range supported by this MMD. Otherwise:

- **msg_list**: a list of octet strings, each a MIDI message (SysEx or otherwise), such that when these messages are transmitted to the device, the program will be restored into the requested slot in the device.

load_globals_command()

Synopsis

load_globals_command(config, globals, records) -> msg_list

Description

Construct the list of MIDI messages necessary to reload the globals of a given category onto a MIDI device of this model with the given configuration.

Parameters

- **config**: an associative array with the configuration of the device that sent the messages. See the "Device Configuration" section;
- **globals**: A character string with the name of the globals category to load;
- **records**: a list of octet strings, each the value of a **data** record previously returned by the "**decode()**" MMD function and altogether representing the globals data item of the category to restore to the device. The format and number of records is arbitrary and specific to the MMD implementation. Also see "Tagged Record Format".

Returns

This function shall return **nil** if the supplied device configuration is invalid, the **globals** argument is not the name of a globals category for this model, or the given data records are incompatible with devices supported by this MMD. Otherwise:

- **msg_list**: a list of octet strings, each a MIDI message (of any type, not necessarily SysEx), such that when these messages are transmitted to the device, the globals data supplied in argument will be restored onto the device.

Device Configuration Attributes

Many of the functions in an MMD require a device configuration table in argument. This table is an associative array that provides the details of the particular device that the requested operation is intended for. Each entry in the configuration table gives the value of a particular configuration attribute.

This section documents the supported device configuration attributes.

unit

Type

Integer in range 0-127 (‘0’ represents unit #1).

Description

The MIDI unit number assigned to this device. See “Unit Number” for more information.

Optional

Required for models that use unit numbers. Ignored for models that do not.

Default

None.

input

Type

Character string.

Description

The name of the MIDI input port used to receive messages from this device.

Optional

No.

Default

None.

output

Type

Character string.

Description

The name of the MIDI output port used to transmit messages to this device.

Optional

No.

Default

None.

Model Information Attributes

The model information table for an MMD contains a combination of the attributes described in this section. While most attributes are optional, some are mandatory and must be included in the model information.

Most optional attributes have a default value, and applications will use the default value if the attribute is omitted from the model information table.

specification

Type

Positive integer.

Description

The version of the MMD specification that this MMD conforms to. This document covers MMD version 2.

Optional

No.

Default

None. Mandatory attribute.

name

Type

Character string.

Description

The name of the model. The name may contain any printable character including spaces and punctuation marks.

Optional

No.

Default

None. Mandatory attribute.

source

Type

Character string.

Description

Contact information for the author and maintainer of this MMD.

Optional

Yes.

Default

None.

version

Type

Character string.

Description

The version of this MMD. This string should be in the conventional *x.y* notation (for example “2.1”). The release date of the MMD is an acceptable alternative (for example “2021.07.24”).

Optional

Yes.

Default

None.

icon

Type

Character string.

Description

Optional path name of a file that contains an image to serve as the icon for this model. This path is relative to the directory that contains the MMD file. If the file name portion of the path is a single asterisk ("*"), it will be substituted with the file name of the MMD to make the icon file name. Either DOS/Windows- or UNIX-style separators can be used in the path name, taking care to escape backslash characters if necessary. For example "../icons/*.png" is equivalent to "..\\icons\\*.png".

Common image file formats are supported. Portable Network Graphics (PNG) with 24 bit per pixel for color is the recommended format. Applications may scale the image to fit in available space on screen. The image should have a 1:1 aspect ratio (as many pixels vertically as horizontally) to avoid distortion during scaling. A size of 256x256 pixels is recommended.

Optional

Yes.

Default

None.

manufacturer

Type

Hexadecimal digit string.

Description

The code that is assigned to the manufacturer of this model by the MIDI Manufacturers Association (MMA) and is used in SysEx messages to uniquely identify the manufacturer.

Two types of manufacturer codes are issued by the MMA:

- short: exactly one non-zero 7-bit word; or
- extended: exactly three 7-bit words where the first word is zero.

Code words are expressed in hexadecimal. Extended code words are separated by whitespaces. For example "1B" is an example of a short manufacturer code; "00 31 4A" is an example of an extended manufacturer code.

Optional

No.

Default

None. Mandatory attribute.

family

Type

Hexadecimal digit string.

Description

A 14-bit code (two 7-bit words) that is assigned by the manufacturer and identifies the product family of this device in MIDI IDENTITY REPLY messages. The code must be expressed using two hexadecimal digits per word, least significant word first. If the most significant word is zero, it can be omitted. For example, "4D" or "2A 01".

Optional

No.

Default

None. Mandatory attribute.

member

Type

Description

Additional 14-bit code that is optionally assigned by the manufacturer to distinguish between multiple models from the same family. Encoded in the same manner as the family code (see “family”).

Optional

Yes.

Default

“00”

probe

Type

Character string.

Description

An optional character string with the name of the recommended method to automatically discover devices of this model. Value must be one of the following:

- "standard": use the standard MIDI device inquiry protocol (IDENTITY REQUEST/REPLY messages);
- "scan": attempt to detect devices of this model by issuing a model-specific query message on every MIDI output port and listening for a model-specific reply;
- “none”: there is no method to probe for devices for this model. They cannot be auto-detected and users must add such devices manually.

Optional

Yes.

Default

“standard”

unit_first, unit_last, unit_factory

Type

Positive integer in range 0-127.

Description

Specifying **at least one** of the unit_first, unit_last or unit_factory attributes indicates that this model of MIDI devices uses unit numbers in SysEx messages. See “Unit Number” for more information.

- **unit_first** is the first unit number in the range supported by the device;
- **unit_last** is the last unit number in the range supported by the device;
- **unit_factory**: the default unit number assigned to devices of this model during manufacturing.

If **unit_first** is greater than **unit_last**, then the given range is taken to include all numbers from **unit_first** to 127 (inclusive) and then from 0 to **unit_last** (inclusive). For example, the combination **unit_first = 127, unit_last = 15** means that this model supports unit numbers 0 through 15 inclusive, and 127. This makes it possible to express a range that includes the value 127 which is used as a wildcard by some models. A device set with unit number 127 then accepts SysEx messages intended for any unit of its model.

If **unit_factory** is set to a value outside the given range, it is ignored and **unit_first** is used as the factory number instead.

Many modern devices that support MIDI-over-USB no longer use this mechanism and their proprietary SysEx messages don't include a unit number. These devices cannot be uniquely addressed when they are daisy-chained via 5-pin DIN MIDI. Information tables for models that still do shall specify at least one of **unit_first**, **unit_last** or **unit_factory**.

Optional

Yes. At least one of **unit_first**, **unit_last** or **unit_factory** must be given if the model uses unit numbers. Omitting all indicates that this model does not use unit numbers.

Default

unit_first: 0;

unit_last: 127;

unit_factory: value of **unit_first**.

slots

Type

Positive integer.

Description

Number of persistent memory slots for stored programs in devices of this model. If the device encompasses multiple memory banks, this is the total number of slots in all banks combined. See “Stored Program” for more information.

Optional

Yes. Omitting this attribute indicates that this model does not have persistent memory for stored programs.

Default

0 (zero; no stored programs).

writable_slots

Type

Character string.

Description

List of the stored program slots in this model whose content can be overwritten with user-defined programs. This attribute should be specified if the number of slots indicated by the **slots** attribute includes factory program slots that cannot be modified by the user.

This attribute must be formatted as a comma-separated list of the user-modifiable slot numbers. A dash (minus sign) may be used to indicate an inclusive range. For example: “1,4,11-20,64”. Numbers larger than the value of the **slots** attribute are ignored.

Optional

Yes. If omitted, every stored program slot is assumed to be user-modifiable.

Default

The range “1-*N*” where *N* is the value of the **slots** attribute, indicating that all stored program slots are modifiable by the user.

timeout

Type

Positive integer.

Description

The maximum time in milliseconds needed by devices of this model to respond to a SysEx command and begin transmitting a response. That is, the time from the moment that the device receives the last byte of the message containing the command until it transmits the first byte of the message containing its response. The timeout shall not include any time needed to transfer the response over MIDI which depends on the speed of the communication link used, i.e. 5-pin DIN (3125 bytes per second) vs. USB (up to megabytes per second) for example.

Applications will use a timeout value of 10 milliseconds if the given value is less than 10, and a value of 10,000 milliseconds if the given value is greater than 10,000.

Optional

Yes.

Default

1,000 (one second).

notes

Type

Character string.

Description

Usage notes for this MMD, informing the user of any limitations of this MMD or the model of MIDI devices it supports.

The value may contain control characters such as horizontal tab (‘\t’) and newline (‘\n’) to control formatting of the text displayed to the user.

Optional

Yes.

Default

None.

Tagged Record Format

Tagged records are octet strings that contain data extracted from the SysEx messages of a device of the supported model. Each string is formatted as a “*tag*:*value*” pair where *tag* indicates the type of the following *value*. The *tag* and *value* fields shall be separated by a colon character (‘:’) with no intervening white spaces.

Multiple tagged records are normally required and listed in sequence to completely describe a data item. The supported tags are described in the following sections.

program

Indicates the start of a sequence of tagged records for a program data item. The *value* for a **program** tag shall be a positive decimal integer indicating the number of the memory slot where this program was stored in the source MIDI device, or ‘0’ (zero) if the program was sourced from the device’s edit buffer. The *value* shall be less than or equal to the value of the **slots** model attribute returned by the **info()** MMD function. See “Model Information Attributes” for more information.

This record shall be followed by zero or one **name** record, and one or more **data** records, in any combination.

globals

Indicates the start of a sequence of tagged records for a globals data item. The *value* is the name of the globals category, and must be one of the category names returned by the **globals()** MMD function. See “MMD Function Requirements” for more information.

This record shall be followed by one or more **data** records.

name

Specifies the name of the current program data item. A **name** record shall be included following a **program** record if and only if the name of the program can be extracted from the SysEx data that was received from the device. The *value* associated to this tag is a character string.

data

The value for this tag shall be a sequence of octets representing actual data extracted from one or more SysEx messages from the source device. The particular formatting of the value is defined by the MMD implementation, and is opaque to applications.

The values of all the **data** records in a program data item shall together be sufficient for the **load_program_command()** MMD function to construct new MIDI messages (SysEx or otherwise) to restore the program to a device of the same model, with the same or a different unit number, and to the same or a different stored program slot, or the edit buffer.

The values of all the **data** records in a globals data item shall together be sufficient for the **load_program_command()** MMD function to construct new MIDI messages (SysEx or otherwise) to restore the globals to a device of the same model, with the same or a different unit number.

Example

The following is an example of a sequence of tagged records (one per line) that contains two program data items (sourced from the edit buffer and stored program slot #5, respectively), and one globals data item. *<octets>* represents an octet string with data extracted from the SysEx messages from the device.

```
program:0
name:Hello World!
data:<octets>
data:<octets>
program:5
name:Fun at the Park
data:<octets>
data:<octets>
globals:Settings
data:<octets>
```