Smart Multibox Programming Guide

This guide gives you the information you need to write custom scripts for the Smart Multibox

About

Introduction

The Smart Multibox is a 2-input, 4-output MIDI interface with a microprocessor "brain". You can use custom scripts to filter, route and translate incoming MIDI messages and send them to any or all of the MIDI outputs.

Scripts are written in Micropython, which is similar to Python 3, but is optimized for running on small devices. As of this writing, the Smart Multibox uses version 1.24.1 of Micropython.

Smart Multibox Editor

Setup

The Smart Multibox Editor app is used to edit scripts and send them to the Smart Multibox. It's available for these platforms:

- MacOS 12 (Monterey) or newer
- Windows 10 or newer

To use the app, connect the Smart Multibox's USB C connector to your computer, then start up the Smart Multibox Editor. In the upper right corner, it should show "Smart Multibox:26". If it instead reports "-not connected-", check your USB connection and click the dropdown menu (that currently says "-not connected-"). Select "Scan for Devices". It should connect immediately to the Smart Multibox at that point.

If you have more than one Smart Multibox, you can use the Global Settings window to assign a different name and Sysex ID number to each unit. It's important to have different IDs and names on them so that the computer can distinguish between each unit. When more than one unit is physically connected to your PC, the dropdown menu in the upper right will show all of the connected Smart Multiboxes. You can use that menu to select which device the editor is currently connected to.

When setting a Smart Multibox's name and sysex ID, it's best to only have one Smart Multibox connected to your computer. Once you've changed the name and ID, the editor app can tell them apart and you can plug more than one into your computer simultaneously.

Usage

On the left side of the screen, a list of all scripts you have downloaded or written is shown. On the right, the contents of the currently selected script is shown, and you can edit it there.

A new blank script can be created using the "+" button and the currently selected script can be deleted using the "-" button. The "Sort" button sorts the list of scripts by title.

At the bottom of the left column, there's a button titled "Download More Scripts", and this allows you to download any of the scripts we've put on our website. This list will continue to grow as we write more examples or users contribute their scripts.

To send the currently selected script to the Smart Multibox, press the button "Send Script To Device". Pressing "Read Script From Device" will read the current script from the Smart Multibox and save it as a new script in the list.

At the bottom of the screen is a message log. This displays any errors that occur during the running of the script, and also the output of any print() command in the script, as long as the Smart Multibox is connected.

Accessories

TRS Port

The Smart Multibox has a 1/4-inch TRS input, which can be used to connect an expression pedal, a 1-button footswitch or 2-button footswitch, and scripts can act based on changes to the connected accessory.

Please see the Hardware Access Functions and Callbacks sections to find out how to write scripts using these accessories.

Expression Pedal

The Smart Multibox accepts a standard tip-to-wiper expression pedal. This is the most common kind, made by Roland/Boss, Mission Engineering, Fractal Audio, etc.

NOTE: *This is different than all the older RJM Music products. We formerly used the other type of expression pedal, ring-to-wiper, but we're starting to move to the more common format, starting with this product.*

External Switch

The TRS port can also accept a 1 or 2 button footswitch. The footswitch should be a simple momentary or latching short-to-ground switch, with no LED in series with the switch. A 1 button footswitch can use a 1/4-inch TS (mono) cable, but please note that the ring switch will always be considered "on" when using a mono cable. When using a 2-button footswitch, a 1/4-inch TRS cable is required.

Programming Reference

General Guidelines

The Smart Multibox uses Micropython as its scripting language. General information can be found on the Micropython website. Most reference materials that cover Python 3 will also apply.

As of this writing, the Smart Multibox uses Micropython 1.24.1

The Micropython build is fairly minimal, most optional features have not been included, for example, file I/O and threads. In addition to the minimal Micropython configuration, these are the optional features that *have* been included:

- bytearrays
- floating point math
- The **array** module
- The **collections** module *(deque and namedtuple types)*
- The **micropython** module
- The **random** module
- The **struct** module

This section will explain the extensions that have been added to Micropython for the sending, receiving and manipulation of MIDI messages.

Callback Driven Operation

The Smart Multibox is designed to primarily use callback functions. User code should not implement loops that run for a long time or infinitely. Doing so will prevent the Smart Multibox from operating properly. Callbacks are available for a number of events, including incoming MIDI messages, button presses and timer ticks. See the Callbacks section for more details.

MIDI Functions

The Smart Multibox has quite a few functions that allow you to create, send and receive MIDI messages. MIDI messages can also be created manually - they are actually Python bytearrays, and can be manipulated with standard Python functions.

Message Creation Functions

midi_msg_make_1byte(msg_type)

Creates and returns a 1 byte MIDI message with the given message type. Message types can be found in the Constants section.

midi_msg_make_cc(channel, number, value)

Creates and returns a MIDI CC (continuous controller) message with the given MIDI channel (1-16), number (0-127) and value (0-127).

midi_msg_make_chan_pressure(channel, value)

Creates and returns a MIDI channel pressure message with the given MIDI channel (1-16) and value (0-127).

midi_msg_make_note(channel, note, velocity)

Creates and returns a MIDI note message with the given MIDI channel (1-16), note number (0-127) and velocity (0-127). Using velocity of 0 is equivalent to a "note off".

midi_msg_make_pc(channel, number)

Creates and returns a MIDI PC (program change) message with the given MIDI channel (1-16) and number (0-127).

midi_msg_make_pitch_bend(channel, value)

Creates and returns a MIDI pitch bend message with the given MIDI channel (1-16) and value (0-16383).

midi_msg_make_poly_pressure(channel, note, value)

Creates and returns a MIDI polyphonic pressure message with the given MIDI channel (1-16), note number (0-127) and value (0-127).

Message Manipulation Functions

midi_msg_get_channel(midi_msg)

Given a MIDI message, returns the MIDI channel the message is on (1-16). Returns -1 if the MIDI message has no channel associated with it.

midi_msg_set_channel(midi_msg, channel)

Given a MIDI message, sets the MIDI channel of the message (1-16). Raises a ValueError exception if the channel value is not valid, or if the message does not have a specific channel.

midi_msg_get_number(midi_msg)

Given a MIDI message, returns the PC, CC or note number from the message is (0-127). Returns -1 if the MIDI message has no number associated with it.

midi_msg_set_number(midi_msg, number)

Given a MIDI message, sets the PC, CC or Note number of the message (0-127). Raises a ValueError exception if the channel value is not valid, or if the message does not have a number associated with it.

midi_msg_get_type(midi_msg)

Given a MIDI message, returns the MIDI message type, which is the status byte with the MIDI channel value zeroed out (128-250). Constants defined for each message type are defined in the Constants section. Returns -1 if the MIDI message is not valid.

midi_msg_set_type(midi_msg, msg_type)

Given a MIDI message, sets the type of the message (128-250). Constants defined for each message type are defined in the Constants section. The existing MIDI channel of the message, if any, is left unchanged. Raises a ValueError exception if the type value or the MIDI message is not valid.

midi_msg_get_value(midi_msg)

Given a MIDI message, returns the CC, velocity or pressure value of the message (0-127). Returns -1 if the MIDI message has no value associated with it.

midi_msg_set_value(midi_msg, channel)

Given a MIDI message, sets the CC, velocity or pressure value of the message (0-127). Raises a ValueError exception if the value is not valid, or if the message does not have a value associated with it.

Other MIDI Related Functions

midi_allow_running_status(in_port, allow)

If allow is set to True, data bytes received on in_port that have no status byte in front of them will have the most recently received status byte prepended to them, per the Running Status feature found in the MIDI specification. If allow is set to False, any data bytes received without a preceding status byte will be ignored. The default setting is False.

midi_msg_is_valid(msg)

Returns True if the given value is a valid MIDI message, False if not.

midi_route_clear()

Removes all MIDI clock routes

midi_route_clock(in_port, out_port)

Adds a routing connection from in_port to out_port, where MIDI clock, start, stop and continue messages received at in_port are automatically forwarded to out_port. See the Constants section for port definitions.

midi_send(port, msg)

Sends a MIDI message to a specified MIDI port. The MIDI port values are defined in the Constants section. Returns the number of bytes sent, or -1 if an error occurred.

usb_route_input(port, enable)

Enables or disables routing from a MIDI input to USB. This is the same type of routing used in the USB MIDI mode of the Smart Multibox. For example, messages coming in to MIDI In 1 will appear on a connected computer on Smart Multibox MIDI In 1. By default, all MIDI to USB routes are disabled.

usb_route_output(port, enable)

Enables or disables routing from USB to a MIDI output. This is the same type of routing used in the USB MIDI mode of the Smart Multibox. For example, sending a message to Smart Multibox MIDI Out 1 on a connected computer will result in that message coming out of MIDI Out 1 on the Smart Multibox. By default, all USB to MIDI routes are disabled.

Hardware Access Functions

These functions allow you to directly control aspects of the Smart Multibox hardware

multibox_blink_led()

Turns the LED off for a short period of time (~20msec), then returns it to its previous color. This routine is regulated so that blinks are only allowed to happen once every 40msec. This prevents the LED from being off all the time when this routine is called frequently.

multibox_set_led(color)

Sets the LED to a specific color. Possible color values are LED_OFF, LED_GREEN, LED_ORANGE and LED_RED.

multibox_set_trs_mode(mode)

Sets the operating mode of the TRS port. Valid values are TRS_EXPPEDAL (enable expression pedal mode) and TRS_EXTSWITCH (enable external switch mode). The default at power on is TRS_EXPPEDAL mode.

Callbacks

The Smart Multibox uses callbacks to communicate system events to the user code. Using one of the below functions, you specify another function that should be called when an event happens, then the system will call that function to report an event. On powerup, none of these callbacks are defined.

multibox_set_button_cb(cb_func)

Sets a callback function to be called when the button is pressed or released. The function should be in the following format:

button_cb(pressed)

Where **pressed** is a Boolean value that is True indicating the button is pressed or False if the button is released.

multibox_set_exp_pedal_cb(cb_func)

Sets a callback function to be called when the connected expression pedal moves. The function should be in the following format:

exp_pedal_cb(value)

Where value is the expression pedal's position, between 0 and 255.

multibox_set_ext_switch_cb(cb_func)

Sets a callback function to be called when the connected external switch changes state. The function should be in the following format:

ext_switch_cb(sw_num, pressed)

Where **sw_num** is EXTSWITCH_TIP or EXTSWITCH_RING and **pressed** is a Boolean value that is True indicating the button is pressed or False if the button is released.

multibox_midi_set_receive_cb(cb_func)

Sets a callback function to be called every time a MIDI message is received. The function should be in the following format:

midi_receive_cb(port, msg)

Where **port** is the MIDI port the message was received on (see the Constants section), and **msg** is the contents of the MIDI message.

multibox_set_tick_cb(cb_func)

Sets a callback function to be called every time the system tick happens (every 1msec). The function should be in the following format:

tick_cb()

Constants

Some of the Smart Multibox functions take special constants as input parameters. These are defined below.

Expression Pedal Constants

# Minimum and maximum values that are sent to the expression pedal callback
EXPPEDAL_MIN = 0
EXPPEDAL_MAX = 127

LED Constants

# Possible LED colors
LED_OFF = 0
LED_RED = 1
LED_GREEN = 2
LED_ORANGE = 3

MIDI Constants

Message Types

MIDI_NOTE_OFF = 0x80
MIDI_NOTE_ON = 0x90
MIDI_POLY_PRESSURE = 0xA0
MIDI_CC = 0xB0
MIDI_PC = 0xC0
MIDI_CHAN_PRESSURE = 0xD0
MIDI_PITCH_BEND = 0xE0
MIDI_SYSEX_START = 0xF0
MIDI_TIME_FRAME = 0xF1
MIDI_SONG_POS = 0xF2
MIDI_SONG_SEL = 0xF3
MIDI_TUNE_REQ = 0xF6
MIDI_SYSEX_END = 0xF7
MIDI_CLOCK = 0xF8
MIDI_START = 0xFA,
MIDI_CONTINUE = 0xFB
MIDI_STOP = 0xFC
MIDI_ACTIVE_SENSE = 0xFE
MIDI_RESET = 0xFF

Other Values

MIDI_MIN_VAL = 0
MIDI_MAX_VAL = 127

Port ID Numbers

MIDI_IN_1 = 0x0
MIDI_IN_2 = 0x1
MIDI_IN_3 = 0x2 # Bidirectional input on MIDI_OUT_1
MIDI_IN_USB = 0x8 # Used when the connected computer sends to the "SMB Internal" USB MIDI port
MIDI_OUT_1 = 0x10
MIDI_OUT_2 = 0x11
MIDI_OUT_3 = 0x12
MIDI_OUT_4 = 0x13
MIDI_OUT_5 = 0x14 # Bidirectional output on MIDI_IN_1
MIDI_OUT_USB = 0x18 # Used to send to the connected computer using the "SMB Internal" USB MIDI port

TRS Port Constants

# External switch IDs used for a 2 button external switch. 1 button switches always use EXTSWITCH_TIP
EXTSWITCH_TIP = 0
EXTSWITCH_RING = 1
# TRS port modes
TRS_EXPPEDAL = 0
TRS_EXTSWITCH = 1