MIDX-20

DUAL USB MIDI Host

Class Compliant USB MIDI devices
Roland/BOSS devices
Fender Mustang™ Amplifiers

MIDI /SYSEX

IMPLEMENATION

Firmware version 1.0
Rev. 2016-09-09

Table of Contents

Read Firmware Version 3

Read 32 Bytes from EEPROM 3

Write 32 Bytes to EEPROM 4

Calculating the checksum 4

The EEPROM data structure 5

MIDX-20 MIDI/SYSEX

The MIDX-20 unit responds to a few MIDI SYSEX commands. These commands were implemented to allow the MIDX-20 Assistant PC program to program the unit.

Read Firmware Version

This command is implement to determine the actual firmware version of the MIDX-20, as future versions may have different EEPROM image and functionality. Use this command to verify the version and to verify the connection status.

Request:

F0h 6Fh 10h 02h 11h
[01h] <chksum> F7h

Answer:

F0h 6Fh 10h 02h 12h
[01h
00h 00h 00h 00h
<MajorVerLoNibble>
<MajorVerHiNibble>
<MinorVerLoNibble>
<MinorVerHiNibble>
]
<chksum> 0xF7

Read 32 Bytes from EEPROM

The 256-byte EEPROM of the MIDX-20 holds all the device settings. It can only be read or written to using 32 bytes in each request. See description of the EEPROM structure for further details.

Request:

F0h 6Fh 10h 02h 11h
[0Dh <AdrLoNibble> <AdrHiNibble>] <chksum> F7h

Answer:

F0h 6Fh 10h 02h 12h
[0Dh <AdrLoNibble> <AdrHiNibble>
32 data values split into lo and hi nibble (totally 64 MIDI bytes) in this form:
<DataLoNibble>
<DataHiNibble>
]
<chksum> 0xF7

Write 32 Bytes to EEPROM

Note: To write the full EEPROM structure (full EEPROM is max 256 bytes) you will have to write several blocks of data.

Request:

F0h 6Fh 10h 02h 11h
[0Eh <AdrLoNibble> <AdrHiNibble>
32 data values split into lo and hi nibble (totally 64 MIDI bytes) in this form:
<DataLoNibble>
<DataHiNibble>
] <chksum> F7h

Answer:

F0h 6Fh 10h 02h 12h
[0Eh <AdrLoNibble> <AdrHiNibble>] <chksum> F7h

Calculating the checksum

The checksum need to be calculated for all bytes within the brackets […] showed above.

```
// Calculate Roland style checksum
BYTE SxCalcChecksum(BYTE* pBuf, WORD bytes)
{
   WORD w;
   BYTE* p = pBuf;
   BYTE  chksum = 0;
   for (w=0; w<bytes; w++)
   {
      chksum += p[w];
      if (chksum>128)
         chksum = chksum-128;
   }
   if (chksum>0)
      chksum = 128-chksum;
   chksum &= 0x7F; // Just in case
   return chksum;
}
```

The EEPROM data structure

The checksum need to be calculated for all bytes within the brackets […] showed above.

Note 1:
The member `nSizeOfStruct must` be set to sizeof(`EEPROM_DATA) = 242 bytes`
The member `nPattern must` be set to 66 (decimal)

Note2:
The MIDI bytes (max 24) is actually what is used after image is written to the EEPROM for footswitch press and release. Your final footswitch MIDI data need to be put in the `nFSOnData` and `nFSOffData` members. The members `nFSOnBytes` and `nFSOffBytes` must reflect how many of bytes occupied by these arrays.

As long as the MIDX-20 internal Wizard mode is not ran, the aforementioned buffers will be used when a footswitch is pressed or released. This allow for a multitude of possibilities when programming the device from an external source. Custom MIDI commands at various channels or even SysEx commands may be initiated while pressing or releasing the footswitches.

```
nFSLatched – Will try to unlatch a latched footswitch
nFSInvert  - Will invert 0/1 for the footswitch
nFSRepeat  - make the PC+/PC- auto repeat if hold.

#pragma pack( push, original_settings )
#pragma pack(1)   // Needs one byte packing to be compatible with MIDX-20

typedef struct
{
    BYTE nFSLatched;        // Input foot switch is latched
    BYTE nFSInvert;         // Reverse polarity
    BYTE nFSRepeat;         // Enable repeat mode for PC+/PC-
    BYTE nFSOpMode;         // 0=CCLAT Send bytes for ON and OFF (latched)
                            // 1=CCMOM Send bytes for ON and OFF (momentary)
                            // 2=PC Fixed - (momentary)
                            // 3=PC Program Decrement mode - (momentary)
                            // 4=PC Program Increment mode - (momentary)
                            // 5=Start - (momentary)
                            // 6=Continue - (momentary)
                            // 7=Stop - (momentary)

    BYTE nFS100;             // FS Houndreds    – Used internally by wizard
    BYTE nFS010;             // FS Tens         – Used internally by wizard
    BYTE nFS001;             // FS Ones         – Used internally by wizard

    BYTE nFSOnBytes;        // Number of bytes to send when FS is 'ON'/DN
    BYTE nFSOnData[24];     // MIDI bytes to send when FS is 'ON'/DN

    BYTE nFSOffBytes;       // Number of bytes to send when FS is 'OFF'/UP
    BYTE nFSOffData[24];    // MIDI bytes to send when FS is 'OFF'/UP

}  EEPROM_SWITCH;

typedef struct
{
```

```
    BYTE nCTExprChnl;       // 0-15 Channel for Expression pedal
                            // (or also FS, if using MIDX-20 internal Wizard)
    BYTE nCTExprCC100;      // Expression pedal CC# Houndreds
    BYTE nCTExprCC010;      // Expression pedal CC# Tens
    BYTE nCTExprCC001;      // Expression pedal CC# Ones

    EEPROM_SWITCH fs[2];    // 0=Tip 1=Ring
}  EEPROM_CTRL;

typedef struct
{
    BYTE nSizeOfStruct;     // Number of bytes in this struct
    BYTE bMergeFlag;        // MIDI MERGE 0/1
    BYTE nMIDXMode;         // 0: Only send to device
                            // 1: Both send and receive to a device
                            // 2: Only receive from controller
                            // 3: Receive and send to a controller

    BYTE nMustangChnlLWR;   // 0-15 Channel for lower USB Mustang MIDI Bridge
    BYTE nMustangChnlUPR;   // 0-15 Channel for upper USB Mustang MIDI Bridge
    EEPROM_CTRL ctrl[2];    // 0= CTRL1, 1= CTRL2

    BYTE nPattern;          // Should be 66 if properly set

} EEPROM_DATA;

#pragma pack( pop, original_settings )
```