Nova System: MIDI system exclusive documentation

Question

Please describe the MIDI sys ex documentation for the Nova System.

Answer

Request of Parameters

Currently this documentation only describes the parts relevant to getting and setting parameters.

Request System Parameter Settings

This is equivalent to selecting "Dump System" in the "MIDI SETUP MENU".

SysEx Start	0xF0	
Manufacture ID	0x00 0x20 0x1F	
SysEx ID	0x00	Should be the same value as set in the "MIDI SETUP MENU".
Model ID	0x63	
Command	0x45	Preset Request
Preset Type	0x02	System Parameters
Preset no.	0x00 0x00	Dummy
SysEx End	0xF7	

Request Preset Parameters

This is related to selecting "Dump Bank" in the "MIDI SETUP MENU". The "Dump Bank" sends all the stored user presets only.

SysEx Start	0xF0	
Manufacture ID	0x00 0x20 0x1F	
SysEx ID	0x00	Should be the same value as set in the "MIDI SETUP MENU".
Model ID	0x63	
Command	0x45	Preset Request
Preset Type	0x01	Preset Parameters
Preset no.	0x00 0x00	LSB.MSB (7 b/B) 0: Current values. 1-30: Factory Presets. 31-90: User Presets. 91-118: Variations (note that only a limited part of each preset is used)
SysEx End	0xF7	

Sending a request for an invalid preset number is replied with:

SysEx Start	0xF0	
Non Real Time	0x7E	
SysEx ID	0x00	Should be the same value as set in the "MIDI SETUP MENU".
NAK	0x7E	
0	0x00	
SysEx End	0xF7	

This is also the reply, if a requested user preset is empty.

MIDI-OX Prepared Examples of Commands

Request System Parameters:	`F0 00 20 1F 00 63 45 02 00 00 F7`
Request Current FX Parameter Settings:	`F0 00 20 1F 00 63 45 01 00 00 F7`
Request User Preset 00-1:	`F0 00 20 1F 00 63 45 01 00 20 F7`

Source code excerpt

The following C source code is from the Nova System. It is meant for inspirational purposes only. It illustrates how the information in the MIDI SysEx messages is structured.

Functions

```
//This function handles the WORD4 decoding

static void MoveMidiToInt (int * pDst, int * pSrc, int n)
{
  unsigned int w;

  while (n--) {
    w = (*pSrc++) & 0x7f;
    w |= ((*pSrc++) & 0x7f)<<7;
    w |= ((*pSrc++) & 0x7f)<<14;
    w |= ((*pSrc++) & 0x07)<<21;
    *pDst++=w;
  }
```

```
}


static int CheckPresetDump(void)
{
  int prenum;
  int * pd;
  int chksum=0;
  int n;

  //Preset number part.
  prenum = pMidiExcl->buffer[PRESET_DATA_INDICATOR_SIZE] + (pMidiExcl-
>buffer[PRESET_DATA_INDICATOR_SIZE+1]<<7);
  if (prenum>PRESET_LAST_USER_NO) return 0;
  pMidiExcl->preset.head.num = prenum;

  //Name part
  memcpy (pMidiExcl->preset.head.name,&(pMidiExcl-
>buffer[PRESET_DATA_INDICATOR_SIZE+PRENUMBYTES]),PRENAMELEN);

  // Misc part
  MoveMidiToInt ((int*)&(pMidiExcl->preset.head.misc),
               &(pMidiExcl-
>buffer[PRESET_DATA_INDICATOR_SIZE+PRENUMBYTES+PRENAMELEN]),MISCWORDS);

  //Modifier part
  MoveMidiToInt ((int*)&(pMidiExcl->preset.head.modPar.modParam),
               &(pMidiExcl-
>buffer[PRESET_DATA_INDICATOR_SIZE+PRENUMBYTES+PRENAMELEN+4*MISCWORDS]),
NBMODIFIERS*SIZEOFMODWORDS);

  //Data part
  MoveMidiToInt ((int*)&(pMidiExcl->preset.data.eng[0][0]),
               &(pMidiExcl-
>buffer[PRESET_DATA_INDICATOR_SIZE+PRENUMBYTES+PRENAMELEN+4*
(MISCWORDS+NBMODIFIERS*SIZEOFMODWORDS)]), EFFECTS*EFFECTWORDS);
  //Check checksum
  // n = 4*(NBMODIFIERS+MISCWORDS+EFFECTS*EFFECTWORDS);
  n=4*(MISCWORDS+NBMODIFIERS*SIZEOFMODWORDS+EFFECTS*EFFECTWORDS);// 4*(5+(1*4)+
(9*16))=612
  pd = &(pMidiExcl->buffer[PRESET_DATA_INDICATOR_SIZE+PRENUMBYTES+PRENAMELEN]);
  while (n--) {
    chksum += *pd++;
    chksum &= 0x7f;
  }
  if (chksum!=pMidiExcl-
>buffer[PRESET_DATA_INDICATOR_SIZE+PRENUMBYTES+PRENAMELEN+4*
(NBMODIFIERS*SIZEOFMODWORDS +MISCWORDS+EFFECTS*EFFECTWORDS)]) {
    //Error
    PostMidiEvent(EVT_CHECKSUM);
    return 0;
  }

  //Check preset number and execute store
  if (prenum==0) {
    memcpy(&kernelData.preset,&pMidiExcl->preset,PRESET_SIZE);
    return recallPreset(UPD_CURRENT_PRESET);
  }
  else {
    if(putPreset(prenum, &pMidiExcl->preset))
      return 1;
    else {
      PostMidiEvent(EVT_CANTSTORE);
      return 0;
    }
  }
}

#define MIDI_KERNELDATASIZE_NOT_PRESET_PART_BYTES (PRESET_DATA_INDICATOR_SIZE+4*
(sizeof(kernelData)-sizeof(PRESET))+CHECKSUM_SIZE)

int checkKernelDataDump(void)
{
  int chksum=0;
  int n;
  int * pd;

  n=MIDI_KERNELDATASIZE_NOT_PRESET_PART_BYTES-PRESET_DATA_INDICATOR_SIZE-
CHECKSUM_SIZE;
  pd = &(pMidiExcl->buffer[PRESET_DATA_INDICATOR_SIZE]);
  while (n--)
  {
    chksum += *pd++;
    chksum &= 0x7f;
  }
  if (chksum!=pMidiExcl->buffer[MIDI_KERNELDATASIZE_NOT_PRESET_PART_BYTES-1]) {
    //Error
    return 0;
  }
  else // checksum ok. Copy to kerneldata structure
  {
    MoveMidiToInt ((int*)&kernelData,
                  &(pMidiExcl->buffer[PRESET_DATA_INDICATOR_SIZE]),
(MIDI_KERNELDATASIZE_NOT_PRESET_PART_BYTES-PRESET_DATA_INDICATOR_SIZE-
CHECKSUM_SIZE)/4 );
    return 1;
  }

  return 0; // error
}

#define KERNELDATA_PRESETDATA    0x01 // use this when transmitting the presetpart
of kerneldata
#define KERNELDATA_NOTPRESETDATA 0x02 // use this when transmitting the non-
presetpart of kerneldata


void SendSystemDump (void)
{
  int i,size;
  int checksum=0;

  TCMIDITX_SendTCExclHead(SYXTYPE_PRESETDATA);    // header
  TCMIDITX_SendOneByte(KERNELDATA_NOTPRESETDATA); // dump all non-preset values
from kerneldata

  size = sizeof(kernelData)-sizeof(PRESET); // we expect the preset part of
kerneldata to be placed last!

  checksum = TCMIDITX_SendIntBulk((int*)&(kernelData), size);

  //Send checksum
  TCMIDITX_SendOneByte(checksum&0x7f);

  TCMIDITX_SendExclEnd();
}

int SendPresetDump (int prenum)
{

  char * name;
  int    i;
  int    checksum;

//  TRACE("PresetDump %d\n",prenum);

  if (prenum==0) {
    //get current setting
    pMidiExcl->preset = kernelData.preset;
  } else {
    if (!getPreset(prenum, &pMidiExcl->preset)) return 0;
  }

  //Dump it
  TCMIDITX_SendTCExclHead    (SYXTYPE_PRESETDATA);
  TCMIDITX_SendOneByte       (KERNELDATA_PRESETDATA);
  TCMIDITX_SendTwoBytes      (prenum&0x7f,(prenum>>7)&0x7f);

  //Send the Name
  name = pMidiExcl->preset.head.name;
  for (i=0; i1) {
    TCMIDITX_SendTwoBytes  (name[0]&0x7f,name[1]&0x7f);
  } else if (PRENAMELEN-i>0) {
    TCMIDITX_SendOneByte   (name[0]&0x7f);
  }

  //Misc part
  checksum = TCMIDITX_SendIntBulk       ((int*)&(pMidiExcl->preset.head.misc),
MISCWORDS);

  //Modifier part
  checksum += TCMIDITX_SendIntBulk       ((int*)&(pMidiExcl->preset.head.modPar),
SIZEOFMODWORDS*NBMODIFIERS);

  //Data part
  checksum += TCMIDITX_SendIntBulk       ((int*)&(pMidiExcl->preset.data.eng[0]
[0]), EFFECTS*EFFECTWORDS);


  //Send checksum
  TCMIDITX_SendOneByte   (checksum&0x7f);

  TCMIDITX_SendExclEnd   ();

  return 1;

}


Data Structure for System Parameters

//-------------------------------------------------------------------
// enums used in kerneldata structure
typedef enum { Routing_Serial,Routing_SemiSerial,Routing_Parallel } routings;

typedef enum { Pedal_Expression,Pedal_GSwitch3,Pedal_Exp_VolumeParam } pedalTypes;

typedef enum {
Midi_Channel_Off,Midi_Channel_1,Midi_Channel_2,Midi_Channel_3,Midi_Channel_4,Midi_
Channel_5,Midi_Channel_6,
               Midi_Channel_7,Midi_Channel_8,Midi_Channel_9,Midi_Channel_10,Midi_C
hannel_11,Midi_Channel_12,Midi_Channel_13,
               Midi_Channel_14,Midi_Channel_15,Midi_Channel_16,Midi_Channel_Omni }
midiChannels;

typedef enum { MidiSyncOFF,MidiSyncON,MidiSyncClk } MidiSync;

typedef enum { enTapMaster_Preset,enTapMaster_Global } enTapMasterType;

typedef enum { OffOnTypeOff,OffOnTypeOn } OffOnType;
typedef enum { PrePostTypePre,PrePostTypePost} PrePostType;

typedef enum { NoYesTypeNo,NoYesTypeYes } NoYesType;

typedef enum { enIOinput_AnaLine,enIOinput_AnaDrive,enIOinput_DigSPdif }
enIOinput;

//typedef enum { enLineGuiTypes_Line,enLineGuiTypes_Guitar } enLineGuiTypes;

typedef enum { Dither_Off,Dither_20,Dither_16,Dither_8 } Dither;

typedef enum { ModMstPreset,ModMstMod} ModMaster;

typedef enum { OutputRange_2dB,OutputRange_8dB,OutputRange_14dB,OutputRange_20dB}
OutputRange;

typedef enum { enTunerOutputMute, enTunerOutputOn } enTunerOutput;

typedef enum { TunerModeCoarse, TunerModeFine } tunerModes;
typedef enum { tunerRangeGuitar, tunerRangeBass, tunerRange7strGtr } tunerRanges;

typedef enum { enFswMode_FxEngine, enFswMode_Preset, enFswMode_count }
FswModeType;

//----------------------
// Kernel data structure
//   Stored/restored to/from E2PROM/FLASH at power down/up
//----------------------
// OBS: Data structure must comply with KRNL_ defines and kernelList_inFlash[]
typedef struct {
  //------Kernel DATA------------  Kernel data stored in flash at powerdown
  int                 CurSettingsSignature;   // used when storing/retrieving
current settings
  // ROUTING ------------------------------------------------------------------
---
  routings            routing;     // Current system routing: Serial-Parallel-
Parallel
  int                 bypassAll;   // Bypass All
  // PEDAL --------------------------------------------------------------------
-
  int                 globalVolumeMapMin; // 0% - 100%  KRNL_GLO_VOL_MOD_MIN,
  int                 globalVolumeMapMid; // 0% - 100%  KRNL_GLO_VOL_MOD_MID,
  int                 globalVolumeMapMax; // 0% - 100%  KRNL_GLO_VOL_MOD_MAX,
  pedalTypes          pedalType;   //
  ModMaster           modMaster;   //  Preset, Mod
  // MIDI CC -------------------------------------------------------------------
-
  EXTCTRL_SRC         modTapTempo;
  EXTCTRL_SRC         modCompBypass;
  EXTCTRL_SRC         modDrvBypass;
  EXTCTRL_SRC         modModBypass;
  EXTCTRL_SRC         modDlyBypass;
  EXTCTRL_SRC         modRevBypass;
  EXTCTRL_SRC         modGateBypass;
  EXTCTRL_SRC         modPitchBypass;
  EXTCTRL_SRC         modEQBypass;
  EXTCTRL_SRC         modBoostBypass;
  EXTCTRL_SRC         modExpPedal;
  // MIDI Setup ----------------------------------------------------------------
  midiChannels        midiChnl;    //  Off,1,2,...,16,Omni
  OffOnType           prgChngeIn;  //  Off,On
  OffOnType           prgChngeOut; //  Off,On
  OffOnType           midiClock;   //  Off,On // JOA not used
  int                 midiSysEx;   //  SysExId
  MidiSync            midiSync;    //  Off,On
  // UTIL ----------------------------------------------------------------------
  int                 debug_nNops;    //  (for debug only)
  int                 debug_nParam1;  //  (for debug only)
  int                 debug_nParam2;  //  (for debug only)
  enTapMasterType     tapMaster;   //  Preset - Global
  OffOnType           boostLock;   //  Off,On
  OffOnType           eqLock;      //  Off,On
  OffOnType           routingLock; //  Off,On
  OffOnType           FactBankLock;//  Off,On
  OffOnType           LoudSpeakerFilter;//  Off,On
  int                 viewAngle;   //  0,...,44
  FswModeType         FswMode;     //
  // IO -----------------------------------------------------------------------
  enIOinput           ioInput;     //
  int                 ioClock;     //  {dHwMode_44K1,dHwMode_48K0,dHwMode_DigClk}
  int                 reserved;    //
  NoYesType           dlySpillover;// ??
  Dither              ioDither;    //  Off,20,16,8
  // TAP ----------------------------------------------------------------------
  int                 taptime;     //  Tapped tempo in ms
  // LEVEL ---------------------------------------------------------------------
-
  int                 ioDigitalInLevel;//  -100:+6dB (!?!?)
  int                 InputInLevel;    //  -100dB - 0dB
  OffOnType           LevelModeAdvanced; //  Off,On
  int                 InputRangeLine;  //     0dB - 24dB
  int                 InputRangeInstr; //    -6dB - 18dB
  int                 BoostMax;        //     0dB - 10dB
  OutputRange         rangeOut;        //  2,8,14,20dBu
  int                 globalVolume;    // -100:0dB
  PrePostType         globalVolumePosision;    // Pre - Post
  OffOnType           killDry;         // off/on
  // TUNER ---------------------------------------------------------------------
---
  enTunerOutput       tunerOutput;     //  Mute, on
  int                 tunerRef;        //  420,421,...,460 Hz
  tunerModes          tunerMode;       //  Coarse, Fine
  tunerRanges         tunerRange;      //  Guitar, Bass, 7str Gtr
  OffOnType           sendTuner;       //  off
  // "HIDDEN" PARAMETERS -------------------------------------------------------
-----------------
  int   curPre;                 // <0 if not valid
  int   testSetup;
  int   edited;                 //
  int   signature;              //
  int   pedalImpMode;           // Lo-Z, Hi-Z
  int   pedalCalMin;            // Min
  int   pedalCalMax;            // Max

  // No KRNL_ access:
  _packed char MidiMapIn[MidiMap_prgNo_size+2];    // ((127+2)/3=43)
  _packed char MidiMapOut[NO_OF_USERPRESETS];      // (60/3=20)
  int   iSampleRate;            // An int shadow of kernelStatic.sampleRate.
  int   iMonoSenseEnabled;      // Not used in AC

  //------Preset DATA------ //Preset data stored in FLASH at powerdown
  PRESET  preset;           //

} KERNELDATA, * PKERNELDATA;
```