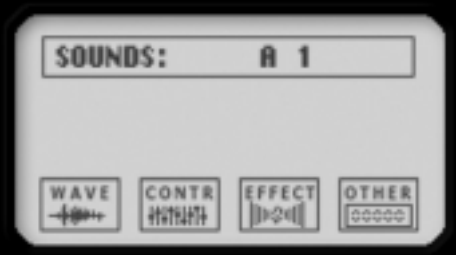

MIDI SYSEX CODE

S2/S3 TURBO

WWW.EQUINOXXX.COM

Tabelle III
MIDI Controller, die die Klangerzeugung beeinflussen

Nicht alle vom S2/S3 empfangenen MIDI Controller beeinflussen die interne Klangerzeugung. Die nachfolgende Tabelle zeigt diejenigen MIDI Controller, die auch die Klangerzeigung betreffen.

(00) BankChange	Zum Aufruf einer Bank mit Sounds (*Hinweis:* Der ursprüngliche MIDI-Standard bezeichnet diesen Controller lediglich als "Controller 0". Die fortschreitende technische Entwicklung der Instrumente führte dazu, daß die Bezeichnung in "Bank Change" geändert wurde).
(01) Modulation	Vibrato. Diese Message wird normalerweise durch ein Wheel erzeugt (beim S2/S3 wird sie im Default durch Wheel 2 erzeugt).
(07) Main Volume	Lautstärke des Midi Channels.
(08) Balance	Balance des Lautstärkeverhältnisses zwischen den beiden Oszillatoren eines "Dual Oscillator" Sounds.
(10) Pan	Bewegung des Sounds im Stereo-Panorama (links–rechts).
(12) Attack	Dauer des ersten Teils der Hüllkurve eines Sounds.
(13) Release	Dauer der Phase "Key On" der Hüllkurve eines Sounds.
(14) Filter 1	Cutoff Frequenz von Filter 1.
(15) Filter 2	Cutoff Frequenz von Filter 2.
(20) Filter 1+2	Cutoff Frequenz von Filter 1 und 2.
(21) Effect 1	Nummer von Effect 1.
(22) Effect 2	Nummer von Effect 2.
(23) Effect 1 Vol	Lautstärke von Effect 1.
(24) Effect 2 Vol	Lautstärke von Effect 2.
(64) Damper	Fußtaster für Sustain (Nachklang). Läßt die Töne weiter-, bzw. ausklingen, bis der Controller den Wert Null erreicht (=Taster wird losgelassen).

(66) Sostenuto	Fußtaster für Sostenuto. Nach Betätigung des Fußpedals wird nur der erste gespielte Ton gehalten bis der Controller den Wert Null erreicht (=Taster wird losgelassen).
(67) Soft Pedal	"Piano"-Pedal. Verringert die Lautstärke des Klanges, solange der Fußtaster betätigt wird.
(90) Rotary S/F	Umschaltung der Geschwindigkeit des Rotary- (Rotations-) Effekts zwischen Slow (langsam) und Fast (schnell).
(123) All Note Off	Unterbricht alle spielenden Töne.

MIDI System Exclusive Code Interface

Introduction

MSECI (Midi System Exclusive Code Interface) is a tool which permits a full operational interface with other devices (Host computer, other GENERALMUSIC device, etc.).

The user can access many resources contained in GENERALMUSIC devices (files, edit structures, global structures).

MSECI permits the programmer to write software for many features, such as Machine Backup, Host Edit Interface, etc..

An expert programmer with a good knowledge of the uses of MSECI can write an excellent software for the complete control of GENERALMUSIC devices.

Communications are achieved using MMA standard similar protocol.

In this document, all software examples are written in C language and the *italic* characters are identifiers defined in include files.

General Overview

MSECI consists of three parts:

1) File Functions

Here you will find the commands for a low level internal file access.

The use of these commands are recommended for expert programmers, because their incorrect use can damage the internal file structure. However, File Functions aren't often useful.

2) Edit Functions

Here you will find the commands to access internal data structures which can be edited.

To write a software program which performs external editing functions, the programmer must be acquainted with the specific internal edit structures, which can differ from one GENERALMUSIC device to another.

3) Device Commands

In this part you will find commands for a general access to the internal system.

The use of these commands is a valid alternative for internal file access and their use is recommended.

This is an example of a general MSECI string:

```
F0  midi system exclusive identifier
2F  GENERAL MUSIC exclusive code
fc  f = function number (0-7)   c=channel(0-F)
ss  ss = subfunction (0-6F, 70-7F reserved)
rc  rc = request channel(0-F)
..
..  data
..
cc  cc=checksum if implemented
F7  end midi system exclusive
```

MSECI also includes a series of HANDSHAKE commands.

A special conversion mode, from 8 bit to 7 bit MIDI data, is used for binary data transmission, in which 7 real bytes are converted to 8 MIDI bytes, known as an OCTECT (fig.1,ex.1,ex.2).

When the above conversion is performed in reverse, the receiving algorithm must be written with care, because the data packet is a finite number of octects and the number of received bytes can be more than the real bytes transmitted, therefore, the receiver must refer to the data header for the exact number of bytes.

The function and subfunction identifiers are defined in the include file "sys_id.h".

***Fig. 1:** 8 to 7 bit conversion*

Example 1: 8 to 7 bit conversion

```
12 45 F7 A3 53 B0 3A — conv8_7 —> 09 22 7F 51 29 58 1D 1E
     (7 bytes)                                (8 bytes)

void conv8_7 ( dest, src )
unsigned char *dest, *src;
{
    int i;
    unsigned char accum = 0;

    for (i=0; i<7; i++)
    {
        accum |= ((*src & 0x1) << i);
        *dest++ = (*src++ >> 1);
    }
    *dest = accum;
}
```

Example 2: 7 to 8 bit conversion

```
void conv7_8 (dest, src)
char *dest, *src;
{
int i;

char bit8 = src [7];
for (i = 7; i > 0; i—, bit8 >>= 1)
*dest++ = (*src++ << 1) + (bit8 & 1);
}
```

Handshake Commands

F0 2F fc ss hh oc F7

f = function number (0-7)

c = channel(0-F)

ss = subfunction (0-6F, 70-7F reserved)

hh = 7F = *ACK* (ok or go on)

7E = *NACK* (not ok, repeat last)

7D = *CANCEL* (abort operation)

7C = *_WAIT* (wait for response)

oc = own channel(0-F)

FILE FUNCTIONS *FILE_F*

FILE DUMP HEADER *F_DHDR*

F0 2F 0c 01 oc <file name> ff <info> <location> cc F7

c = channel (0-F)

oc = own channel (0-F)

file name = 8 chars file name + 3 chars extension

ff = flags

info = 2 octects = 16 bytes of bynary info time/date/length/ instrument ID/file id

location = 1 up 80 chars string NULL terminate

cc = checksum (xor of all bytes from 2F)

After the receiver has received the header, the _WAIT *command is sent back and the checksum test is then performed; if the checksum test fails, the receiver sends back a* NACK *command and the host must repeat the header transmission. If the test is 'ok', the receiver sends back an* ACK *command and the host can then send the next packet.*

When the receiver has performed the <info> with conversion (conv7_8), the result is an 14 byte structure, as shown in ex. 3 below.

Example: 3 INFO structure

```
typedef struct
    {
        short   Time;
        short   Date;
        long    Length;
        char    DeviceClass;
        char    DeviceSubClass;
        char    DeviceRelease;
        char    FileType;
        char    FileFormat;
```

```
    char    FileRelease;
}INFO;
```

FILE DUMP DATA BLOCK *F_DPKT*

F0 2F 0c 02 oc oo <pkt> cc F7
c = channel (0-F)
oc = own channel (0-F)
oo = number of octects
pkt = binary data in octects
cc = checksum (xor of all bytes from 2F)

After the receiver has received the data packet, the _WAIT *command is sent back and the checksum test is then performed; if the checksum test fails, the receiver sends back a* NACK *command and the host must repeat the same data. If the test is 'ok', the receiver converts (conv7_8 ex.2) then sends back an* ACK *command and the host can then send the next packet.*

FILE DUMP REQUEST *F_DREQ*

F0 2F 0c 03 rc <file name> <location> cc F7
c = channel (0-F)
rc = request channel (0-F)
file name = 8 chars file name + 3 chars extension
location = 1 up 80 chars string NULL terminate
cc = checksum (xor of all bytes from 2F)

After the receiver has received the F_DREQ *command, the* _WAIT *command is sent back and the checksum test is then performed; if the checksum test fails, the receiver sends back a* NACK *command and the host must repeat the command. If the test is 'ok', the receiver tries to open the file. If an error occurs during this operation, an* F_ERR *command is sent back, otherwise an* F_DHDR *is sent back.*

FILE DIRECTORY HEADER *DIR_HDR*

F0 2F 0c 00 oc <dir name> ff <info> <location> cc F7
c = channel (0-F)
oc = own channel (0-F)
dir name = 8 chars file name + 3 chars extension
ff = flags
info = 2 octects = 16 bytes of bynary info time/date/length/ instrument ID/file id
location = 1 up 80 chars string NULL terminate
cc = checksum (xor of all bytes from 2F)

This command is used to send directory information.

FILE DIRECTORY REQUEST *DIR_DRQ*

F0 2F 0c 04 rc <dir name> <location> cc F7
c = channel (0-F)
rc = request channel (0-F)
dir name = 8 chars file name + 3 chars extension
location = 1 up 80 chars string NULL terminate
cc = checksum (xor of all bytes from 2F)

FILE ERROR *F_ERR*

F0 2F 0c 7B oc ee cc F7
c = channel (0-F)
oc = own channel (0-F)
ee = error number (0-7F) (ex. 4)

Example 4: FILE ERROR IDENTIFIERS

```
enum {
    No_error,
    Job_invalid, Job_in_use, Operation_invalid,
    Drive_invalid, Drive_no_write,
    Media_invalid, Media_corrupt, Media_protected,
        Media_full,
    Media_not_inserted, Media_not_equal,
    File_not_found, File_open, File_in_write, File_protected,
    File_exists,
    MFH_error
    };
```

EDIT FUNCTIONS *EDIT_F*

EDIT PARAMETER REQUEST *PAR_REQ*

F0 2F 2c 02 rc oo <hdr> cc F7
c = channel (0-F)
rc = request channel (0-F)
oo = octects number = 1
hdr = 1 octect of request header (ex. 5)
cc = checksum (xor of all bytes from 2F)

Example 5: EDIT HEADER (see "edt_ifc.h")

```
typedef struct
{
    uchar family,group,index,type ;
    uchar nr_bytes,user_nr;
} EDT_IFC_HEAD;
```

Below is an example of a request parameter of the slot "MOVE EVENTS" of the sequencer:

```
EDT_ICF_HEAD hdr;
    hdr.family = F_song;      /* see "edt_ifc.h" */
    hdr.group = G_move;       /* see "s2_sequ.h", or other
        device's include file*/
    hdr.index = 0;                          /* not used */
    hdr.type = Group_data;                  /* see ädt_ifc.h"*/
    hdr.nr_bytes = 0;                       /*used only for re-
        sponse */
    hdr.user_nr = 0x7F;       /* not used */
```

EDIT PARAMETER ANSWER *PAR_ASW*

F0 2F 2c 04 rc oo <data> cc F7

c = channel (0-F)

rc = request channel (0-F)

oo = octects number

data = 1st octect contains data header and the next contains data parameters (ex. 5)

cc = checksum (xor of all bytes from 2F)

EDIT PARAMETER SEND+REQUEST *PAR_SND*

F0 2F 2c 03 rc oo <data> cc F7

c = channel (0-F)

rc = request channel (0-F)

oo = octects number

data = 1st octect contains data header and the next contains data parameters (ex. 5)

cc = checksum (xor of all bytes from 2F)

When the device receives this command, after checking for data errors or data range, it sends back the correct data.

EDIT EXECUTE *EXECUTE*

F0 2F 2c 05 rc oo <hdr> cc F7

c = channel (0-F)

rc = request channel (0-F)

oo = octects number = 1

hdr = 1 octect of edit execute header (ex. 5)

cc = checksum (xor of all bytes from 2F)

In many edit slots, the user must perform an execute command to complete the edit.

For example, when we need to modify a sequence with a MOVE operation, after we have edited the MOVE parameter we must perform an EXECUTE command.

EDIT UPDATE *_UPDATE*

F0 2F 2c 06 ff gg oc F7

c = channel (0-F)

ff = family

gg = group

oc = own channel. (0-F)

When the receiver receives an _UPDATE *command, it must perform a* PAR_REQ *command to refresh the data.*

DEVICE COMMAND *DEVICE_CMD*

STATUS REQUEST *STAT_REQUEST*

F0 2F 5c 00 oc F7
c = channel (0-F)
oc = own channel (0-F)

STATUS ANSWER *STAT_ANSWER*

F0 2F 5c 01 oc <data> cc F7
c = channel oc = own channel.
<data> = 3 octets filled with the STAT structure
cc = checksum (xor of all byte from 2F)

Example 6: STAT (see "sys_id.h")

```
typedef struct
{
    unsigned char iClass,iSubClass,iRelease;
    unsigned char dummy;
    long    TotalMem,FreeMem,FreeSampleMem;
    short   ReadyFor; /*bit0=Sound,bit1=Setup,bit2=Song/
            perf*/
    short   ActBankPerf;
} STAT;
```

BANK PERFORMANCE CHANGE *BANK_PERF_CHG*

F0 2F 5c 02 oc bb pp F7
c = channel (0-F)
oc = own channel (0-F)
bb = bank
pp = performance.

PREPARE SOUND ACCESS *PREPARE_soundAccess*

F0 2F 5c 03 oc F7
c = channel (0-F)
oc = own channel.(0-F)

UNPREPARE SOUND ACCESS *UNPREPARE_soundAccess*

F0 2F 5c 04 oc F7
c = channel (0-F)
oc = own channel. (0-F)

PREPARE BANK ACCESS *PREPARE_bankAccess*

F0 2F 5c 05 bb oc F7
c = channel (0-F)
bb = bank
oc = own channel. (0-F)

UNPREPARE BANK
ACCESS *UNPREPARE_bankAccess*

F0 2F 5c 06 bb oc F7
c = channel (0-F)
bb = bank
oc = own channel. (0-F)

UNPREPARE GENERAL
ACCESS *UNPREPARE_generalAccess*

F0 2F 5c 0A oc F7
c = channel (0-F)
oc = own channel. (0-F)

PREPARE EFFECT
ACCESS *PREPARE_effectAccess*

F0 2F 5c 07 oc F7
c = channel (0-F)
oc = own channel. (0-F)

PREPARE STYLE
ACCESS *PREPARE_StyleAccess*

F0 2F 5c 18 bb oc F7
c = channel (0-F)
bb = bank
oc = own channel. (0-F)

UNPREPARE EFFECT
ACCESS *UNPREPARE_effectAcces*

F0 2F 5c 08 oc F7
c = channel (0-F)
oc = own channel. (0-F)

UNPREPARE STYLE
ACCESS *UNPREPARE_StyleAccess*

F0 2F 5c 19 bb oc F7
c = channel (0-F)
bb = bank
oc = own channel. (0-F)

PREPARE GENERAL
ACCESS *PREPARE_generalAccess*

F0 2F 5c 09 oc F7
c = channel (0-F)
oc = own channel. (0-F)

DATA DUMP HEADER *DATA_HEADER*

F0 2F 5c 0C oc dd bb pp <file name1> f1 <inf1> <file name2>
ck F7
c = channel (0-F)
oc = own channel (0-F)

dd = enum {Sound,Sample,Soundmap,Effect1, Effect2,General,Song,Perf}

bb = ASCI Bank number (used only on Song & Perf)

pp = ASCI Perf number (used only on Perf)

<file name1> = 11 byte filename

f1 = file flags

<inf1> = 2 octects (time/date/length/instr.ID/file ID)

<file name2> = 11 byte filename (used only on Perf for song name)

ck = checksum (xor of all byte from 2F)

DELETE *DELETE*

F0 2F 5c 0E oc dd bb pp <file name> ck F7

c = channel (0-F)

oc = own channel (0-F)

dd = enum {Sound, Sample, Soundmap, Effect1, Effect2, General, Song, Perf, Global, StylePerf, Realtime Perf, Riff} (see "sys_id.h")

bb = ASCI Bank number (used only on Song & Perf)

pp = ASCI Perf number (used only on Perf)

<file name > = 11 byte filename

ck = checksum (xor of all byte from 2F)

DATA DUMP *DATA_DUMP*

F0 2F 5c 0D oc oo <pkt> cc F7

c = channel (0-F)

oc = own channel (0-F)

oo = number of octects

pkt = binary data in octects

cc = checksum (xor of all bytes from 2F)

After the receiver has received the data packet, the _WAIT *command is sent back and the checksum test is then performed; if the checksum test fails, the receiver sends back a* NACK *command and the host must repeat the same data. If the test is 'ok', the receiver converts (conv7_8 ex.2) then sends back an* ACK *command and the host can then send the next packet.*

DIRECTORY REQUEST *DIR_REQUEST*

F0 2F 5c 0F oc dd bb pp <file name1> ck F7

c = channel (0-F)

oc = own channel (0-F)

dd = enum {Sound, Sample, Soundmap, Effect1, Effect2, General, Song, Perf, Global, StylePerf, Realtime-Perf, Riff}

bb = ASCI Bank number (used only on Song & Perf)

pp = ASCI Perf number (used only on Perf)

<file name > = 11 byte filename (wildcard)

ck = checksum (xor of all byte from 2F)

DIRECTORY ANSWER *DIR_ANSWER*

F0 2F 5c 10 oc dd bb pp <file name1> f1 <inf1> <file name2> ck F7

c = channel (0-F)

oc = own channel (0-F)

dd = enum {Sound,Sample,Soundmap,Effect1, Effect2,General,Song,Perf,Global, StylePerf,Real-timePerf, Riff}

bb = ASCI Bank number (used only on Song & Perf)

pp = ASCI Perf number (used only on Perf)

<file name1> = 11 byte filename

f1 = file flags

<inf1> = 2 octects (time/date/length/instr.ID/file ID)

<file name2> = 11 byte filename (used only on Perf for song name)

ck = checksum (xor of all byte from 2F)

Must repeat for all the file and directory. At the end a cancel is send for terminate dump.

DATA REQUEST *DATA_REQUEST*

F0 2F 5c 0B oc dd bb pp <file name> ck F7

c = channel (0-F)

oc = own channel (0-F)

dd = enum {Sound,Sample,Soundmap,Effect1, Effect2,General,Song,Perf}

bb = ASCI Bank number (used only on Song & Perf)

pp = ASCI Perf number (used only on Perf)

<file name > = 11 byte filename (used only on Sound & Sample)

ck = checksum (xor of all byte from 2F)

MESSAGE CAPTURE ON *MESSAGECAPTUREON*

F0 2F 5c 11oc F7

c = channel (0-F)

oc = own channel. (0-F)

MESSAGE CAPTURE OFF *MESSAGECAPTUREOFF*

F0 2F 5c 12 oc F7

c = channel (0-F)

oc = own channel. (0-F)

MESSAGE SEND *MESSAGESEND*

F0 2F 5c 13 <message> oc ck F7

c = channel (0-F)

<message> = message string (asci) (see MIOS doc.)

oc = own channel. (0-F)

ck = checksum (xor of all byte from 2F)

MESSAGE ANSWER *MESSAGEANSWER*

F0 2F 5c 11oc rr F7

c = channel (0-F)

oc = own channel. (0-F)

rr = return code (1-5)

ENABLE EDIT UPDATE ***ENABLEEDITUPDATE***

F0 2F 5c 15 oc F7
c = channel (0-F)
oc = own channel. (0-F)

WRITE KEY ***PUT_KEY***

F0 2F 5c 17 mm ll F7
c = channel (0-F)
mm = Msb key
ll = Lsb key

DISABLE EDIT UPDATE ***DISABLEEDITUPDATE***

F0 2F 5c 16 oc F7
c = channel (0-F)
oc = own channel. (0-F)

DEVICE ERROR ***D_ERR***

F0 2F 5c 7B oc ee cc F7
c = channel (0-F)
oc = own channel (0-F)
ee = error number (0-7F) (ex. 7)

Example 7: DEVICE ERROR IDENTIFIERS

```
enum {
    ActiveBankAccess=MFH_error+1,
  IncompatibleObject,BadObjOperation, StyleRecAccess
    };
```