marian

MARC X Control API

Version 1.0
Last updated 28.05.2004

Introduction

This documentation covers the software interface to the Marc X driver, which is useful for third party application software to control the Marc X audio signals. It includes:

- Analog Input Level Control
- Mixer Channel Control
- Sum Channel Control
- Output Volume Control
- Output Signal Routing
- Signal Level Analysis

The software interface is delivered by the 32-Bit Windows™ DLL file *"mxdrvifc.dll"*. Thus, it makes it very easy to use it in any 32-Bit-Windows™ based application software. The documentation includes a C++ language header file (*mxdrvifc.h*) which contains all function prototypes and other declarations like constants and structures. Additional, this documentation comes with the Marc X user manual which helps to understand how the Marc X is working and how the Marc X Manager software visualizes the cards functionality.

In the following, the Marc X Control API is called *MXDRVIFC.*

Common Initialization/Finalization

The application software should first initialize the *MXDRVIFC* by use of the function `mxdrvifc_Init`. Multiple clients of the interface are supported via the same DLL module.

If the software application finishes the use of *MXDRVIFC* it must call `mxdrvifc_Done`.

It is essential for the client software to get the number of installed and active Marc X PCI cards via `mxdrvifc_GetNumCards` because the number of the card is a main identifier for many functions.

The Functions

mxdrvifc_Init

```
BOOL MXCALL mxdrvifc_Init();
// initializes usage of MXDRVIFC
```

mxdrvifc_Done

```
VOID MXCALL mxdrvifc_Done();
// finalizes usage of MXDRVIFC
```

mxdrvifc_GetNumCards

```
LONG MXCALL mxdrvifc_GetNumCards();
// returns the number of installed, active Marc X PCI cards
```

Handling Parameter Values

Every mixer parameter and also every output parameter can be read and set by use of only two functions: `mx_GetValue` and `mx_SetValue`. Each parameter is uniquely identified by a set of ID's – the Card ID, the Channel Type ID, the Channel ID and the Parameter ID.

The Card ID is a simple number beginning from 0 to the number of installed cards – 1. The number of installed cards can be easily determined through the function `mxdrvifc_GetNumCards`. All other ID's are explained in the following sections.

The Functions

mx_GetValue

```
LONG MXCALL mx_GetValue (
     LONG  crdid,
     LONG  typid,
     LONG  chnid,
     LONG  parid,
     PLONG pValue
);
// Gets the current Value of a specific mixer parameter
// Input:
//   crdid  = Card Id
//   typid  = Channel Type Id (see mx_typid defines)
//   chnid  = Channel Id (see mx_chnid defines)
//   parid  = Parameter Id (see mx_parid defines)
//   pValue = Pointer to a LONG variable for the result
// Output:
//   *pValue filled will the current parameter value
//   return < 0 ? Error : Success
```

mx_SetValue

```
LONG MXCALL mx_SetValue (
     LONG  crdid,
     LONG  typid,
     LONG  chnid,
     LONG  parid,
     LONG  Value
);
// Sets the Value of a specific mixer parameter
// Input:
//   crdid  = Card Id
//   typid  = Channel Type Id (see mx_typid defines)
//   chnid  = Channel Id (see mx_chnid defines)
//   parid  = Parameter Id (see mx_parid defines)
//   Value  = New Value for the Parameter
// Output:
//   return < 0 ? Error : Success
```

Mixer Control

Mixer Architecture

The Mixer of the Marc X consists of 16 stereo channels, if the card is operated in ADAT-Mode (see users manual). In Non-ADAT operation mode, the Marc X Mixer consists of 8 stereo channels.

The first half of the mixer channels control the audio signals of the input channels of the card. The second half of the mixer channels control the audio signals of the playback channels of the card. “Playback channels” does not mean output channels. It really means the playback signals which any audio application sends to the drivers software. Which audio signal can be heard at a output depends on the Output Routing (see Output Control).

The signals of all available stereo channels are mixed to three individually stereo sums – AUX1, AUX2 and MASTER.

Mixer Channel Control

Each mixer channel is identified by an ID for its type as mixer channel and an individual channel ID.

Type ID (`mx_typid`)	**Channel ID (`mx_chnid`)**	**Marc X Device**
0	0	Analog Input 1-2
0	1	Analog Input 3-4 (ANX)
0	2	Analog Input 5-6 (ANX)
0	3	Digital Input (S/PDIF)
0	4	ADAT Input 1-2 (ADAT operation mode only)
0	5	ADAT Input 3-4 (ADAT operation mode only)
0	6	ADAT Input 5-6 (ADAT operation mode only)
0	7	ADAT Input 7-8 (ADAT operation mode only)
0	8	Analog Playback 1-2
0	9	Analog Playback 3-4
0	10	Analog Playback 5-6
0	11	Digital Playback
0	12	ADAT Playback 1-2 (ADAT operation mode only)
0	13	ADAT Playback 3-4 (ADAT operation mode only)
0	14	ADAT Playback 5-6 (ADAT operation mode only)
0	15	ADAT Playback 7-8 (ADAT operation mode only)

See the appropriate type ID’s (`mx_typid_xxx`) and channels ID’s (`mx_chnid_xxx`) also in the *mxdrvifc.h*.

Each mixer channel has a number of parameters. The following table shows these parameter in the order the audio signal flows through the channel.

Parameter ID (`mx_parid`)	**Name**	**Value Range**	**Remarks**
0	Source Select	`typedef enum INPROUTE type cast to LONG`	Determines the physical audio connector for the channel. Applicable for Analog Input 1-2 and Digital Input only
1	Gain	`Analog Inputs: 0..0x0A400 == -INF ..+18 dB All other: 0..0x10000 == -INF ..+6 dB`	Signal Level Adjustment.
2	AUX1 Volume	`0..0x10000 == -INF ..+6 dB`	Signal level for mix at AUX1 Bus
3	AUX1 Pre Fader	`1 || 0 == on || off`	Determines whether or not the channel signal is mixed to the AUX1 Bus pre (on) or post (off) the fader section
4	AUX2 Volume	`0..0x10000 == -INF ..+6 dB`	Signal level for mix at AUX2 Bus
5	AUX2 Pre Fader	`1 || 0 == on || off`	Determines whether or not the channel signal is mixed to the AUX2 Bus pre (on) or post (off) the fader section

6	Balance	`Value` / `Left` / `Right`
		`0` / `+6 dB` / `-INF`
		`0x08000` / `0 dB` / `0 dB`
		`0x10000` / `-INF` / `+6 dB`
			Determines the level of the left and right channel at the left and right channel of the Master sum.

7	Mute	`1 || 0 == on || off`	Mutes (On) the channel signal at the AUX1, AUX2 and Master Bus.
8	Solo	`1 || 0 == on || off`	Set the channel signal at the Master Sum to Solo (On).
9	Volume	`0..0x10000 == -INF ..+6 dB`	Signal level for the mix at the Master Sum

Generally, all the values for dB ranges are like PCM sample values. The maximum value is +6 dB. Every *"shift right"* operation of the parameter value decreases the dB value by 6 dB. Thus, the following conversion routines apply:

Parameter Value to dB Value:

```
dBValue = Log2 (ParamValue/0x10000)*6 + 6;
```

dB Value to Parameter Value:

```
ParamValue = Round (Power (2, (dBValue-(-90))/6));
```

The gain of the Analog Inputs is set directly in the AD converter. Thus, another value to dB conversion applies. Only the bits D4..D7 are significant. Please see the conversion table in the file *ak4524-gain.pdf*. Additional, when the gain of the analog inputs is manipulated, this also influences the analog recording level, not only the mixer input level.

Sum Channel Control

Each Sum channel is identified by an ID for its type as sum channel and an individual sum channel ID.

Type ID (`mx_typid`)	Channel ID (`mx_chnid`)	Sum Channel
1	0	Master
1	1	Aux 1
1	2	Aux 2

See the appropriate type ID's (`mx_typid_xxx`) and channels ID's (`mx_chnid_xxx`) also in the *mxdrvifc.h*.

Each sum channel has a parameter for the level adjustment of the audio mix. The following table shows this parameter and it's value range.

Parameter ID (`mx_parid`)	Name	Value Range	Remarks
0	Volume	`0..0x8000 == -INF .. 0 dB`	For AUX1 and AUX2 use. Sets the audio mix level for the left and the right channel to the same value
0	Volume Left	`0..0x8000 == -INF .. 0 dB`	For MASTER use. Sets the audio mix level for the left channel.
1	Volume Right	`0..0x8000 == -INF .. 0 dB`	For MASTER use. Sets the audio mix level for the right channel.

The conversion from dB value to parameter value and back should proceed like described in *"Mixer Channel Control".*

Output Control

Output Architecture

The main task of the output control is the output signal routing and the control of the output signal level. Each channel of the output control is assigned to a physical output of the Marc X. In ADAT operation mode of the card there are 4 stereo output channels, in Non-ADAT mode there are 8 stereo output channels.

Output Channel Control

Each output channel is identified by an ID for its type as output channel and an individual channel ID.

Type ID (`mx_typid`)	**Channel ID (`mx_chnid`)**	**Marc X Device**
2	0	Analog Output 1-2
2	1	Analog Output 3-4 (ANX)
2	2	Analog Output 5-6 (ANX)
2	3	Digital Output (S/PDIF)
2	4	ADAT Output 1-2 (ADAT operation mode only)
2	5	ADAT Output 3-4 (ADAT operation mode only)
2	6	ADAT Output 5-6 (ADAT operation mode only)
2	7	ADAT Output 7-8 (ADAT operation mode only)

See the appropriate type ID's (`mx_typid_xxx`) and channels ID's (`mx_chnid_xxx`) also in the *mxdrvifc.h*.

Each output channel has a number of parameters, shown in this table.

Parameter ID (`mx_parid`)	**Name**	**Value Range**	**Remarks**
0	Source Select	`typedef enum OUTROUTE type cast to LONG`	Determines the source signal which should be used for the output. Any of the available signals can be used – input, playback or sum signals
1	Mute	`1 || 0 == On || Off`	Mutes the Output (on)
2	Volume Left	`0..0x8000 == -INF .. 0 dB`	Controls output level left channel
3	Volume Right	`0..0x8000 == -INF .. 0 dB`	Controls output level right channel

The conversion from dB value to parameter value and back should proceed like described in *"Mixer Channel Control"*.

Audio Level Analysis

The Marc X hardware analyzes the level of every audio signal available. Thus, any client software is able to get the results of the level analysis for its own purposes – mainly for the implementation of level meters.

How it works

The Marc X hardware stores level values in a special part of the card memory. But before a level value is stored, the hardware compares it with the previous stored value which is only overwritten, if the new value is higher than the older. The stored value is reset to zero, if the driver software reads the value from the hardware. This way, the driver software always gets the maximum level which occurred between two read operations.

The driver software does the same for registered client software. It reads the level value from the hardware compares it with the last stored, client related level value and stores it only, if the client level value is less than the new level value. The client related level value is reset to zero, if the client software read its level value.

In this manner the hardware works with every sample, the driver does it approximately every 1 ms and it is up to the client software to get the level value in intervals which are appropriate for the clients purpose.

The implementation of a level meter can be done very easy. The client software could have 2 processes. The first process gets the level and displays it only if the actual displayed level display is less than the new value. The second process simply decreases the level display by a determined dB value in a determined time.

The Functions

Structure MXLEVELINFO

```
typedef struct {
  LONG  Left;         // OUT: Level Left (PRE Gain/Fader)
  LONG  Right;        // OUT: Level Right (PRE Gain/Fader)
  LONG  MxLeft;       // OUT: Level Left (POST Gain/Fader)
  LONG  MxRight;      // OUT: Level Right (POST Gain/Fader)

  LONG  reserved;

  LONG  Code;         // OUT: Success of Request/Get Level operation
  LONG  Requests;     // IN:  Number of Clients who need the Level Analysis
  ULONG Handle;       // OUT: private for the driver
} CHNLEVELINFO, *PCHNLEVELINFO;

typedef struct {
  LONG         CardId;         // IN: ID of target card
  LONG         reserved;       // OUT:
  CHNLEVELINFO Levels[19];     // Level Info for every channel
} MXLEVELINFO, *PMXLEVELINFO;
```

The structure `MXLEVELINFO` with its array of `CHNLEVELINFO` is used to request and get the level values for the different stereo channels. For each stereo channel a `CHNLEVELINFO` structure is assigned in the `Levels` array of `MXLEVELINFO`. Please see the array indexes in the following table and the *"mxdrvifc.h"*.

Index	Channel
0	Analog Input 1-2
1	Analog Input 3-4
2	Analog Input 5-6
3	Digital Input (S/PDIF)
4	ADAT Input 1-2
5	ADAT Input 3-4
6	ADAT Input 5-6
7	ADAT Input 7-8
8	Analog Playback 1-2
9	Analog Playback 3-4
10	Analog Playback 5-6
11	Digital Playback
12	ADAT Playback 1-2
13	ADAT Playback 3-4
14	ADAT Playback 5-6
15	ADAT Playback 7-8
16	MASTER Sum
17	AUX1 Sum
18	AUX2 Sum

mx_RequestLevel

```
LONG MXCALL mx_RequestLevel (PMXLEVELINFO pLevelInfo);
// Switchs Level Anaysis for specific stereo channels on/off
// Input:
//   pLevelInfo = pointer to level information structure
//   pLevelInfo->CardId = ID of target card
//   pLevelInfo->Levels[].Requests = initialized with >0 if Level required
// Output:
//   pLevelInfo->Levels[].Code = Error Code for Request
//   pLevelInfo->Levels[].Handle = Driver specific handle for client channel
//   return < 0 ? Error : Success
```

This function switches the level analysis for specific stereo channels on and off. The driver software uses the information in `pLevelInfo->Levels[].Requests` for the level analysis state of the channel.

The client software can check the success of the operation for each channel by use of `pLevelInfo->Levels[].Code`. A value less than 0 indicates that the request could not be executed. This may occur with the Digital/ADAT input channel only because of inappropriate clock settings (see users manual).

Example: The client software request the levels for the S/PDIF Input and any of the ADAT inputs. With the default clock settings the driver assumes that both digital input clocks are not synchronized. Since the Marc X always works with one clock only, the signal of the S/PDIF input can be evaluated by using the clock of this input as reference. In this case, the ADAT input clock can not be evaluated.

The client software should also observe the information in `pLevelInfo->Levels[].Code` after calling `mx_GetLevel` because it may change because of recording/playback requests from audio recording applications. Read more about this in *mx_GetLevel*.

The client software must use always the identical `MXLEVELINFO` variable for `mx_RequestLevel` and `mx_GetLevel` calls because the driver writes client related information into this structure.

The client software must deactivate the level analysis for all channels before it exits.

mx_GetLevel

```
LONG MXCALL mx_GetLevel (PMXLEVELINFO pLevelInfo);
// Get the Levels for all requested stereo channels
// Input:
//   pLevelInfo = pointer to level information structure
//   pLevelInfo->CardId = ID of target card
// Output:
//   pLevelInfo->Levels[].Code  = Error Code for Get ( < 0 ? Error : Success)
//   pLevelInfo->Levels[].Left  = Level of Signal Left (PRE Gain/Fader)
//   pLevelInfo->Levels[].Right = Level of Signal Right (PRE Gain/Fader)
//   pLevelInfo->Levels[].MxLeft = Level of Signal Left (POST Gain/Fader)
//   pLevelInfo->Levels[].MxLeft = Level of Signal Right (POST Gain/Fader)
//   return < 0 ? Error : Success
```

Using this function, the client software can get the levels of the different channels. The client software gets the level peak values that occurred between this and the previous call. A call of `mx_GetLevel` reset the driver stored peak value to zero.

Before the first call of `mx_GetLevel`, the `MXLEVELINFO` variable must be initialized with `mx_RequestLevel`.

The level value in `Left`, `Right`, `MxLeft`, `MxRight` of the `CHNLEVEKINFO` structure is expressed like a PCM sample value. A conversion to a dB value can be done in the same way like the parameter values are converted (see *Mixer Channel Control*).

The following table contains the value ranges of the level values.

`pLevelInfo->Levels[].Left`	`0..0x08000 == -INF ..  0 dB`
`pLevelInfo->Levels[].Right`	`0..0x08000 == -INF ..  0 dB`
`pLevelInfo->Levels[].MxLeft`	`0..0x10000 == -INF .. +6 dB`
`pLevelInfo->Levels[].MxRight`	`0..0x10000 == -INF .. +6 dB`

The level values in `Left`, `Right` of the `CHNLEVELINFO` structure always represent the level of the source signal. That means for

- Analog Input Channels the level after the gain control
- All other Mixer Channels the level prior the gain control
- Sum Channels the level prior the fader

The level values in `MxLeft`, `MxRight` of the `CHNLEVELINFO` structure always represent the level post fader. This ways it is very easy for the client software to implement a switchable "pre fader metering".

The client software should use `pLevelInfo->Levels[].Code` to evaluate to validity of the level values. This code can contain an error state even if `mx_RequestLevel` not returned an error for that channel. This can be the case for the S/PDIF and ADAT input channels only.

Example: Like mentioned, the Marc X works always with one reference clock only, means also with one sample rate only. If a client software requests a level information for the S/PDIF input, the driver uses the clock of this input. Imagine the digital input runs at 44.1 kHz and a recording software requests a recording from the analog input at 48 kHz. Since recording and playback have always the higher priority towards level analysis, the driver, according its default clock settings, would switch to the internal clock and would set it to 48 kHz. In this case, the level of the digital input could not be evaluated and the `pLevelInfo->Levels[].Code` would signal an error. After the recording has finished, the driver looks at the level request table and ensures the right clock settings for the requested levels. The `pLevelInfo->Levels[].Code` would signal no error again.