

© 06/2006 MARIAN, No part of this documentation may be reproduced or transmitted in any form or by any means without permission in writing from Marian OHG

1. MME and the extension	3
2. Availability of the Extension	3
3. Identification of MARIAN Record and Playback Devices.....	4
4. Extended Functions for Playback Devices	5
4.1 Volume	6
4.1.1 WODM_GETMUTE	6
4.1.2 WODM_SETMUTE	6
4.1.3 WODM_MON_GETVOLUME	6
4.1.4 WODM_MON_SETVOLUME.....	7
4.2 Level Capturing	7
4.2.1 WODM_ENABLELEVEL	7
4.2.2 WODM_DISABLELEVEL	7
4.2.3 WODM_GETLEVEL.....	7
4.3 Routing.....	8
4.3.1 OUTROUTING Formats.....	8
4.3.2 WODM_GETROUTING	10
4.3.3 WODM_SETROUTING.....	11
4.4 Playback Format.....	11
4.4.1 WODM_GETSAMPLERATE.....	11
4.4.2 WODM_GETFORMAT	11
4.4.3 WODM_SETFORMAT	11
5. Extended Functions for Record Devices	13
5.1 Input Level and Monitoring	14
5.1.1 WIDM_GETVOLUME	14
5.1.2 WIDM_SETVOLUME	14
5.1.3 WIDM_GETMUTE.....	15
5.1.4 WIDM_SETMUTE	15
5.1.5 WIDM_MON_GETVOLUME	15
5.1.6 WIDM_MON_SETVOLUME.....	16
5.1.7 WIDM_MON_GETMUTE	16
5.1.8 WIDM_MON_SETMUTE.....	17
5.2 Level Capturing	18
5.2.1 WIDM_ENABLELEVEL	18
5.2.2 WIDM_DISABLELEVEL	18
5.2.3 WIDM_GETLEVEL.....	18
5.3 Routing.....	19
5.3.1 INPROUTING Formats	19
5.3.2 WIDM_GETROUTING	20
5.3.3 WIDM_SETROUTING.....	21
5.4 Record Format.....	22
5.4.1 WIDM_GETSAMPLERATE.....	22
5.4.2 WIDM_GETFORMAT	22
5.4.3 WIDM_SETFORMAT	22
5.5 Pitch.....	22
5.5.1 WIDM_GETPITCH	22
5.5.2 WIDM_SETPITCH	22
6. Monitoring	23

7.	Start/Stop Synchronization.....	24
7.1	Synchronous Playback Start.....	24
7.2	Synchronous Playback Pause und Restart	24
7.3	Synchronous Playback Stop	24
7.4	Synchronous Record Start.....	24
7.5	Synchronous Record Pause and Restart	24
7.6	Synchronous Record Stop	24
7.7	Synchronous Record and Playback Start.....	25
7.8	Synchronous Record and Playback Pause and Restart	25
7.9	Synchronous Record and Playback Stop	25
8.	Special Features Prodif 96 PRO and Prodif PLUS.....	26
9.	Contact the Developers.....	27

1. MME and the extension

The recording and playback of audio material as well as the control of audio devices are possible via standardized driver interfaces by different manufacturers. Traditionally, most of the user programs use the multimedia interfaces of Microsoft Windows™ besides ASIO and EASI. The driver extension described in the following uses the Microsoft Multimedia System too and is based on the so-called MME level. Those are the functions *waveInxxx* and *waveOutxxx*, which are provided by the Module *winmm.dll* (32 Bit) and *mmsystem.dll* (16 Bit) of the Windows Multimedia System. These functions are described at the Microsoft SDK. Additionally MME provides two functions which allow an extension of the standard functionality: *waveInMessage* and *waveOutMessage*. These functions provide the possibility for applications to send user defined messages with two parameters to a WaveIn/Out device. The MARIAN MME Extension is based on these functions and provides the possibility for applications to access additional features of the MARIAN cards.

2. Availability of the Extension

Until March 2006 the MARIAN MME Extension has been implemented for the following products: ARC 88, Prodif 96 PRO, Prodif PLUS, Marc 8 Midi (Siena), Marc 2, Marc 2 PRO, Marc 4 Midi, Marc 4 Digi. The extension is available under the operating systems Windows95/98/ME™ and Windows NT4/2000/XP™.

Please note this when using Windows 2000/XP™:

Beginning from driver version 2.0 the Recording and playback device are supplied via WDM-Audio (kernel) and the software interface to the MME-device is supplied by Microsoft. Thus, these device do not understand the MARIAN-specific messages which can be sent to them. Additionally these devices cannot be identified as part of a MARIAN product (see *waveInGetdevCaps* and *waveOutGetDevCaps*). If any software needs to use the MARIAN MME Extension, the “Classic MME Drivers” have to be activated first. The appropriate option can be found within the “Settings” of driver’s GUI. After the classic MME devices are activated, additional MME devices are available which are able to understand the MARIAN specific messages. These devices can be recognized by the correct manufacturer id and product id and also by a “(MME)” inside the device name. Only these devices can be controlled via the MARIAN MME Extension.

3. Identification of MARIAN Record and Playback Devices

The driver provides a record device for every stereo pair of the input channels available at a MARIAN card. For every stereo pair of the output channels a playback device is provided. Names and features of these devices can be found out via the MME standard functions *waveInGetNumDevs/waveOutGetNumDevs* and *waveInGetDevCaps/waveOutGetDevCaps*.

To identify the wave device as a MARIAN device, the fields *wMid* and *wPid* of the structures *WAVEINCAPS* respectively *WAVEOUTCAPS* can be used. For the appropriate values see the following schedule respectively the header file enclosed.

wMid (Manufacturer Id)

Identifier	Value	Manufacturer
MM_MARIAN	190	MARIAN

wPid (Product Id)

Identifier	Value	Device
MM_MARIAN_ARC88_WAVEIN	5	ARC88 Record Device
MM_MARIAN_ARC88_WAVEOUT	6	ARC88 Playback Device
MM_MARIAN_PD96PRO_ANLG_WAVEOUT	7	Prodif 96 PRO Analog Playback Device
MM_MARIAN_PD96PRO_ANLG_WAVEIN	8	Prodif 96 PRO Analog Record Device
MM_MARIAN_PD96PRO_DIG_WAVEOUT	9	Prodif 96 PRO Digital Playback Device
MM_MARIAN_PD96PRO_DIG_WAVEIN	10	Prodif 96 PRO Digital Record Device
MM_MARIAN_PD96PRO_ADAT_WAVEOUT	11	Prodif 96 PRO Adat Record Device
MM_MARIAN_PD96PRO_ADAT_WAVEIN	12	Prodif 96 PRO Adat Playback Device
MM_MARIAN_PDPLUS_ANLG_WAVEOUT	13	Prodif PLUS Analog Playback Device
MM_MARIAN_PDPLUS_ANLG_WAVEIN	14	Prodif PLUS Analog Record Device
MM_MARIAN_PDPLUS_DIG_WAVEOUT	15	Prodif PLUS Digital Playback Device
MM_MARIAN_PDPLUS_DIG_WAVEIN	16	Prodif PLUS Digital Record Device
MM_MARIAN_PDPLUS_ADAT_WAVEOUT	17	Prodif PLUS Adat Playback Device
MM_MARIAN_PDPLUS_ADAT_WAVEIN	18	Prodif PLUS Adat Record Device
MM_MARIAN_ARC8896_WAVEIN	19	Marc 8 Midi Record Device
MM_MARIAN_ARC8896_WAVEOUT	20	Marc 8 Midi Playback Device
MM_MARIAN_MARC2_ANALOG_WAVEIN	23	Marc 2 Analog Record Device
MM_MARIAN_MARC2_ANALOG_WAVEOUT	24	Marc 2 Analog Playback Device
MM_MARIAN_MARC2_DIGITAL_WAVEIN	25	Marc 2 Digital Record Device
MM_MARIAN_MARC2_DIGITAL_WAVEOUT	26	Marc 2 Digital Playback Device
MM_MARIAN_MARC4M_WAVEIN	31	Marc 4 Midi Record Device
MM_MARIAN_MARC4M_WAVEOUT	32	Marc 4 Midi Playback Device
MM_MARIAN_MARC4D_ANALOG_WAVEIN	37	Marc 4 Digi Analog Record Device
MM_MARIAN_MARC4D_ANALOG_WAVEOUT	38	Marc 4 Digi Analog Playback Device
MM_MARIAN_MARC4D_DIGITAL_WAVEIN	39	Marc 4 Digi Digital Record Device
MM_MARIAN_MARC4D_DIGITAL_WAVEOUT	40	Marc 4 Digi Digital Playback Device
MM_MARIAN_MARC2P_ANALOG_WAVEIN	41	Marc 2 PRO Analog Record Device
MM_MARIAN_MARC2P_ANALOG_WAVEOUT	42	Marc 2 PRO Analog Playback Device
MM_MARIAN_MARC2P_DIGITAL_WAVEIN	43	Marc 2 PRO Digital Record Device
MM_MARIAN_MARC2P_DIGITAL_WAVEOUT	44	Marc 2 PRO Digital Playback Device

4. Extended Functions for Playback Devices

The following new messages have been introduced for the function *WaveOutMessage*. The required message has to be transferred to the parameter `uMessage` of this function. The messages have been implemented to the Headerfile enclosed.

```

DRVM_USER           = 4000h
WODM_GETMUTE        = DRVM_USER + 1;
WODM_SETMUTE        = DRVM_USER + 2;
WODM_MON_GETVOLUME  = DRVM_USER + 3;
WODM_MON_SETVOLUME  = DRVM_USER + 4;
WODM_ENABLELEVEL    = DRVM_USER + 7;
WODM_DISABLELEVEL   = DRVM_USER + 8;
WODM_GETLEVEL       = DRVM_USER + 9;
WODM_GETROUTING     = DRVM_USER + 10;
WODM_SETROUTING     = DRVM_USER + 11;
WODM_GETSAMPLERATE  = DRVM_USER + 12;
WODM_GETFORMAT      = DRVM_USER + 13;
WODM_SETFORMAT      = DRVM_USER + 14;

```

WaveOutMessage is declared as follows:

```

DWORD waveOutMessage (
    HWAVEOUT hwo,
    UINT     uMsg,
    DWORD    dwParam1,
    DWORD    dwParam2
);

```

Parameter:

<code>hwo</code>	Handle of the playback device (return value of <i>waveOutOpen</i>)
<code>uMsg</code>	One of the new defined WODM messages
<code>dwParam1</code>	Parameter 1, dependent on message
<code>dwParam2</code>	Parameter 2, dependent on message

Return: *WaveOutMessage* returns the standard `mmsyserr_xxx` Codes. *WaveOutMessage* returns the general error code `mmsyserr_error`, if a function can't be carried out, because of a hardware specific restriction.

Headerfile: `mmsystem.h`

WIN32 Applications under Windows95/98™

For various functions a pointer will be expected as a parameter. For having the wave driver a 16 Bit Protect Mode Architecture under Windows 95/98™, these pointer will be interpreted in SEG:OFS format. Use the following functions of the WOW32.DLL (part of Windows95/98™), to make such pointer available to your WIN32 Application as well as to the driver:

```

DWORD WINAPI WOWGlobalAllocLock16(
    WORD    wFlags,           // object allocation flags
    DWORD   cb,              // number of bytes to allocate
    LPWORD  phMem            // handle to global memory object
);

LPVOID WINAPI WOWGetVDMPointer(
    DWORD   vp,              // 16:16 address
    DWORD   dwBytes,        // size of vp block
    BOOL    fProtectedMode // protected mode flag
);

```

For further information about this functions and its environment see the „Microsoft SDK“ under „Programming and Tools Guides“ | „Programming Techniques“ | „Generic Thanks“.

4.1 Volume

4.1.1 WODM_GETMUTE

Availability: Arc88 Analog Out, Prodif PLUS Analog Out, Marc 8 Midi Analog Out, Marc 2 Analog Out, Marc 2 PRO Analog Out, Marc 4 Midi/Digi Analog Out

Via this function you can get the mute state of the playback devices two channels.

uMessage	dwParam1	dwParam2
WODM_GETMUTE	Pointer to 32-bit value, which will be filled as follows: Lower WORD = 0 = Left channel On Lower WORD = 1 = Left channel Off Higher WORD = 0 = Right channel On Higher WORD = 1 = Right channel Off	Unused

4.1.2 WODM_SETMUTE

Availability: ARC 88 Analog Out, Prodif PLUS Analog Out, Marc 8 Midi Analog Out, Marc 2 Analog Out, Marc 2 PRO Analog Out, Marc 4 Midi/Digi Analog Out

Via this function you set the mute state of the playback devices two channels.

uMessage	dwParam1	dwParam2
WODM_SETMUTE	32-bit value, which contents will be evaluated as follows: Lower WORD = 0 = Left channel On Lower WORD = 1 = Left channel Off Higher WORD = 0 = Right channel On Higher WORD = 1 = Right channel Off	Unused

4.1.3 WODM_MON_GETVOLUME

Availability: ARC 88 Analog Out

Via this function you can get the adjusted volume, in which the two input channels will be mixed to the appropriate output channels. The function corresponds in its behaviour to the MME standard function *WaveOutGetVolume*, but is related to the mixer volume mentioned above.

Note, that every routable input source (see [WIDM_SETROUTING](#) at chapter 5.3.3) has its own volume value. I.e. after changing the signal routing via *WIDM_SETROUTING*, *WODM_MON_GETVOLUME* will return a new value, associated to the new input source.

uMessage	dwParam1	dwParam2
WODM_MON_GETVOLUME	Pointer to a 32-bit value. The lower word returns the volume of the left channel; the higher word returns the volume of the right channel. 0000h equivalent to -34.5 dB FFFFh equivalent to +12 dB Within this range intermediate steps will be resolved linear.	Unused

4.1.4 WODM_MON_SETVOLUME

Availability: ARC 88 Analog Out

Via this function you set the volume, in which the two input channels are mixed to the appropriate output channels. The function corresponds in its behaviour to *WaveOutSetVolume*, but is related to the mixer volume mentioned above.

Note, that every routable input source (see [WIDM SETROUTING](#) at chapter 5.3.3) has its own volume value. I.e. *WODM_MON_SETVOLUME* will change the volume value of the just routed input source.

uMessage	dwParam1	dwParam2
WODM_MON_SETVOLUME	32-bit value. The lower word contains the value for the left channel; the higher word contains the value for the right channel. 0000h equivalent to -34.5 dB FFFFh equivalent to +12 dB Within this range intermediate steps will be resolved linear.	Unused

4.2 Level Capturing

4.2.1 WODM_ENABLELEVEL

Availability: All MARIAN Playback devices

Via this function you can enable the capturing and evaluation of the audio signal level, which is send to a playback device by an application. Enabling this function leads to a lower system performance.

uMessage	dwParam1	dwParam2
WODM_ENABLELEVEL	Unused	Unused

4.2.2 WODM_DISABLELEVEL

Availability: All MARIAN playback devices

Via this function you can disable the capturing and evaluation of the audio signal level, which is send to a playback device by an application.

uMessage	dwParam1	dwParam2
WODM_DISABLELEVEL	Unused	Unused

4.2.3 WODM_GETLEVEL

Availability: All MARIAN playback devices

Via this function you get the last stored peak value of the signal amplitude. After that, the driver will set that internal value to 0 and continues to capture the level peak values. For using this function you have to call *WODM_ENABLELEVEL* first.

uMessage	dwParam1	dwParam2
WODM_GETLEVEL	Pointer to a 8 byte memory block which will be filled as follows: Lower DWORD = PEAK left channel Higher DWORD = PEAK right channel	Unused

4.3 Routing

Via the output routing functions [WODM_GETROUTING](#) and [WODM_SETROUTING](#) (chapter 0 and 4.3.3) you can get respectively determine the source signal for the output of a playback device.

4.3.1 OUTROUTING Formats

The functions [WODM_GETROUTING](#) and [WODM_SETROUTING](#) expect a DWORD parameter determining the appropriate output routing. This parameter is device specific and will be explained in the following.

4.3.1.1 OUTROUTING Format Prodif 96 PRO, Prodif PLUS

```
typedef struct {
    BYTE    Route;
    BYTE    AdatLeftChn;
    BYTE    AdatRightChn;
    BYTE    reserved;
} OUTROUTING, *POUTROUTING;
```

Route

Source signal for the output.

Valid values:

```
#define oroutePlay      0      // playback signal of the application
#define oroute_OppOut  1      // output signal is signal of the „other“
                             // output. When using for Analog Out,
                             // the source signal is Digital Out;
                             // when using for Digital Out, the source
                             // signal is Analog Out
#define oroute_AnaInp  2      // output signal is signal at Analog In
#define oroute_DigInp  3      // output signal is signal at Digital In
#define oroute_AdatInp 4      // output signal is signal at Adat In
                             // Which Adat input channels are played back
                             // at the left respectively
                             // right channel of the output, is
                             // determined // by the fields AdatLeftChn
                             // and AdatRightChn.
```

AdatLeftChn, AdatRightChn

Adat channel for left respectively right channel of the Analog Output

Valid values:

0..7 for Adat channels 1..8

Comments:

The routing value `orouteAdatInp` is applicable to the analog playback device only. Eventually it leads to a change to the cards ADAT operating mode. Eventually the routing values `orouteDigInp` and `orouteOppositeOut` (for Analog Out) lead to a change to the cards Stereo Digital operating mode. For further information about the Prodif's operating modes see chapter 8 under "[Special features of the Prodif 96 Pro and Prodif PLUS](#)".

Please note the automatic input monitor modes, described under "[INPROUTING Prodif 96 PRO, Prodif PLUS und MARC 2](#)" at chapter 5.3.1.2.

4.3.1.2 OUTROUTING Format Marc 8 Midi

```
typedef struct {
    BOOLEAN InpMonOnRec;
    BOOLEAN InpMonOnPunch;
    __int16 Route;
} OUTROUTING, *POUTROUTING;
```

Route :

Signal source for the output.

Valid values:

```
#define oroutePlay -1 // Playback signal of the application
#define orouteInp12 0 // Signal at the input 1-2
#define orouteInp34 1 // Signal at the input 3-4
#define orouteInp56 2 // Signal at the input 5-6
#define orouteInp78 3 // Signal at the input 7-8
```

InpMonOnRec :

Enables or disables the Auto-Record-Monitor-Mode. If the mode is activated, there will be an automatic routing of the set input to the output when starting a recording for this input (via *waveInStart*).

Valid values:

TRUE || FALSE

InpMonOnPunch :

Enables or disables the Monitor-OnPunch-Mode. If the mode is activated, there will be an automatic routing of the set input to the output if that input detects a Record PunchIn or Record PunchOut signal via [WIDM_MON_SETMUTE](#). For further information see „[Monitoring](#)“ at chapter 6.

Valid values:

TRUE || FALSE

4.3.1.3 OUTROUTING Format Marc 2 and Marc 2 PRO

```
typedef struct {
    BYTE Route;
    BYTE reserved;
    BYTE reserved;
    BYTE reserved;
} OUTROUTING, *POUTROUTING;
```

Route

Source signal for the output.

Valid values:

```
#define oroute_Play 0 // output signal is playback signal of the
// application

#define oroute_OppOut 1 // output signal is signal of the „other“
// output. When using for Analog Out,
// the source signal is Digital Out
// when using for Digital Out, the source
// signal is Analog Out

#define oroute_AnaInp 2 // output signal is signal at Analog In

#define oroute_DigInp 3 // output signal is signal at Digital In
```

Please note the automatic input monitor modes, described under „[INPROUTING Prodif 96 PRO, Prodif PLUS und MARC 2](#)“ at chapter 5.3.1.2.

4.3.1.4 OUTROUTING Format Marc 4 Midi

```
typedef struct {
    BOOLEAN InpMonOnRec;
    BOOLEAN InpMonOnPunch;
    __int16 Route;
} OUTROUTING, *POUTROUTING;
```

Source signal for the output.

Valid values:

```
#define oroutePlay -1    // Playback signal of the application
#define orouteInp12 0   // Signal at input 1-2
#define orouteInp34 1   // Signal at input 3-4
```

InpMonOnRec :

Enables or disables the Auto-Record-Monitor-Mode. If the mode is activated, there will be an automatic routing of the set input to the output if you start a record for this input (via *waveInStart*).

Valid values:

```
TRUE || FALSE
```

InpMonOnPunch :

Enables or disables the Monitor-OnPunch-Mode. If the mode is activated, there will be an automatic routing of the set input to the output if that input detects a Record PunchIn or Record PunchOut signal via [WIDM MON SETMUTE](#). For further information see chapter 6 under „[Monitoring](#)“.

Valid values:

```
TRUE || FALSE
```

4.3.1.5 OUTROUTING Format Marc 4 Digi

```
typedef struct {
    BYTE    Route;
    BYTE    reserved;
    BYTE    reserved;
    BYTE    reserved;
} OUTROUTING, *POUTROUTING;
```

Route

Signalquelle für Ausgang.

Zulässige Werte:

```
#define oroute_AnaPlay12 0 // Output signal is application's playback
                          // signal at device „Analog 1-2“
#define oroute_AnaInp12  1 // Output signal is signal at input
                          // „Analog 1-2“
#define oroute_AnaPlay34 2 // Output signal is application's playback
                          // signal at device „Analog 3-4“
#define oroute_AnaInp34  3 // Output signal is signal at input
                          // „Analog 3-4“
#define oroute_DigPlay   4 // Output signal is application's playback
                          // signal at device „Digital“
#define oroute_DigInp    5 // Output signal is signal at input
                          // „Digital“
```

4.3.2 WODM_GETROUTING

Availability: All MARIAN playback devices but ARC 88. Not applicable to the ADAT playback devices of the Prodif 96 PRO and Prodif PLUS.

Via this function you find out the source signal, which is played back at the appropriate playback devices output. Eventually the routing value contains information about the automatic routing modes (see „[Monitoring](#)“ at chapter 6)

uMessage	dwParam1	dwParam2
WODM_GETROUTING	Pointer to 32-bit value. The interpretation of this value is device specific: Prodif's , Marc 8 Midi , Marc 2 , Marc 2 PRO , Marc 4 Midi , Marc 4 Digi	Unused

4.3.3 WODM_SETROUTING

Availability: All MARIAN playback devices but ARC 88. Not applicable to the ADAT playback devices of the Prodif 96 PRO and Prodif PLUS.

Via this function you determine the source signal, which will be played back at the appropriate output of the playback device. Eventually you determine the automatic routing modes too. (see „[Monitoring](#)“ at chapter 6)

uMessage	dwParam1	dwParam2
WODM_GETROUTING	32-bit value. The interpretation of this value is device specific: Prodif's , Marc 8 Midi , Marc 2 , Marc 2 PRO , Marc 4 Midi , Marc 4 Digi	Unused

4.4 Playback Format

4.4.1 WODM_GETSAMPLERATE

Availability: All MARIAN playback devices.

This function provides the possibility to interrogate the current sample rate, which the playback device is working with. Usually that will be the sample rate, which the playback device has been opened with via *waveOutOpen*. If the playback device is controlled by another clock than the internal clock (eg. external Wordclock or Clock of the Digital Input) the sample rate of that clock will be returned.

uMessage	dwParam1	dwParam2
WODM_GETSAMPLERATE	Pointer to 32-bit value, which is filled with the current sample rate.	Unused

4.4.2 WODM_GETFORMAT

Availability: Digital Playback devices Prodif 96 PRO and Prodif PLUS; excluding ADAT Playback devices.

Via this function you receive device specific format information. In case of stereo digital playback devices that will be the information of the digital data format.

uMessage	dwParam1	dwParam2
WODM_GETFORMAT	Pointer to 32-bit value which is filled with the format information Prodif 96 PRO: 0x00000000 S/PDIF Audio 0x00000010 AES/EBU Audio Prodif PLUS: 0x00000020 S/PDIF Audio 0x00000088 AES/EBU Audio 0x00000080 AES/EBU Non Audio	Unused

4.4.3 WODM_SETFORMAT

Availability: Digital playback devices Prodif 96 PRO and Prodif PLUS; excluding ADAT Playback devices.

Via this function you define the device specific formats. In case of stereo digital playback devices that will be the digital data format.

uMessage	dwParam1	dwParam2
WODM_SETFORMAT	32-bit value including the format information: Prodif 96 PRO: 0x00000000 S/PDIF Audio 0x00000010 AES/EBU Audio Prodif PLUS: 0x00000020 S/PDIF Audio 0x00000088 AES/EBU Audio 0x00000080 AES/EBU Non Audio	Unused

5. Extended Functions for Record Devices

The following new messages have been introduced for the function *WaveInMessage*. The required messages have to be transferred to the parameter *uMessage* of this function. The messages have been implemented to the Headerfile enclosed.

```

DRVM_USER           = 4000h
WIDM_GETVOLUME     = DRVM_USER + 50
WIDM_SETVOLUME     = DRVM_USER + 51
WIDM_GETROUTING    = DRVM_USER + 52
WIDM_SETROUTING    = DRVM_USER + 53
WIDM_GETPITCH      = DRVM_USER + 54
WIDM_SETPITCH      = DRVM_USER + 55
WIDM_ENABLELEVEL   = DRVM_USER + 56
WIDM_DISABLELEVEL = DRVM_USER + 57
WIDM_GETLEVEL      = DRVM_USER + 58
WIDM_MON_GETVOLUME = DRVM_USER + 59
WIDM_MON_SETVOLUME = DRVM_USER + 60
WIDM_MON_GETMUTE   = DRVM_USER + 61
WIDM_MON_SETMUTE   = DRVM_USER + 62
WIDM_GETSAMPLERATE = DRVM_USER + 63
WIDM_GETFORMAT     = DRVM_USER + 64
WIDM_SETFORMAT     = DRVM_USER + 65
WIDM_GETMUTE       = DRVM_USER + 66
WIDM_SETMUTE       = DRVM_USER + 67

```

WaveInMessage is declared as follows:

```

DWORD waveInMessage (
    HWAVEIN hwi,
    UINT uMsg,
    DWORD dwParam1,
    DWORD dwParam2
);

```

Parameter:

<i>hwi</i>	Handle of the record devices (return value of <i>waveInOpen</i>)
<i>uMsg</i>	One of the new defined WIDM messages
<i>dwParam1</i>	Parameter 1 dependent on message
<i>dwParam2</i>	Parameter 1 dependent on message

Return: *WaveInMessage* returns the Standard *mmsyserr_xxx* Codes. *WaveInMessage* returns the general error code *mmsyserr_error*, if a function can't be carried out, caused by a hardware specific restriction.

Headerfile: *mmsystem.h*

WIN32 Applications under Windows95/98™

For several functions a pointer will be expected as a parameter. For having the wave driver a 16 Bit Protect Mode Architecture under Windows 95/98™, these pointer will be interpreted in SEG:OFS format. Use the following functions of the WOW32.DLL (part of Windows 95/98™), to make that kind of pointer applicable to your WIN32 Application as well as to the driver:

```

DWORD WINAPI WOWGlobalAllocLock16(
    WORD wFlags,           // object allocation flags
    DWORD cb,             // number of bytes to allocate
    LPWORD phMem          // handle to global memory object
);
LPVOID WINAPI WOWGetVDMPointer(
    DWORD vp,             // 16:16 address
    DWORD dwBytes,       // size of vp block
    BOOL fProtectedMode // protected mode flag
);

```

For further information to this functions see „Microsoft SDK“ under „Programming and Tools Guides“ | „Programming Techniques“ | „Generic Thanks“.

5.1 Input Level and Monitoring

5.1.1 WIDM_GETVOLUME

Availability: All analog MARIAN record devices but not Prodif PLUS and Marc 2 PRO.

Via this function you can read back the gain value from the preamplifier of the two input channels. This function corresponds in its behaviour to the *WaveOutGetVolume*, but is related to the gain value mentioned above.

ARC88: Note, that every routable input source (see [WIDM_SETROUTING](#) chapter 5.3.3) has its own level value. I.e. after changing the signal routing via *WIDM_SETROUTING*, a different value (related to the new input source) will be returned by *WIDM_GETVOLUME*.

uMessage	dwParam1	dwParam2
WIDM_GETVOLUME	Pointer to 32-bit value. The lower word returns the volume of the left channel; the higher word returns the volume of the right channel. ARC88: 0x0000 0 dB 0xFFFF +22.5 dB Prodif 96 PRO: 0x0000 -95.5 dB 0xFFFF +31.5 dB Marc 8 Midi, Marc 2, Marc 4 Midi/Digi: 0x0000 -72 dB 0xFFFF +18 dB	Unused

5.1.2 WIDM_SETVOLUME

Availability: All analog MARIAN record device but not Prodif PLUS and Marc 2 PRO.

Via this function you set the gain value for the preamplifier of the two input channels. This function corresponds in its behaviour to *WaveOutSetVolume*, but is related to the gain value mentioned above.

ARC88: Note, that every routable input source (see [WIDM_SETROUTING](#) chapter 5.3.3) has its own level value. I.e. *WIDM_SETVOLUME* will change the gain value of the just routed input source.

uMessage	dwParam1	DwParam2
WIDM_SETVOLUME	32-bit value. The lower word contains the value for the left channel; the higher word contains the value for the right channel. ARC88: 0x0000 0 dB 0xFFFF +22.5 dB Prodif 96 PRO: 0x0000 -95.5 dB 0xFFFF +31.5 dB Marc 8 Midi, Marc 2, Marc 4 Midi/Digi: 0x0000 -72 dB 0xFFFF +18 dB	Unused

5.1.3 WIDM_GETMUTE

Availability: Marc 8 Midi, Marc 4 Midi and analog record devices Marc 2, Marc 4 Digi

This function detects the mute state of the two input channels.

uMessage	dwParam1	dwParam2
WIDM_GETMUTE	Pointer to 32-bit value, which will be filled as follows: Lower WORD = 0 = Left channel on Lower WORD = 1 = Left channel off Higher WORD = 0 = Right channel on Higher WORD = 1 = Right channel off	Unused

5.1.4 WIDM_SETMUTE

Availability: Marc 8 Midi, Marc 4 Midi and analog record devices Marc 2, Marc 4 Digi

This function sets the mute state of the two input channels.

uMessage	dwParam1	dwParam2
WIDM_SETMUTE	32-bit value containing the mute state: Lower WORD = 0 = Left channel on Lower WORD = 1 = Left channel off Higher WORD = 0 = Right channel on Higher WORD = 1 = Right channel off	Unused

5.1.5 WIDM_MON_GETVOLUME

Availability: ARC88 Analog

Via this function you can get the adjusted volume, in which the two input channels are mixed to the appropriate output channels. This function corresponds in its behaviour to [WODM_MON_GETVOLUME](#).

Note, that every routable input source (see [WIDM_SETROUTING](#) at chapter 5.3.3) has its own volume value. I.e. after changing the signal routing via *WIDM_SETROUTING*, a different value (related to the new input source) will be returned by *WIDM_MON_GETVOLUME*.

uMessage	dwParam1	dwParam2
WIDM_MON_GETVOLUME	Pointer to 32-bit value. The lower word returns the volume of the left channel; the higher word returns the volume of the right channel. 0000h equivalent to -34.5 dB FFFFh equivalent to +12 dB	Unused

5.1.6 WIDM_MON_SETVOLUME

Availability: ARC88 Analog

Via this function you set the volume, in which two input channels are mixed to the appropriate output channels. This function corresponds in its behaviour to [WODM_MON_SETVOLUME](#).

Note, that every routable input source (see [WIDM_SETROUTING](#) at chapter 5.3.3) has its own volume value. I.e. *WIDM_MON_SETVOLUME* will change the volume value of the just routed input source.

uMessage	dwParam1	dwParam2
WIDM_MON_SETVOLUME	32-bit value. The lower word contains the value for the left channel; the higher word contains the value for the right channel. 0000h equivalent to -34.5 dB FFFFh equivalent to +12 dB	Unused

5.1.7 WIDM_MON_GETMUTE

Availability: All MARIAN record device but the ADAT record devices of the Prodif's

Via this function you detect, whether the input signal of the record device can be heard at the defined output. You can get the defined monitor output hardware specific via [WIDM_GETROUTING](#) or [WODM_GETROUTING](#).

ARC88: Note, that every routable input source (see [WIDM_SETROUTING](#) at chapter 5.3.3) has its own mute state. I.e., after changing the signal routing via *WIDM_SETROUTING*, a different mute state (related to the new input source) will be returned by *WIDM_MON_GETMUTE*.

uMessage	dwParam1	dwParam2
WIDM_MON_GETMUTE	Pointer to 32-bit value, which will be filled as follows: Lower WORD = 0 = Left Channel On Lower WORD = 1 = Left Channel Off Higher WORD = 0 = Right Channel On Higher WORD = 1 = Right Channel Off	Unused

5.1.8 WIDM_MON_SETMUTE

Availability: All MARIAN record devices but ADAT record device of the Prodif's.

Via this function you decide, whether to hear the input signal of the record device at the defined output. For using this function to signalise a PunchIn/PunchOut recording and the corresponding automatic monitoring too, you have to enable this function at the monitor mode first.

You can define the monitor output and the monitor mode hardware specific via [WIDM_SETROUTING](#) or [WODM_SETROUTING](#).

Note: The ARC88 is the only card with the input signal mixed to the output. For every other card, the output will be switched to the input. Therefore a playback signal will not be heard. Only for the ARC88 you can monitor the left channel separately from the right channel at the output, every other card can do this as a stereo pair only. Thereby the mute information for the left channel is also used for the right channel.

ARC88: Note, that every routable input source (see [WIDM_SETROUTING](#) at chapter 5.3.3) has its own mute state. I.e., *WIDM_MON_SETMUTE* will change the mute state of the just routed input source.

uMessage	dwParam1	dwParam2
WIDM_MON_SETMUTE	32-bit value, its contents will be evaluated as follows: Lower WORD = 0 = Left Channel on Lower WORD = 1 = Left Channel off Higher WORD = 0 = Right Channel on Higher WORD = 1 = Right Channel off	Unused

5.2 Level Capturing

5.2.1 WIDM_ENABLELEVEL

Availability: All MARIAN record devices

Via this function you can enable the capturing and evaluating of the audio signal level, which is at the input of a record device. Enabling this function leads to a lower system performance.

uMessage	dwParam1	dwParam2
WIDM_ENABLELEVEL	Unused	Unused

5.2.2 WIDM_DISABLELEVEL

Availability: All MARIAN record devices

Via this function you can disable the capturing and evaluating of audio signal level, which is at the input of a record device.

uMessage	dwParam1	dwParam2
WIDM_DISABLELEVEL	Unused	Unused

5.2.3 WIDM_GETLEVEL

Availability: All MARIAN record devices

Via this function you get the last stored peak value of the signal amplitude. After that the driver will set this internal value to 0 and continues to capture the peak values. For using this function you have to call the function WIDM_ENABLELEVEL first.

uMessage	dwParam1	dwParam2
WIDM_GETLEVEL	Pointer to 8 Byte memory block, which will be filled as follows: Lower DWORD = PEAK left Channel Higher DWORD = PEAK right Channel	Unused

5.3 Routing

5.3.1 INPROUTING Formats

The functions [WIDM_GETROUTING](#) and [WIDM_SETROUTING](#) (chapter 0 and 5.3.3) expect a DWORD Parameter, which determines the appropriate input routing. That parameter is device specific and will be explained in the following.

5.3.1.1 INPROUTING ARC 88

```
typedef struct {
    BYTE    routeLeft;
    BOOLEAN MuteWaveOnMon;
    BYTE    routeRight;
    BYTE    MonMode;
} INPROUTING, *PINPROUTING;
```

routeLeft,
routeRight

determine the source signal for the left respectively right channel of the record device.

Valid values:

```
#define iroute_Line 0 // Input source is RCA jack with line level
#define iroute_Mic 1 // Input source is RCA jack with microphone level
#define iroute_CD 2 // Input source is internal CD connector
// Valid for Record device 3-4 only
```

MuteWaveOnMon

determines while monitoring (see [WIDM_MON_SETMUTE](#) at chapter 5.1.8) whether the input signal is mixed or switched to the output.

Valid values:

```
TRUE // Playback signal is disabled while Monitoring
FALSE // Monitor signal will be mixed to the playback
signal
```

MonMode

determines, when the monitoring of the input at the output happens respectively is permitted.

Valid values:

```
#define monNever 0 // The complete Monitoring is deactivated
#define monOnRec 1 // Automatic Monitoring while record activated
#define monOnPunch 2 // Monitoring at PunchIn/PunchOut permitted
```

5.3.1.2 INPROUTING Prodif 96 PRO, Prodif PLUS, MARC 2 and Marc 2 PRO

```
typedef struct {
    BYTE    Route;
    BYTE    MonRoute;
    BYTE    MonMode;
    BYTE    reserved;
} INPROUTING, *PINPROUTING;
```

Route

Determines the source signal for the digital record device. While using the analog record device it will be ignored.

Valid values:

```
#define iroute_Nothing 0 // Input source disabled
#define iroute_Optical 1 // Input source is optical input
#define iroute_Electrical 2 // Input source is electrical input
#define iroute_CD 3 // Input source is internal CD connector
// Not applicable to Prodif PLUS
```

Route Marc 2 PRO

```
#define iroute_Nothing    0 // Input Source disabled
#define iroute_Optical    1 // Input Source is optical input
#define iroute_AesEbu     2 // Input Source is XLR AES/EBU
#define iroute_CD         3 // Input Source is internal CD connector
#define iroute_Spdif      4 // Input Source is S/PDIF input
```

MonRoute

Determines the monitor output for record device.

Range of values:

```
#define mroute_Nothing    0 // No Monitor output
#define mroute_AnaOut     1 // Analog Out is Monitor output
#define mroute_DigOut     2 // Digital Out is Monitor output
```

MonMode

determines, when the monitoring of the input at the output happens respectively is permitted.

```
#define monNever    0 // complete Monitoring is deactivated
#define monOnRec    1 // Automatic Monitoring while record is activated
#define monOnPunch  2 // Monitoring at PunchIn/PunchOut is permitted.
```

5.3.1.3 INPROUTING Marc 4 Digi

```
typedef struct {
    BYTE    Route;
    BYTE    MonRoute;
    BYTE    MonMode;
    BYTE    reserved;
} INPROUTING, *PINPROUTING;
```

Route

Determines the input source for the recording devices.

Value Range Digital Input:

```
#define iroute_Nothing    0 // Input Source disabled
#define iroute_Optical    1 // Input Source is optical Input
#define iroute_AesEbu     2 // Input Source is XLR AES/EBU
#define iroute_DigCD      3 // Input Source is internal Digital CD
#define iroute_Spdif      4 // Input Source is RCA S/PDIF Input
```

Value Range Analog Input:

```
#define iroute_Nothing    0 // Input Source disabled
#define iroute_AnaCD      5 // Input Source is internal Analog CD
#define iroute_Jack       6 // Input Source is Line Input
```

MonRoute

Determines the monitor output for record device.

Range of values:

```
#define mroute_Nothing    0 // No Monitor output
#define mroute_AnaOut12   1 // Analog Out 1-2 is Monitor output
#define mroute_AnaOut34   2 // Analog Out 3-4 is Monitor output
#define mroute_DigOut     3 // Digital Out is Monitor output
```

MonMode

determines, when the monitoring of the input at the output happens respectively is permitted.

```
#define monNever    0 // complete Monitoring is deactivated
#define monOnRec    1 // Automatic Monitoring while record is activated
#define monOnPunch  2 // Monitoring at PunchIn/PunchOut is permitted.
```

5.3.2 WIDM_GETROUTING

Availability: ARC88 Analog, Prodif's but ADAT Inputs, MARC 2, MARC 2 PRO, MARC 4 Digi

Via this function you receive the adjusted source signal for the record device.

uMessage	dwParam1	dwParam2
WIDM_GETROUTING	Pointer to 32-bit value, which has to be interpreted hardware specific: ARC88 , Prodif's und MARC 2/PRO , MARC 4 Digi	Unused

5.3.3 WIDM_SETROUTING

Availability: ARC88 Analog, Prodif's except ADAT Inputs, MARC 2, MARC 2 PRO, MARC 4 Digi

Via this function you determine the source signal for the record device.

uMessage	dwParam1	dwParam2
WIDM_SETROUTING	32-bit value – interpreted hardware specific. ARC88 , Prodif's and MARC 2/PRO , MARC 4 Digi	Unused

5.4 Record Format

5.4.1 WIDM_GETSAMPLERATE

Availability: All MARIAN record devices

This function provides the possibility to interrogate the current sample rate the record device is running with. Usually that will be the sample rate the record device has been opened with via *waveInOpen*. If the record device is controlled by another clock than the internal clock (eg. external word clock or clock of the Digital Input), the sample rate of that clock will be returned.

uMessage	dwParam1	dwParam2
WIDM_GETSAMPLERATE	Pointer to 32-bit value which will be filled with the current Sample rate	Unused

5.4.2 WIDM_GETFORMAT

Not implemented yet. Future use reserved.

5.4.3 WIDM_SETFORMAT

Not implemented yet. Future use reserved.

5.5 Pitch

Via the pitch functions you can affect the sample rate of the devices. The pitch value serves as multiplier for the sample rate adjusted at *WaveInOpen* or *WaveOutOpen*. Note, that changing the pitch value via *WIDM_SETPITCH* or *waveOutSetPitch* affects all devices, which are connected with the same clock. Variable clock setups are available for the Prodig's, MARC 2, Marc 2 PRO and MARC 4 Digi and will be implemented by the appropriate driver interfaces. The ARC88, Marc 8 Midi and MARC 4 Midi have an internal Clock only. The clock determines the sample rate for all devices.

5.5.1 WIDM_GETPITCH

This function provides the possibility to get the adjusted pitch value and represents the pendant to the standard MME function *WaveOutGetPitch*. The format of the pitch value is equivalent to the pitch value format of the function *WaveOutGetPitch*.

uMessage	dwParam1	dwParam2
WIDM_GETPITCH	Pointer to 32-bit value, which will be filled with the pitch value.	Unused

5.5.2 WIDM_SETPITCH

Via this function you can set the pitch value. The format of the pitch value is equivalent to the pitch value format of the function *WaveOutSetPitch*.

uMessage	dwParam1	dwParam2
WIDM_SETPITCH	32-bit value, which contains the pitch value.	Unused

6. Monitoring

All MARIAN sound systems support monitoring. That is a hardware supported and therefore latency free possibility to monitor In/Out channels at output channels. The MARIAN MME Extension distinguishes between three Monitoring modes: *Immediate Monitoring*, *Auto Record Monitoring* and *Punch Monitoring*.

Immediate Monitoring

This mode leads to an immediate play back of a source signal at the requested output. Source signals can be hardware specific In- or Outputs. Realize this function via [WODM SETROUTING](#), (see chapter 4.3.)

Auto Record Monitoring

This Mode leads to play back of a source signal at the requested output if you start the record for the appropriate record device via *waveInStart*. The playback of this source signal ends if the record is concluded via *waveInStop* or *waveInReset*. This monitor mode and the requested monitor output will be determined hardware specific via [WODM SETROUTING](#) or [WIDM SETROUTING](#).

Punch Monitoring

This Mode leads to play back of a source signal at the requested output if you start a PunchIn record for the appropriate record device. The playback of this input source will end if the record is stopped via PunchOut. PunchIn and PunchOut will be signalled to the record device via [WIDM MON SETMUTE](#), (see chapter 5.1.8). This monitor mode and the requested monitor output will be determined hardware specific via [WODM SETROUTING](#) or [WIDM SETROUTING](#).

7. Start/Stop Synchronization

While using several Record and Playback devices via MME, the problem of starting and stopping all the involved record and playback devices synchronous occurs, caused by the exclusive separate addressability. To solve this problem you can connect the devices with the help of the appropriate driver applications and the *SyncBus*. This leads to the following functionality. As a matter of principle start and stop commands always affect ALL record and playback devices which are configured as a cooperation. Those are called „Cooperated Devices“ in the following.

7.1 Synchronous Playback Start

Requirements: all necessary, cooperated playback devices have been set in the pause state via *WaveOutPause* and have been provided with buffers via *WaveOutWrite*.

Actions to perform: Call *WaveOutRestart* for all cooperated, paused devices

Resulting behaviour: If you call *WaveOutRestart* for the first time, all cooperated playback devices will start synchronous.

7.2 Synchronous Playback Pause und Restart

Actions to perform:

- For Playback pause: Call *WaveOutPause* for any, cooperated playback device.
- For Playback restart: Call *WaveOutRestart* for this device.

Resulting behaviour:

All cooperated Playback devices will stop the playback. The current playback position maintains. After *WaveOutRestart* all cooperated playback devices will start synchronous.

7.3 Synchronous Playback Stop

Actions to perform: Call *WaveOutReset* for all running, cooperated playback devices

Resulting behaviour: If you call *WaveOutReset* for the first time, all cooperated playback devices will synchronous stop the playback. The current playback position will be set to zero.

7.4 Synchronous Record Start

Requirements: all necessary, cooperated record devices have been provided with buffers via *WaveInAddBuffer*.

Actions to perform: Call *WaveInStart* for all cooperated record devices.

Resulting behaviour: If you call *WaveInStart* for the first time, all cooperated record devices will start synchronous.

7.5 Synchronous Record Pause and Restart

Actions to perform:

- For record pause: Call *WaveInStop* for any cooperated record device.
- For record restart: Call *WaveInStart* for this device.

Resulting behaviour:

All cooperated record devices will stop recording. The current record position maintains. After *WaveInStart* the cooperated record devices will start synchronous.

Comments: Via *WaveInStop* the current record buffer returns to the Application. A following *WaveInStart* will be successful on the issue of synchronizing only, if at least one record buffer is available for the record device.

7.6 Synchronous Record Stop

Actions to perform: Call *WaveInReset* for all running, cooperated record devices.

Resulting behaviour: While first calling *WaveInReset* all cooperated record devices will stop the playback synchronous. The current record position will be set to zero.

7.7 Synchronous Record and Playback Start

Requirements:

All necessary, cooperated playback devices have been set in the pause state via *WaveOutPause* and have been provided with buffers via *WaveOutWrite*.

All necessary, cooperated record devices have been provided with buffers via *WaveInAddBuffer*.

Actions to perform:

- Call *WaveOutRestart* for all cooperated, paused devices
- Call *WaveInStart* for all cooperated Record devices

Resulting behaviour: If you call a start command for the first time, all cooperated devices will start synchronous.

Comments: The order, in which the devices are prepared for the start or started, doesn't matter.

7.8 Synchronous Record and Playback Pause and Restart

Actions to perform:

- Call *WaveOutPause* for any cooperated Playback device.
- Call *WaveOutRestart* for this device.

Resulting behaviour:

All cooperated devices will stop. The current record and playback position maintains. After *WaveOutRestart* all cooperated devices will start synchronous.

Comments: The combination of *WaveInStop* and *WaveInStart* will not lead to the same result. It rather leads to the pause/restart of the cooperated record devices only!

7.9 Synchronous Record and Playback Stop

Actions to perform:

Call *WaveOutReset* for all running, cooperated playback devices.

Call *WaveInReset* for all running, cooperated record devices.

Resulting behaviour: While first calling a reset function, all cooperated devices will stop synchronous. The current record and playback position is set to zero.

Comments: The order in which the devices are stopped doesn't matter.

8. Special Features Prodif 96 PRO and Prodif PLUS

As a matter of principle the Prodif's operate in two operating modes – the *Stereo Digital Mode* and the *ADAT Mode*. These operating modes are mutually exclusive. Depending on the demand on the driver, it will change the cards to the appropriate operating mode. Changing the operating mode is provided only, if there are no audio actions on the card. If the changing is not possible the driver will return the error code *mmsyserr_error* when calling a required MME Function.

Stereo Digital Mode

This mode provides the possibility to use record- and playback devices for S/PDIF and AES/EBU besides the analog devices. This is the Standard operating mode after starting the Windows operating system. Switching from ADAT operating mode to Stereo Digital Mode takes place under these circumstances:

- Open a stereo digital device via *WaveOutOpen* or *WaveInOpen*.
- Define the digital input or output as a source signal for the analog output (see [WODM_SETROUTING](#) at chapter 4.3).
- Synchronize a device to the clock of the digital input.

ADAT Mode

This mode provides the possibility to use the ADAT record and playback devices besides the analog devices. Switching from Stereo Digital Mode to ADAT Mode takes place under these circumstances:

- Open a ADAT device via *WaveOutOpen* or *WaveInOpen*.
- Define a ADAT input as a source signal for the analog output (see [WODM_SETROUTING](#) at chapter 4.3)
- Synchronize the cards clock generator 2 to the ADAT input.

9. Contact the Developers

Should you have any problems or questions please do not hesitate to contact:



Aribert Laschke & Andreas Hellner OHG

Eisenacher Straße 72

04155 Leipzig

Germany

Email: developers@marian.de