|  |
| --- |
| Developer Kit for Sound Blaster Series |
| Second Edition |

|  |
| --- |
| Library Reference |

Library Reference

File Format

License Agreement/Limitation And Disclaimer Of Warranties

**PLEASE NOTE : BY DOWNLOADING AND/OR USING THE SOFTWARE AND/OR MANUAL ACCOMPANYING THIS LICENSE AGREEMENT, YOU ARE HEREBY AGREEING TO THE FOLLOWING TERMS AND CONDITIONS:**

The software and related written materials, including any instructions for use, are provided on an "AS IS" basis, without warranty of any kind, express or implied. This disclaimer of warranty expressly includes, but is not limited to, any implied warranties of merchantability and/or of fitness for a particular purpose. No oral or written information given by Creative Technology Ltd., its suppliers, distributors, dealers, employees, or agents, shall create or otherwise enlarge the scope of any warranty hereunder. Licensee assumes the entire risk as to the quality and the performance of such software and licensee application. Should the software, and/or Licensee application prove defective, you, as licensee (and not Creative Technology Ltd., its suppliers, distributors, dealers or agents), assume the entire cost of all necessary correction, servicing, or repair.

RESTRICTIONS ON USE

Creative Technology Ltd. retains title and ownership of the manual and software as well as ownership of the copyright in any subsequent copies of the manual and software, irrespective of the form of media on or in which the manual and software are recorded or fixed. By downloading and/or using this manual and software, Licensee agrees to be bound to the terms of this agreement and further agrees that :

**(1) CREATIVE'S FTP/COMPUSERVE ARE THE ONLY ONLINE SITES WHERE USERS MAY DOWNLOAD ELECTRONIC FILES CONTAINING THE MANUAL AND/OR SOFTWARE,**

**(2) LICENSEE SHALL USE THE MANUAL AND/OR SOFTWARE ONLY FOR THE PURPOSE OF DEVELOPING LICENSEE APPLICATIONS COMPATIBLE WITH CREATIVE’S SOUND BLASTER SERIES OF PRODUCTS, UNLESS OTHERWISE AGREED TO BY FURTHER WRITTEN AGREEMENT FROM CREATIVE TECHNOLOGY LTD.; AND,**

**(3) LICENSEE SHALL NOT DISTRIBUTE OR COPY THE MANUAL FOR ANY REASON OR BY ANY MEANS (INCLUDING IN ELECTRONIC FORM) OR DISTRIBUTE, COPY, MODIFY, ADAPT, REVERSE ENGINEER, TRANSLATE OR PREPARE ANY DERIVATIVE WORK BASED ON THE MANUAL OR SOFTWARE OR ANY ELEMENT THEREOF OTHER THAN FOR THE ABOVE SAID PURPOSE, WITHOUT THE EXPRESS WRITTEN CONSENT OF CREATIVE TECHNOLOGY LTD.. CREATIVE TECHNOLOGY LTD. RESERVES ALL RIGHTS NOT EXPRESSLY GRANTED TO LICENSEE IN THIS LICENSE AGREEMENT.**

LIMITATION OF LIABILITY

In no event will Creative Technology Ltd., or anyone else involved in the creation, production, and/or delivery of this software product be liable to licensee or any other person or entity for any direct or other damages, including, without limitation, any interruption of services, lost profits, lost savings, loss of data, or any other consequential, incidental, special, or punitive damages, arising out of the purchase, use, inability to use, or operation of the software, and/or licensee application, even if Creative Technology Ltd. or any authorised Creative Technology Ltd. dealer has been advised of the possibility of such damages. Licensee accepts said disclaimer as the basis upon which the software is offered at the current price and acknowledges that the price of the software would be higher in lieu of said disclaimer. Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitations and exclusions may not apply to you.

Information in this document is subject to change without notice. Creative Technology Ltd. shall have no obligation to update or otherwise correct any errors in the manual and software even if Creative Technology Ltd. is aware of such errors and Creative Technology Ltd. shall be under no obligation to provide to Licensee any updates, corrections or bug-fixes which Creative Technology Ltd. may elect to prepare.

Creative Technology Ltd. does not warrant that the functions contained in the manual and software will be uninterrupted or error free and Licensee is encouraged to test the software for Licensee's intended use prior to placing any reliance thereon.

Copyright 1993-1996 by Creative Technology Ltd. All rights reserved.

Sound Blaster, Sound Blaster Pro, Sound Blaster 16, and Wave Blaster are trademarks of Creative Technology Ltd.

IBM is a registered trademark of International Business Machines Corporation.

MS-DOS is a registered trademark and Windows is a trademark of Microsoft Corporation.

All other products are trademarks or registered trademarks of their respective owners.

Contents

Introduction

How to use this Manual

Document Conventions

Function Reference Document Format

Chapter 1 High Level Digitized Sound Drivers

High-Level Language Interface

Function Prefixes

Include Files

Functions by Category

CT-VOICE.DRV Assembly Interface

CTVDSK.DRV Assembly Interface

CTWMEM.DRV Assembly Interface

CTWDSK.DRV Assembly Interface

Chapter 2 High-Level Auxiliary Driver

High-Level Language Interface

Function Prefix

Include Files

Functions by Category

Assembly Interface

Chapter 3 Creative Multimedia System Driver

Device Driver Entry-Point

Callback Function

Device Driver Messages

Auxiliary Audio Device Messages

Sound Device Messages

Signal Processing Device Messages

Device Driver Data Structures

Common Data Structure

Auxiliary Audio Device Data Structures

Sound Device Data Structures

Signal Processing Device Data Structures

Device Driver Error Messages

Device Driver Message Reference

Auxiliary Audio Device Message Reference

Sound Device Message Reference

Signal Processing Device Message Reference

Device Driver Data Structure Reference

Common Data Structure Reference

Auxiliary Audio Device Data Structure Reference

Sound Device Data Structure Reference

Signal Processing Device Data Structure Reference

Chapter 4 MIDI Driver

High-Level Language Interface

Function Prefix

Include Files

Functions by Category

Manifest Constants

Assembly Interface

Chapter 5 CD-ROM Audio

High-Level Language Interface

Function Prefix

Include Files

Functions by Category

Values Passed To / Returned By Functions

Error Codes

Appendix A : File Format

Creative Voice File (VOC) Format

Header Block

Data Block

Creative ADPCM Wave Type Format

Index

Introduction

This manual describes the library functions included with the Developer Kit for the Sound Blaster series of audio cards. It also describes the Creative Voice File (.VOC) format and the Creative ADPCM wave type which has been registered with Microsoft.

How to use this Manual

This manual is organized into the following chapters.

Chapter 1, "High-Level Digitized Sound Drivers", presents descriptions of the library functions for the Creative High-Level Digitized Sound Drivers.

Chapter 2, "High-Level Auxiliary Driver", presents descriptions of the library functions for the AUXDRV driver.

Chapter 3, "Creative Multimedia System Driver", presents descriptions of the functions for the Creative low-level digitized sound I/O, auxiliary services and signal processing services.

Chapter 4, "MIDI Driver", presents descriptions of the library functions for the Creative MIDI driver.

Chapter 5, "CD-ROM Audio", presents descriptions of the library functions for CD‑ROM audio interface.

Appendix A "File Format", describes the Creative Digitized sound File (.VOC) format and the Creative ADPCM wave type registered by Creative.

Document Conventions

To help you to locate and identify information easily, this manual uses visual cues and standard text formats. The following typographic conventions are used throughout this manual:

|  |  |
| --- | --- |
| **Example** | **Description** |
| **voice\_drv,**  **ctvdOutput** | Bold letters indicate variable names, library functions or commands. These are case-sensitive (i.e. upper and lower case are significant). Bold letters are also used for keywords or for emphasis in certain words. |
| **MIXERVOL\_CD** | Bold all capital letters indicate manifest constant. |
| CT-VOICE.DRV | All capital letters indicate file names, directory names or constants. |
| *placeholders* | Italic letters represent actual values or variables that you are expected to provide. |
| ctrl+enter | Small capital letters signify names of keys on the keyboard. Notice that a plus (+) indicates a combination of keys. For example, ctrl+enter means to hold down the ctrl key while pressing the enter key. |
| program | This font is used for example codes. |
| program  .  .  .  fragment | Vertical ellipsis in an example program indicate that part of the program has been intentionally omitted. |
| [ ] | Square brackets in a command line indicate that the enclosed item is optional. It should not be typed verbatim. |
| < > | Angle brackets in a command line indicate that you must provide the actual value of the enclosed item. It should not be typed verbatim. |
| / | Slash in a command line indicates an either/or choice. It should not be typed verbatim. |
| Sound Blaster Pro  (SBPRO) | Acronyms are usually spelled out the first time they are used. |

Function Reference Document Format

The following format is used for the library function documentations:

**Action** Gives a brief description of the function.

**Syntax** Lists the declaration syntax for the function; parameter names are italicized.

**Parameters** Describes the function parameters.

**Remarks** Gives a more detailed description of the function and how it is used.

**Return Value** Describes the value returned by the function (if any).

**See Also** Lists the functions (if any) that are related to the function being described.

**ASM Interface** Lists the corresponding assembly language interface functions.

Chapter 1

High-Level Digitized Sound Drivers

This chapter documents the interfaces to the following Creative high-level loadable digitized sound drivers:

|  |  |
| --- | --- |
| CT-VOICE.DRV | .VOC memory driver |
| CTVDSK.DRV | .VOC disk double buffering driver |
| CTWMEM.DRV | .WAV memory driver |
| CTWDSK.DRV | .WAV disk double buffering driver |

It is divided into two parts. The first part covers high-level language interfaces and the second, register base interfaces. Cross-references to the register base interfaces are provided from the high-level language interfaces.

The interfaces require a minimum driver version number of 4.00.

High-Level Language Interface

This section describes the high-level language interface to the digitized sound drivers.

The high-level language interfaces of the digitized sound drivers have deliberately been kept very similar. Essentially we hope that if you understand one driver well, you will practically have mastered them all. Hence, the high-level language interface in this chapter has been organized in such a way as to highlight the considerable similarities between them.

In the following illustrations, we use ? as a wildcard. For example, if you see **ct?dInput**. It refers to the functions **ctvdInput** and **ctwdInput**.

Function Prefixes

High-level digitized sound function names begin with the following prefixes:

|  |  |
| --- | --- |
| **Prefix** | **Driver** |
| ctvm | CT-VOICE |
| ctvd | CTVDSK |
| ctwm | CTWMEM |
| ctwd | CTWDSK |

Include Files‑

The followings are the required include files for the digitized sound drivers:

|  |  |  |  |
| --- | --- | --- | --- |
| **Driver** | **C Language** | **Turbo Pascal** | **Microsoft Basic** |
| CT-VOICE | SBKVOICE.H | SBKVOICE.INC | SBKVOICE.BI |
| CTVDSK | SBKVOICE.H | SBKVOICE.INC | SBKVOICE.BI |
| CTWMEM | SBKWAVE.H | SBKWAVE.INC | SBKWAVE.BI |
| CTWDSK | SBKWAVE.H | SBKWAVE.INC | SBKWAVE.BI |

Functions by Category

The digitized sound driver functions may be divided into the following categories:

|  |  |
| --- | --- |
| **Category** | **Functions** |
|  |  |
| Initialization/Termination | **ct*??*GetEnvSettings** |
|  | **ct*??*Init** |
|  | **ct*??*Terminate** |
|  |  |
|  |  |
| Input | **ct*?*dInput** |
|  | **ct*?*mInputCM** |
|  | **ct*?*mInputXM** |
|  |  |
|  |  |
| Output | **ct*?*dOutput** |
|  | **ctvdOutputOffset** |
|  | **ct*?*mOutputCM** |
|  | **ct*?*mOutputXM** |
|  |  |
|  |  |
| I/O control | **ctv*?*BreakLoop** |
|  | **ct*??*Continue** |
|  | **ct*??*Pause** |
|  | **ct*?*dSetDiskBuffer** |
|  | **ct*??*SetDMABuffer** |
|  | **ct*??*SetIOParam** |
|  | **ct*??*Stop** |
|  |  |
|  |  |
| Query | **ct*?*dGetDrvError** |
|  | **ct*?*dGetExtError** |
|  | **ct*??*GetIOParam** |
|  | **ct*??*GetParam** |
|  |  |
|  |  |
| Miscellaneous | **ct*??*SetSpeaker** |
|  |  |
|  |  |

.VOC **BreakLoop** functions

**Action** Breaks out from a repeat loop in the current digitized sound output process.

**Syntax** **C** **WORD ctvmBreakLoop( WORD** *wIOHandle***,****WORD** *wBreakMode***)**

**WORD ctvdBreakLoop( WORD** *wIOHandle***,WORD** *wBreakMode* **)**

**Pascal** **ctvmBreakLoop(** *wIOHandle***,** *wBreakMode***:word ) :word**

**ctvdBreakLoop(** *wIOHandle***,** *wBreakMode***:word ) :word**

**Basic** **ctvmBreakLoop%(** *wIOHandle%***,** *wBreakMode%* **)**

**ctvdBreakLoop%(** *wIOHandle%***,** *wBreakMode%* **)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

wBreakMode

Break-out method.

**Remarks** If *wBreakMode* is zero, the driver will complete the current repeat loop before breaking out. If *wBreakMode* is non-zero, the break-out takes place immediately. In both cases, the driver will proceed to the block immediately following the ‘End Repeat Loop’ block.

**Return Value** Zero if successful. Non-zero if the digitized sound output was not in a loop.

See Also Output functions

ASM Interface ctvmBreakLoop: CT-VOICE.DRV Function 40

ctvdBreakLoop: CTVDSK.DRV Function 37

**Continue** functions

**Action** Continues the paused digitized sound output process.

**Syntax C** **WORD ctvmContinue( WORD** *wIOHandle***)**

**WORD ctvdContinue( WORD** *wIOHandle***)**

**WORD ctwmContinue( WORD** *wIOHandle* **)**

**WORD ctwdContinue( WORD** *wIOHandle***)**

**Pascal** **ctvmContinue(** *wIOHandle* **:word) :word**

**ctvdContinue(** *wIOHandle***:word ) :word**

**ctwmContinue(** *wIOHandle* **:word ) :word**

**ctwdContinue(** *wIOHandle***:word ) :word**

**Basic** **ctvmContinue%(** *wIOHandle%***)**

**ctvdContinue%(** *wIOHandle%***)**

**ctwmContinue%(** *wIOHandle%* **)**

**ctwdContinue%(** *wIOHandle%***)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

**Remarks** None

**Return Value** Zero if successful. Non-zero if the digitized sound output was not in a paused state.

See Also Pause functions

ASM Interface ctvmContinue: CT-VOICE.DRV Function 39

ctvdContinue: CTVDSK.DRV Function 36

ctwmContinue: CTWMEM.DRV Function 14

ctwdContinue: CTWDSK.DRV Function 14

**GetDrvError** functions

**Action** Gets the driver's error code for the last operation.

**Syntax C WORD ctvmGetDrvError( void )**

**WORD ctvdGetDrvError( void )**

**WORD ctwmGetDrvError( void )**

**WORD ctwdGetDrvError( void )**

**Pascal** **ctvmGetDrvError :word**

**ctvdGetDrvError :word**

**ctwmGetDrvError :word**

**ctwdGetDrvError :word**

**Basic** **ctvmGetDrvError%( )**

**ctvdGetDrvError%( )**

**ctwmGetDrvError%( )**

**ctwdGetDrvError%( )**

**Parameters** None

**Remarks** If the error involves DOS, invoke the corresponding **ct*?*dGetExtError** function to obtain the error code returned by the failed DOS call.

**Return Value** Refer to the following table for the error codes:

|  |  |
| --- | --- |
| **Error Code** | **Meaning** |
| 1 | DOS memory allocation error |
| 2 | Another digitized sound I/O process is currently active |
| 3 | DOS read file error |
| 4 | DOS write file error |
| 5 | DOS lseek error on file |
| 6 | Disk full |
| 7 | Invalid file format |
| 8 | Disk buffer is not allocated |
| 9 | DOS open file error |
| 10 | Sound card I/O error |
| 11 | Incorrect driver version |
| 12 | IRQ error |
| 13 | Not Sound Blaster series hardware |
| 14 | Creative DSP copyright message error |
| 15 | DMA buffer is not allocated |
| 16 | 8-bit DMA error |
| 17 | 16-bit DMA error |
| 18 | Invalid I/O handle |
| 19 | Null pointer passed |
| 20 | Invalid parameter |
| 21 | Environment string error |
| 22 | Low-level driver error |
| 23 | Extended memory error |
| 24 | Incorrect buffer size |
| 25 | DMA buffer crosses page boundary |
| 26 | Driver not initialized |
| 27 | Sound device is not active |

See Also Disk GetExtError functions

ASM Interface ctvmGetDrvError: CT-VOICE.DRV Function 41

ctvdGetDrvError: CTVDSK.DRV Function 14

ctwmGetDrvError: CTWMEM.DRV Function 15

ctwdGetDrvError: CTWDSK.DRV Function 6

**GetEnvSettings** functions

**Action** Passes the BLASTER environment string to the driver for it to interpret the hardware settings to use.

**Syntax C WORD ctvmGetEnvSettings( const char far \****lpszBlaster***)**

**WORD ctvdGetEnvSettings( const char far \****lpszBlaster***)**

**WORD ctwmGetEnvSettings( const char far \****lpszBlaster***)**

**WORD ctwdGetEnvSettings( const char far \****lpszBlaster***)**

**Pascal ctvmGetEnvSettings(** *lpszBlaster***:pointer ) :word**

**ctvdGetEnvSettings(** *lpszBlaster***:pointer ) :word**

**ctwmGetEnvSettings(** *lpszBlaster***:pointer ) :word**

**ctwdGetEnvSettings(** *lpszBlaster***:pointer ) :word**

**Basic ctvmGetEnvSettings%(** *lpszBlaster&***)**

**ctvdGetEnvSettings%(** *lpszBlaster&***)**

**ctwmGetEnvSettings%(** *lpszBlaster&***)**

**ctwdGetEnvSettings%(** *lpszBlaster&***)**

**Parameters** *lpszBlaster*

Far pointer to the BLASTER environment string, without the "BLASTER=" prefix.

**Remarks** This function must be called first, even before the corresponding **ct*??*Init** function.

**Return Value** Zero if successful. Otherwise, a bit-or'ed combination of the values listed in the table below is returned:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 0x0001 | *lpszBlaster* is NULL, or points to an empty string |
| 0x0002 | Base I/O address not specified, or is out of range |
| 0x0004 | IRQ number not specified, or is out of range |
| 0x0008 | 8-bit DMA channel not specified, or is out of range |
| 0x0010 | 16-bit DMA channel not specified, or is out of range |

See Also Init functions

ASM Interface ctvmGetEnvSettings: CT-VOICE.DRV Function 28

ctvdGetEnvSettings: CTVDSK.DRV Function 26

ctwmGetEnvSettings: CTWMEM.DRV Function 0

ctwdGetEnvSettings: CTWDSK.DRV Function 0

Disk **GetExtError** functions

**Action** Gets the error code returned by the failed DOS call for a driver I/O operation.

**Syntax C WORD ctvdGetExtError( void )**

**WORD ctwdGetExtError( void )**

**Pascal ctvdGetExtError :word**

**ctwdGetExtError :word**

**Basic ctvdGetExtError%( )**

**ctwdGetExtError%( )**

**Parameters** None

**Remarks** This function should only be called if the return value from **ct*??*GetDrvError** indicates a DOS error.

**Return Value** Refer to the DOS Technical Reference manual for the error code descriptions.

See Also GetDrvError functions

ASM Interface ctvdGetExtError: CTVDSK.DRV Function 14

ctwdGetExtError: CTWDSK.DRV Function 6

**GetIOParam** functions

**Action** Gets the value of an I/O parameter.

**Syntax C WORD ctvmGetIOParam( WORD** *wIOHandle***,**

**WORD** *wParamType***,**

**DWORD far \****lpdwParam***)**

**WORD ctvdGetIOParam( WORD** *wIOHandle***,**

**WORD** *wParamType***,**

**DWORD far \****lpdwParam***)**

**WORD ctwmGetIOParam( WORD** *wIOHandle***,**

**WORD** *wParamType***,**

**DWORD far \****lpdwParam***)**

**WORD ctwdGetIOParam( WORD** *wIOHandle***,**

**WORD** *wParamType*,

**DWORD far \****lpdwParam***)**

**Pascal ctvmGetIOParam(** *wIOHandle***,**  *wParamType* **:word;**

**var** *lpdwParam***:longint ) :word**

**ctvdGetIOParam(** *wIOHandle***,**  *wParamType* **:word;**

**var** *lpdwParam***:longint ) :word**

**ctwmGetIOParam(** *wIOHandle***,**  *wParamType* **:word;**

**var** *lpdwParam***:longint ) :word**

**ctwdGetIOParam(** *wIOHandle***,**  *wParamType* **:word;**

**var** *lpdwParam***:longint ) :word**

**Basic ctvmGetIOParam%(** *wIOHandle%***,** *wParamType%***,**

**SEG** *lpdwParam&***)**

**ctvdGetIOParam%(** *wIOHandle%***,** *wParamType%***,**

**SEG** *lpdwParam&***)**

**ctwmGetIOParam%(** *wIOHandle%***,** *wParamType%***,**

**SEG** *lpdwParam&***)**

**ctwdGetIOParam%(** *wIOHandle%***,** *wParamType%***,**

**SEG** *lpdwParam&***)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

wParamType

Specifies the I/O parameter value to retrieve. Refer to **ct*??*SetIOParam** for the set of valid constants.

lpdwParam

Far pointer to a double-word storage for the value of the specified I/O parameter.

**Remarks** None

**Return Value** Zero if successful. Non-zero otherwise.

See Also SetIOParam functions

ASM Interface ctvmGetIOParam: CT-VOICE.DRV Function 32

ctvdGetIOParam: CTVDSK.DRV Function 31

ctwmGetIOParam: CTWMEM.DRV Function 7

ctwdGetIOParam: CTWDSK.DRV Function 9

.VOC **GetParam** functions

**Action** Gets information on the driver and sound card.

**Syntax C WORD ctvmGetParam( WORD** *wParamType***,**

**DWORD far \****lpdwParam* **)**

**WORD ctvdGetParam( WORD** *wParamType***,**

**DWORD far \****lpdwParam***)**

**Pascal ctvmGetParam(** *wParamType* **:word;**

**var** *lpdwParam* **:longint ) :word**

**ctvdGetParam(** *wParamType***:word;**

**var** *lpdwParam* **:longint ) :word**

**Basic ctvmGetParam%(** *wParamType%***,**

**SEG** *lpdwParam&* **)**

**ctvdGetParam%(** *wParamType%***,**

**SEG** *lpdwParam&***)**

**Parameters** *wParamType*

Specifies the parameter value to retrieve.

lpdwParam

Far pointer to a double-word storage for the value of the specified parameter.

**Remarks** This function may be called before **ct*??*Init**.

*wParamType* must be one of the following constants:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **CTVOC\_DRIVERVERSION** | Driver version |
| **CTVOC\_CARDTYPE** | Card type number |
| **CTVOC\_LPCARDNAME** | Card name |
| **CTVOC\_INPUTCHANNELS** | Number of input channels |
| **CTVOC\_OUTPUTCHANNELS** | Number of output channels |
| **CTVOC\_DRIVERSIZE** | Driver size, less embedded DMA buffer |
| **CTVOC\_INPUTHANDLES** | Number of input handles supported |
| **CTVOC\_OUTPUTHANDLES** | Number of output handles supported |
| **CTVOC\_DRIVERBUILD** | Driver build number |
| **CTVOC\_EMBDDMABUFSIZE** | Size of embedded DMA buffer |
| **CTVOC\_SAMPLINGRANGE** | Sampling rate limits |
| **CTVOC\_NEEDDMABUFFER** | Require a DMA transfer buffer |

Details on the data written into the DWORD-storage pointed to by *lpdwParam*:

**CTVOC\_DRIVERVERSION**

Byte 1 is assigned the major version number. Byte 0 is assigned the minor version number. The high-word is set to zero.

All DOS .VOC or .WAV drivers supporting the new API will have version numbers greater than or equal to 0x0400.

**CTVOC\_CARDTYPE**

Use this constant to obtain the card type number supported by the driver in use.

**CTVOC\_LPCARDNAME**

The DWORD is assigned a far pointer to an ASCIIZ string giving the name of the supported sound card.

**CTVOC\_INPUTCHANNELS**

**CTVOC\_OUTPUTCHANNELS**

The DWORD is assigned 1 or 2, depending on whether the sound card is capable of supporting only 1 (mono) or 2 (mono and stereo) channels.

**CTVOC\_DRIVERSIZE**

The DWORD is assigned the size (in bytes) of the driver, less the embedded DMA buffer.

This information allows you to resize the memory block allocated for holding the driver, so as to reclaim some memory when you wish to allocate your own DMA buffer. This is useful when the default DMA buffer is deemed too small and you need to assign a larger buffer to the driver with **ctv*?*SetDMABuffer**.

See remarks under **CTVOC\_EMBDDMABUFSIZE** for information on how to retrieve the size of the embedded DMA buffer.

**CTVOC\_INPUTHANDLES**

**CTVOC\_OUTPUTHANDLES**

The DWORD is assigned the number of input or output handles supported by the driver.

Invoke all input or output functions with a handle in the range zero to one less than this returned value.

**CTVOC\_DRIVERBUILD**

The low-word is assigned the build number of the driver. The high-word is set to zero.

**CTVOC\_EMBDDMABUFSIZE**

The DWORD is assigned the size of the embedded DMA buffer, in units of 2KB per half-buffer.

See remarks under **CTVOC\_DRIVERSIZE** for information on the use of this parameter value.

**CTVOC\_SAMPLINGRANGE**

In order to obtain the sampling range, you have to specify whether a minimum/maximum, mono/stereo, ADC/DAC sampling limit is to be retrieved. Before this function is called, these 3 parameters must be set in the DWORD pointed to by *lpdwParam*, in the following manner:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Byte** | **Value** | **Meaning** |  |  |
| 0 | 0  1 | To obtain | minimum  maximum | sampling rate. |
| 1 | 1  2 | To obtain | mono  stereo | sampling rate. |
| 2 | 0  1 | To obtain | ADC  DAC | sampling rate. |
| 3 | 0 | This byte must be zero. | | |

On return from this function, the DWORD-storage is assigned the requested sampling rate limit.

**CTVOC\_NEEDDMABUFFER**

Boolean value. TRUE indicates a DMA transfer buffer is needed.

**Return Value** Zero if successful. Non-zero otherwise.

See Also None

ASM Interface ctvmGetParam: CT-VOICE.DRV Function 29

ctvdGetParam: CTVDSK.DRV Function 27

.WAV **GetParam** functions

**Action** Gets information on the driver and sound card.

**Syntax C WORD ctwmGetParam( WORD** *wParamType***,**

**DWORD far \****lpdwParam* **)**

**WORD ctwdGetParam( WORD** *wParamType***,**

**DWORD far \****lpdwParam***)**

**Pascal ctwmGetParam(** *wParamType* **:word;**

**var** *lpdwParam* **:longint ) :word**

**ctwdGetParam(** *wParamType***:word;**

**var** *lpdwParam* **:longint ) :word**

**Basic ctwmGetParam%(** *wParamType%*,

**SEG** *lpdwParam&* **)**

**ctwdGetParam%(** *wParamType%*,

**SEG** *lpdwParam&***)**

**Parameters** *wParamType*

Specifies the parameter value to retrieve.

lpdwParam

Far pointer to a double-word storage for the value of the specified parameter.

**Remarks** *wParamType* must be one of the following constants:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **CTWAV\_DRIVERVERSION** | Driver version |
| **CTWAV\_CARDTYPE** | Card type number |
| **CTWAV\_LPCARDNAME** | Card name |
| **CTWAV\_INPUTCHANNELS** | Number of input channels |
| **CTWAV\_OUTPUTCHANNELS** | Number of output channels |
| **CTWAV\_INPUTHANDLES** | Number of input handles supported |
| **CTWAV\_OUTPUTHANDLES** | Number of output handles supported |
| **CTWAV\_DRIVERBUILD** | Driver build number |
| **CTWAV\_SAMPLINGRANGE** | Sampling rate limits |
| **CTWAV\_NEEDDMABUFFER** | Require a DMA transfer buffer |

Details on the data written into the DWORD-storage pointed to by *lpdwParam*:

**CTWAV\_DRIVERVERSION**

Byte 1 is assigned the major version number. Byte 0 is assigned the minor version number. The high-word is set to zero.

All DOS .VOC or .WAV drivers supporting the new API will have version numbers greater than or equal to 0x0400.

**CTWAV\_CARDTYPE**

Use this constant to obtain the card type number supported by the driver in use.

**CTWAV\_LPCARDNAME**

The DWORD is assigned a far pointer to an ASCIIZ string giving the name of the supported sound card.

**CTWAV\_INPUTCHANNELS**

**CTWAV\_OUTPUTCHANNELS**

The DWORD is assigned 1 or 2, depending on whether the sound card is capable of supporting only 1 (mono) or 2 (mono and stereo) channels.

**CTWAV\_INPUTHANDLES**

**CTWAV\_OUTPUTHANDLES**

The DWORD is assigned the number of input or output handles supported by the driver.

Invoke all input or output functions with a handle in the range zero to one less than this returned value.

**CTWAV\_DRIVERBUILD**

The low-word is assigned the build number of the driver. The high-word is set to zero.

**CTWAV\_SAMPLINGRANGE**

In order to obtain the sampling range, you have to specify whether a minimum/maximum, mono/stereo, ADC/DAC sampling limit is to be retrieved. Before this function is called, these 3 parameters must be set in the DWORD pointed to by *lpdwParam*, in the following manner:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Byte** | **Value** | **Meaning** |  |  |
| 0 | 0  1 | To obtain | minimum  maximum | sampling rate. |
| 1 | 1  2 | To obtain | mono  stereo | sampling rate. |
| 2 | 0  1 | To obtain | ADC  DAC | sampling rate. |
| 3 | 0 | This byte must be zero. | | |

On return from this function, the DWORD-storage is assigned the requested sampling rate limit.

**CTWAV\_NEEDDMABUFFER**

Boolean value. TRUE indicates a DMA transfer buffer is needed.

**Return Value** Zero if successful. Non-zero otherwise.

See Also None

ASM Interface ctwmGetParam: CTWMEM.DRV Function 1

ctwdGetParam: CTWDSK.DRV Function 1

**Init** functions

**Action** Initializes the driver and the sound card.

**Syntax C WORD ctvmInit( void )**

**WORD ctvdInit( void )**

**WORD ctwmInit( void )**

**WORD ctwdInit( void )**

**Pascal ctvmInit :word**

**ctvdInit :word**

**ctwmInit :word**

**ctwdInit :word**

**Basic ctvmInit%( )**

**ctvdInit%( )**

**ctwmInit%( )**

**ctwdInit%( )**

**Parameters** None

**Remarks** Do not forget to call the corresponding **ct*??*GetEnvSettings** before calling this function.

The **ctvdInit** and **ctwdInit** functions will hook the Timer interrupt (INT 8h), Video interrupt (INT 10h), Disk interrupt (INT 13h) and DOS Idle interrupt (INT 28h) during a successful initialization.

**Return Value** Zero if successful. Non-zero otherwise.

See Also GetDrvError functions

**GetEnvSettings** functions

**Terminate** functions

ASM Interface ctvmInit: CT-VOICE.DRV Function 3

ctvdInit: CTVDSK.DRV Function 38

ctwmInit: CTWMEM.DRV Function 2

ctwdInit: CTWDSK.DRV Function 3

**Input** functions

**Action** Starts digitized sound input.

**Syntax C WORD ctvmInputCM( WORD** *wIOHandle***,**

**BYTE far \****lpBuf***,** **DWORD** *dwBufLen***)**

**WORD ctvmInputXM( WORD** *wIOHandle***,** **WORD** *wXMBHandle***,**

**DWORD** *dwXMBOffset***,** **WORD** *wKBBufferSize* **)**

**WORD ctwmInputCM( WORD** *wIOHandle***,**

**BYTE far \****lpBuf***,** **DWORD** *dwBufLen* **)**

**WORD ctwmInputXM( WORD** *wIOHandle***, WORD** *wXMBHandle***,**

**DWORD** *dwXMBOffset***,** **WORD** *wKBBufferSize* **)**

**WORD ctvdInput( WORD** *wIOHandle***, WORD** *wFileHandle***)**

**WORD ctwdInput( WORD** *wIOHandle***, WORD** *wFileHandle***)**

**Pascal ctvmInputCM(** *wIOHandle* **:word;** *lpBuf* **:pointer;**

*dwBufLen***:longint ) :word**

**ctvmInputXM(** *wIOHandle***,** *wXMBHandle* **:word;**

*dwXMBOffset* **:longint;**

*wKBBufferSize* **:word ) :word**

**ctwmInputCM(** *wIOHandle* **:word;** *lpBuf* **:pointer;**

*dwBufLen* **:longint ) :word**

**ctwmInputXM(** *wIOHandle***,** *wXMBHandle* **:word;**

*dwXMBOffset* **:longint;**

*wKBBufferSize* **:word ) :word**

**ctvdInput(** *wIOHandle***,** *wFileHandle***:word ) :word**

**ctwdInput(** *wIOHandle***,** *wFileHandle***:word ) :word**

**Basic ctvmInputCM%(** *wIOHandle%***,** *lpBuf&***,** *dwBufLen&***)**

**ctvmInputXM%(** *wIOHandle%***,** *wXMBHandle%***,**

*dwXMBOffset&***,** *wKBBufferSize%* **)**

**ctwmInputCM%(** *wIOHandle%***,** *lpBuf%***,** *dwBufLen&* **)**

**ctwmInputXM%(** *wIOHandle%***,** *wXMBHandle%***,**

*dwXMBOffset&***,** *wKBBufferSize%* **)**

**ctvdInput(** *wIOHandle%*,*wFileHandle%***)**

**ctwdInput(** *wIOHandle%*, *wFileHandle%***)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

lpBuf

Far pointer to a conventional memory buffer.

dwBufLen

Length of the conventional memory buffer to use for recording. Measured in bytes.

wXMBHandle

Handle of an extended memory block.

dwXMBOffset

Starting offset within the extended memory block.

wKBBufferSize

Size of the extended memory block to use for recording. Measured in kilobytes.

wFileHandle

DOS file handle of a file opened/created with write access.

**Remarks** Here's how to differentiate among the six functions:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **File Format** | | **Recording Destination** | | |
| **Function** | .VOC | .WAV | Conventional  Memory | Extended  Memory | Disk  File |
| **ctvmInputCM** |  |  |  |  |  |
| **ctvmInputXM** |  |  |  |  |  |
| **ctwmInputCM** |  |  |  |  |  |
| **ctwmInputXM** |  |  |  |  |  |
| **ctvdInput** |  |  |  |  |  |
| **ctwdInput** |  |  |  |  |  |

**ctvmInputCM** and **ctvmInputXM** do not create a .VOC File Header Block in front of the Digitized sound Data Block. It is your responsibility to add the File Header Block if you wish to save it as a disk .VOC file.

**ctvdInput** will create a .VOC File Header Block in front of the Digitized sound Data Block.

The three **ctw*?*Input*??*** functions will create a .WAV File Header before the data.

The recording parameters must be set up, using **ct*??*SetIOParam**, before invoking this function.

After initiating the recording, this function returns to your program immediately. The recording itself takes place in the background.

The digitized sound status word corresponding to the specified I/O handle is set to 0xFFFF during the recording. It is set to zero when the recording is terminated such as buffer has been filled up or stop functions is invoked..

**Return Value** Zero if successful. Non-zero otherwise.

See Also GetDrvError functions if interest is in ct*?*dInput

**SetIOParam** functions

**Stop** functions

ASM Interface ctvmInputCM: CT-VOICE.DRV Function 33

ctvmInputXM: CT-VOICE.DRV Function 34

ctwmInputCM: CTWMEM.DRV Function 8

ctwmInputXM: CTWMEM.DRV Function 9

ctvdInput: CTVDSK.DRV Function 32

ctwdInput: CTWDSK.DRV Function 12

**Output** functions

**Action** Starts digitized sound output.

**Syntax C WORD ctvmOutputCM( WORD** *wIOHandle***, BYTE far \****lpBuf***)**

**WORD ctvmOutputXM( WORD** *wIOHandle***,** **WORD** *wXMBHandle***,**

**DWORD** *dwXMBOffset* **)**

**WORD ctwmOutputCM( WORD** *wIOHandle***,** **BYTE far \****lpBuf* **)**

**WORD ctwmOutputXM( WORD** *wIOHandle***, WORD** *wXMBHandle***,**

**DWORD** *dwXMBOffset* **)**

**WORD ctvdOutput( WORD** *wIOHandle***, WORD** *wFileHandle***)**

**WORD ctwdOutput( WORD** *wIOHandle***, WORD** *wFileHandle***)**

**Pascal ctvmOutputCM(** *wIOHandle* **:word;** *lpBuf* **:pointer ) :word**

**ctvmOutputXM(** *wIOHandle***,** *wXMBHandle* **:word;**

*dwXMBOffset* **:longint ) :word**

**ctwmOutputCM(** *wIOHandle* **:word;** *lpBuf* **:pointer ) :word**

**ctwmOutputXM(** *wIOHandle***,** *wXMBHandle* **:word;**

*dwXMBOffset* **:longint ) :word**

**ctvdOutput(** *wIOHandle***,** *wFileHandle***:word ) :word**

**ctwdOutput(** *wIOHandle***,** *wFileHandle***:word ) :word**

**Basic ctvmOutputCM%(** *wIOHandle%***,** *lpBuf&***)**

**ctvmOutputXM%(** *wIOHandle%***,** *wXMBHandle%***,**

*dwXMBOffset&* **)**

**ctwmOutputCM%(** *wIOHandle%***,** *lpBuf%* **)**

**ctwmOutputXM%(** *wIOHandle%***,** *wXMBHandle%***,**

*dwXMBOffset&* **)**

**ctvdOutput(** *wIOHandle%*,*wFileHandle%***)**

**ctwdOutput(** *wIOHandle%*, *wFileHandle%***)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

lpBuf

Far pointer to a conventional memory buffer.

wXMBHandle

Handle of an extended memory block.

dwXMBOffset

Starting offset within the extended memory block.

wFileHandle

DOS file handle.

**Remarks** Here's how to differentiate among the six functions:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **File Format** | | **Playback Data Source** | | |
| **Function** | .VOC | .WAV | Conventional  Memory | Extended  Memory | Disk  File |
| **ctvmOutputCM** |  |  |  |  |  |
| **ctvmOutputXM** |  |  |  |  |  |
| **ctwmOutputCM** |  |  |  |  |  |
| **ctwmOutputXM** |  |  |  |  |  |
| **ctvdOutput** |  |  |  |  |  |
| **ctwdOutput** |  |  |  |  |  |

**ctvmOutputCM** must be called with *lpBuf* pointing to the first Data Block. Similarly, **ctvmOutputXM** must be called with *dwXMBOffset* giving the starting location of the first .VOC Data Block within the extended memory block.

With the other four functions, you must have the buffer pointer (or starting offset, or file pointer, whichever the case may be) pointing to the start of the file.

After initiating the playback, this function returns to your program immediately. The playback itself takes place in the background.

The digitized sound status word corresponding to the specified I/O handle is set to 0xFFFF during the playback. It is set to zero when the playback is terminated.

**Return Value** Zero if successful. Non-zero otherwise.

See Also GetDrvError functions if interest is in ct*?*dOutput

**Continue** functions

**Pause** functions

**Stop** functions

ASM Interface ctvmOutputCM: CT-VOICE.DRV Function 35

ctvmOutputXM: CT-VOICE.DRV Function 36

ctwmOutputCM: CTWMEM.DRV Function 10

ctwmOutputXM: CTWMEM.DRV Function 11

ctvdOutput: CTVDSK.DRV Function 33

ctwdOutput: CTWDSK.DRV Function 11

.VOC Disk **OutputOffset** functions

**Action** Starts digitized sound from an offset of a disk file.

**Syntax C WORD ctvdOutputOffset( WORD** *wIOHandle***, WORD** *wFileHandle*

**LONG***lOffsetFromCurFilePos* **)**

**Pascal ctvdOutputOffset(** *wIOHandle***,** *wFileHandle* **:word;**

*lOffsetFromCurFilePos* **:longint ) :word**

**Basic ctvdOutputOffset%(** *wIOHandle%***,** *wFileHandle%***,**

*lOffsetFromCurFilePos&* **)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

wFileHandle

DOS file handle.

lOffsetFromCurFilePos

Offset of the file position relative to current file position in unit of bytes.

**Remarks** Contrast with the **ctvdOutput**, it starts a digitized sound from a specified offset of a disk file.

Positive value of *lOffsetFromCurFilePos* means offset forward and negative means offset backward.

The application is responsible to ensure that the offset specified is a start of a .VOC block.

**Return Value** Zero if successful. Non-zero otherwise.

See Also Output functions

**GetDrvError** functions

ASM Interface ctvdOutputOffset: CTVDSK.DRV Function 39

**Pause** functions

**Action** Pauses the active digitized sound output process.

**Syntax C WORD ctvmPause( WORD** *wIOHandle***)**

**WORD ctvdPause( WORD** *wIOHandle* **)**

**WORD ctwmPause( WORD** *wIOHandle* **)**

**WORD ctwdPause( WORD** *wIOHandle* **)**

**Pascal ctvmPause(** *wIOHandle* **:word ) :word**

**ctvdPause(** *wIOHandle* **:word ) :word**

**ctwmPause(** *wIOHandle* **:word ) :word**

**ctwdPause(** *wIOHandle* **:word ) :word**

**Basic ctvmPause%(** *wIOHandle%***)**

**ctvdPause%(** *wIOHandle%* **)**

**ctwmPause%(** *wIOHandle%* **)**

**ctwdPause%(** *wIOHandle%* **)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

**Remarks** The digitized sound status word remain unchanged.

**Return Value** Zero if successful. Non-zero otherwise.

See Also Continue functions

**Stop** functions

ASM Interface ctvmPause: CT-VOICE.DRV Function 38

ctvdPause: CTVDSK.DRV Function 35

ctwmPause: CTWMEM.DRV Function 13

ctwdPause: CTWDSK.DRV Function 13

Disk **SetDiskBuffer** functions

**Action** Sets up the disk buffer for an I/O handle.

**Syntax C WORD ctvdSetDiskBuffer( WORD** *wIOHandle***, BYTE far \****lpBuffer***,**

**WORD** *w2KBHalfBufferSize* **)**

**WORD ctwdSetDiskBuffer( WORD** *wIOHandle***, BYTE far \****lpBuffer***,**

**WORD** *w2KBHalfBufferSize* **)**

**Pascal ctvdSetDiskBuffer(** *wIOHandle* **:word;** *lpBuffer* **:pointer;**

*w2KBHalfBufferSize* **:word ) :word**

**ctwdSetDiskBuffer(** *wIOHandle* **:word;** *lpBuffer* **:pointer;**

*w2KBHalfBufferSize* **:word ) :word**

**Basic ctvdSetDiskBuffer%(** *wIOHandle%***,** *lpBuffer&***,**

*w2KBHalfBufferSize%* **)**

**ctwdSetDiskBuffer%(** *wIOHandle%***,** *lpBuffer&***,**

*w2KBHalfBufferSize%* **)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

lpBuffer

Far pointer to the user-allocated disk buffer.

w2KBHalfBufferSize

The disk buffer size, in units of 2KB per half-buffer.

**Remarks** Apart from requiring one DMA buffer per I/O handle, the disk drivers also require a corresponding disk buffer for digitized sound I/O. The disk buffer serves as an intermediate storage for the digitized sound data before it is transferred to the DMA buffer. Hence the disk buffer must be at least twice the size of the DMA buffer.

As an example, if *w2KBHalfBufferSize* is 4, it means a 16KB disk buffer is to be used. Consequently, the *w2KBHalfBufferSize* for the DMA buffer can be no more than 2.

This two-layer buffered approach will help achieve smoother I/O at high sampling rates, provided the disk buffer is large enough.

To illustrate, take digitized sound output for example. The digitized sound data is pre-loaded into the disk buffer before being transferred to the DMA buffer at each DSP interrupt. If the disk buffer is larger, it will be able to hold more data, thus reducing the frequency of disk access.

You must allocate a disk buffer and call this function before starting any digitized sound I/O process.

The recommended range for *w2KBHalfBufferSize* is from 2 to 32.

No digitized sound I/O can be carried out until a valid disk buffer has been set up.

If the allocated buffer does not start on a paragraph (16-byte) boundary, you must allocate the disk buffer with an additional 15 bytes, because the driver will do a paragraph adjustment internally.

**Return Value** Zero if successful. Non-zero otherwise.

See Also SetDMABuffer functions

ASM Interface ctvdSetDiskBuffer: CTVDSK.DRV Function 28

ctwdSetDiskBuffer: CTWDSK.DRV Function 2

**SetDMABuffer** functions

**Action** Sets up the DMA buffer for an I/O handle.

**Syntax C WORD ctvmSetDMABuffer( WORD** *wIOHandle***,**

**DWORD** *dw32BitAddx***,**

**WORD** w2KBHalfBufferSize **)**

**WORD ctvdSetDMABuffer( WORD** *wIOHandle***,**

**DWORD** *dw32BitAddx***,**

**WORD** *w2KBHalfBufferSize* **)**

**WORD ctwmSetDMABuffer( WORD** *wIOHandle***,**

**DWORD** *dw32BitAddx***,**

**WORD** *w2KBHalfBufferSize* **)**

**WORD ctwdSetDMABuffer( WORD** *wIOHandle***,**

**DWORD** *dw32BitAddx***,**

**WORD** w2KBHalfBufferSize **)**

**Pascal ctvmSetDMABuffer(** *wIOHandle* **:word;** *dw32BitAddx* **:longint;**

*w2KBHalfBufferSize* **:word ) :word**

**ctvdSetDMABuffer(** *wIOHandle* **:word;** *dw32BitAddx* **:longin**t;

*w2KBHalfBufferSize* **:word ) :word**

**ctwmSetDMABuffer(** *wIOHandle* **:word;** *dw32BitAddx* **:longint;**

*w2KBHalfBufferSize* **:word ) :word**

**ctwdSetDMABuffer(** *wIOHandle* **:word;** *dw32BitAddx* **:longint;**

*w2KBHalfBufferSize* **:word ) :word**

**Basic ctvmSetDMABuffer%(** *wIOHandle%***,** *dw32BitAddx&***,**

*w2KBHalfBufferSize%* **)**

**ctvdSetDMABuffer%(** *wIOHandle%***,** *dw32BitAddx&***,**

*w2KBHalfBufferSize%* **)**

**ctwmSetDMABuffer%(** *wIOHandle%***,** *dw32BitAddx&***,**

*w2KBHalfBufferSize%* **)**

**ctwdSetDMABuffer%(** *wIOHandle%***,** *dw32BitAddx&***,**

*w2KBHalfBufferSize%* **)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

dw32BitAddx

Specifies the 32-bit linear address of the DMA buffer. Currently, the buffer must be sited entirely within the lowest 1MB of memory.

w2KBHalfBufferSize

The DMA buffer size, in units of 2KB per half-buffer.

**Remarks** You must allocate the DMA buffer in order to make use of digitized sound I/O. The allocated DMA buffer must not straddle a physical page (64KB) boundary.

The valid range for *w2KBHalfBufferSize* is from 1 to 16. If an invalid value of *w2KBHalfBufferSize* is specified, an error code will be returned; and no digitized sound I/O will be carried out until a valid DMA buffer has been set up.

If the allocated buffer does not start on a paragraph (16-byte) boundary, you must allocate the buffer with an additional 15 bytes, because the driver will do a paragraph adjustment internally.

**Return Value** Zero if successful. Non-zero otherwise.

See Also Disk SetDiskBuffer functions if interest is in ct*?*dSetDMABuffer

ASM Interface ctvmSetDMABuffer: CT-VOICE.DRV Function 30

ctvdSetDMABuffer: CTVDSK.DRV Function 29

ctwmSetDMABuffer: CTWMEM.DRV Function 5

ctwdSetDMABuffer: CTWDSK.DRV Function 5

.VOC **SetIOParam** functions

**Action** Sets the value of an I/O parameter.

**Syntax C WORD ctvmSetIOParam( WORD** *wIOHandle***, WORD** *wParamType***,**

**DWORD** *dwParam***)**

**WORD ctvdSetIOParam( WORD** *wIOHandle***, WORD** *wParamType***,**

**DWORD** *dwParam***)**

**Pascal ctvmSetIOParam(** *wIOHandle***,** *wParamType* **:word;**

*dwParam***:longint ) :word**

**ctvdSetIOParam(** *wIOHandle***,** *wParamType* **:word;**

*dwParam***:longint ) :word**

**Basic ctvmSetIOParam%(** *wIOHandle%***,** *wParamType%***,** *dwParam&***)**

**ctvdSetIOParam%(** *wIOHandle%***,** *wParamType%***,** *dwParam&* **)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

wParamType

Specifies the I/O parameter to set.

dwParam

Value to assign to the I/O parameter.

**Remarks** *wParamType* must be one of the following constants:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **CTVOC\_IO\_LPSTATUSWORD** | Digitized sound status word address |
| **CTVOC\_IN\_SAMPLESPERSEC** | Sampling rate for recording |
| **CTVOC\_IN\_NCHANNELS** | Number of channels to use for recording |
| **CTVOC\_IN\_LEFTINPUTS** | Left-channel sources for recording |
| **CTVOC\_IN\_RIGHTINPUTS** | Right-channel sources for recording |
| **CTVOC\_IN\_FORMAT** | Digitized sound format for recording |
| **CTVOC\_IN\_BITSPERSAMPLE** | Bits per recorded sample |
| **CTVOC\_IN\_FILTER** | low-pass filter for recording |
| **CTVOC\_OUT\_FILTER** | low-pass filter for playback |

Further details on the parameters, and the valid *dwParam* values:

**CTVOC\_IO\_LPSTATUSWORD**

If you wish to monitor the progress of digitized sound I/O, the far address of the status word must be assigned before starting any digitized sound I/O process.

A zero status word indicates digitized sound I/O has stopped, while a non-zero status word indicates digitized sound I/O is currently active.

**CTVOC\_IN\_SAMPLESPERSEC**

Before recording, you have to specify the desired sampling rate. Otherwise the default of 11025 Hz will be used. The acceptable range is from 5000 to 44100 Hz. The maximum sampling rate is also determined by the card used.

If the sampling rate specified is not within the valid range, the nearest valid sampling rate is used.

**CTVOC\_IN\_NCHANNELS**

Before recording, you have to specify the desired recording mode by setting *dwParam* to the number of recording channels (1 for mono, 2 for stereo).

**CTVOC\_IN\_LEFTINPUTS**

**CTVOC\_IN\_RIGHTINPUTS**

If you do not want to keep the microphone input as the default recording source, you have the choice of setting up the left and right input channels separately. The *dwParam* is formed from one or more of the constants listed below:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **MIXERSWI\_MIC** | Microphone |
| **MIXERSWI\_CD\_R** | CD Audio right channel |
| **MIXERSWI\_CD\_L** | CD Audio left channel |
| **MIXERSWI\_LINE\_R** | Line-In right channel |
| **MIXERSWI\_LINE\_L** | Line-In left channel |
| **MIXERSWI\_MIDI\_R** | MIDI right channel |
| **MIXERSWI\_MIDI\_L** | MIDI left channel |

For example, to get stereo FM and CD input, set up the left input channel with (**MIXERSWI\_MIDI\_L | MIXERSWI\_CD\_L**), and set up the right input channel with (**MIXERSWI\_MIDI\_R | MIXERSWI\_CD\_R**).

Note that for mono recording, the hardware will take samples from the left input channel only. To compensate for this, you will have to turn on **MIXERSWI\_LINE\_L** and **MIXERSWI\_LINE\_R** on the left channel if you wish to have a mono LINE recording.

**CTVOC\_IN\_FORMAT**

You have to specify the digitized sound format before recording, otherwise the default format **VOC\_FORMAT\_08\_PCM** will be used. The supported format tags are as follows:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **VOC\_FORMAT\_08\_PCM** | 8-bit unsigned PCM data |
| **VOC\_FORMAT\_16\_PCM** | 16-bit signed PCM data |
| **VOC\_FORMAT\_ALAW** | CCITT A-Law data |
| **VOC\_FORMAT\_MULAW** | CCITT -Law data |
| **VOC\_FORMAT\_CREATIVE\_ADCPM** | Creative ADPCM data |

Currently, data in the latter 3 formats is compressed/decompressed in real time with the help of the Creative Advanced Signal Processor.

**CTVOC\_IN\_BITSPERSAMPLE**

See the table below for the values *dwParam* must be set to, depending on the format tag:

|  |  |
| --- | --- |
| **Format Tag** | **Bits Per Sample** |
| **VOC\_FORMAT\_08\_PCM** | 8 |
| **VOC\_FORMAT\_16\_PCM** | 16 |
| **VOC\_FORMAT\_ALAW** | 8 |
| **VOC\_FORMAT\_MULAW** | 8 |
| **VOC\_FORMAT\_CREATIVE\_ADCPM** | 4 |

**CTVOC\_IN\_FILTER**

Before recording, you can specify the low-pass filter settings. The *dwParam* is formed from one of the constants listed below:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **FILTER\_OFF** | filter off |
| **FILTER\_LOW** | low cut-off frequency |
| **FILTER\_HIGH** | high cut-off frequency |

**CTVOC\_OUT\_FILTER**

Before playback, you can specify the low-pass filter settings. The *dwParam* is formed from one of the constants listed below:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **FILTER\_OFF** | filter off |
| **FILTER\_ON** | filter on |

**Return Value** Zero if successful. Non-zero otherwise.

See Also GetIOParam functions

ASM Interface ctvmSetIOParam: CT-VOICE.DRV Function 31

ctvdSetIOParam: CTVDSK.DRV Function 30

.WAV **SetIOParam** functions

**Action** Sets the value of an I/O parameter.

**Syntax C WORD ctwmSetIOParam( WORD** *wIOHandle***, WORD** *wParamType***,**

**DWORD** *dwParam***)**

**WORD ctwdSetIOParam( WORD** *wIOHandle***, WORD** *wParamType***,**

**DWORD** *dwParam***)**

**Pascal ctwmSetIOParam(** *wIOHandle***,** *wParamType* **:word;**

*dwParam***:longint ) :word**

**ctwdSetIOParam(** *wIOHandle***,** *wParamType* **:word;**

*dwParam***:longint ) :word**

**Basic ctwmSetIOParam%(** *wIOHandle%***,** *wParamType%***,** *dwParam&***)**

**ctwdSetIOParam%(** *wIOHandle%***,** *wParamType%***,** *dwParam&* **)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

wParamType

Specifies the I/O parameter to set.

dwParam

Value to assign to the I/O parameter.

**Remarks** *wParamType* must be one of the following constants:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **CTWAV\_IO\_LPSTATUSWORD** | Digitized sound status word address |
| **CTWAV\_IN\_SAMPLESPERSEC** | Sampling rate for recording |
| **CTWAV\_IN\_NCHANNELS** | Number of channels to use for recording |
| **CTWAV\_IN\_LEFTINPUTS** | Left-channel sources for recording |
| **CTWAV\_IN\_RIGHTINPUTS** | Right-channel sources for recording |
| **CTWAV\_IN\_FORMAT** | Digitized sound format for recording |
| **CTWAV\_IN\_BITSPERSAMPLE** | Bits per recorded sample |
| **CTWAV\_IN\_FILTER** | low-pass filter for recording |
| **CTWAV\_OUT\_FILTER** | low-pass filter for playback |

Further details on the parameters, and the valid *dwParam* values:

**CTWAV\_IO\_LPSTATUSWORD**

If you wish to monitor the progress of digitized sound I/O, the far address of the status word must be assigned before starting any digitized sound I/O process.

A zero status word indicates digitized sound I/O has stopped, while a non-zero status word indicates digitized sound I/O is currently active.

**CTWAV\_IN\_SAMPLESPERSEC**

Before recording, you have to specify the desired sampling rate. Otherwise the default of 11025 Hz will be used. The acceptable sampling rates are 11025 Hz, 22050 Hz and 44100 Hz. The maximum sampling rate is also determined by the card used.

If the sampling rate specified is not valid, the nearest valid sampling rate is used.

**CTWAV\_IN\_NCHANNELS**

Before recording, you have to specify the desired recording mode by setting *dwParam* to the number of recording channels (1 for mono, 2 for stereo).

**CTWAV\_IN\_LEFTINPUTS**

**CTWAV\_IN\_RIGHTINPUTS**

If you do not want to keep the microphone input as the default recording source, you have the choice of setting up the left and right input channels separately. The *dwParam* is formed from one or more of the constants listed below:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **MIXERSWI\_MIC** | Microphone |
| **MIXERSWI\_CD\_R** | CD Audio right channel |
| **MIXERSWI\_CD\_L** | CD Audio left channel |
| **MIXERSWI\_LINE\_R** | Line-In right channel |
| **MIXERSWI\_LINE\_L** | Line-In left channel |
| **MIXERSWI\_MIDI\_R** | MIDI right channel |
| **MIXERSWI\_MIDI\_L** | MIDI left channel |

For example, to get stereo FM and CD input, set up the left input channel with (**MIXERSWI\_MIDI\_L | MIXERSWI\_CD\_L**), and set up the right input channel with (**MIXERSWI\_MIDI\_R | MIXERSWI\_CD\_R**).

Note that for mono recording, the hardware will take samples from the left input channel only. To compensate for this, you will have to turn on **MIXERSWI\_LINE\_L** and **MIXERSWI\_LINE\_R** on the left channel if you wish to have a mono LINE recording.

**CTWAV\_IN\_FORMAT**

You have to specify the digitized sound format before recording, otherwise the default format **WAVE\_FORMAT\_PCM** will be used. The supported format tags are as follows:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **WAVE\_FORMAT\_PCM** | 8-bit unsigned PCM data |
| **WAVE\_FORMAT\_ALAW** | CCITT A-Law data |
| **WAVE\_FORMAT\_MULAW** | CCITT -Law data |
| **WAVE\_FORMAT\_CREATIVE\_ADCPM** | Creative ADPCM data |

Currently, data in the latter 3 formats is compressed/decompressed in real time with the help of the Creative Advanced Signal Processor.

**CTWAV\_IN\_BITSPERSAMPLE**

See the table below for the values *dwParam* must be set to, depending on the format tag:

|  |  |
| --- | --- |
| **Format Tag** | **Bits Per Sample** |
| **WAVE\_FORMAT\_PCM** | 8 or 16 |
| **WAVE\_FORMAT\_ALAW** | 8 |
| **WAVE\_FORMAT\_MULAW** | 8 |
| **WAVE\_FORMAT\_CREATIVE\_ADCPM** | 4 |

**CTWAV\_IN\_FILTER**

Before recording, you can specify the low-pass filter settings. The *dwParam* is formed from one of the constants listed below:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **FILTER\_OFF** | filter off |
| **FILTER\_LOW** | low cut-off frequency |
| **FILTER\_HIGH** | high cut-off frequency |

**CTWAV\_OUT\_FILTER**

Before playback, you can specify the low-pass filter settings. The *dwParam* is formed from one of the constants listed below:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **FILTER\_OFF** | filter off |
| **FILTER\_ON** | filter on |

**Return Value** Zero if successful. Non-zero otherwise.

See Also GetIOParam functions

ASM Interface ctwmSetIOParam: CTWMEM.DRV Function 6

ctwdSetIOParam: CTWDSK.DRV Function 8

**SetSpeaker** functions

**Action** Turns the DAC speaker on or off.

**Syntax C void ctvmSetSpeaker( WORD** *wfOnOff***)**

**void ctvdSetSpeaker( WORD** *wfOnOff***)**

**void ctwmSetSpeaker( WORD** *wfOnOff* **)**

**void ctwdSetSpeaker( WORD** *wfOnOff* **)**

**Pascal ctvmSetSpeaker(** *wfOnOff* **:word )**

**ctvdSetSpeaker(** *wfOnOff***:word )**

**ctwmSetSpeaker(** *wfOnOff* **:word )**

**ctwdSetSpeaker(** *wfOnOff* **:word )**

**Basic ctvmSetSpeaker(** *wfOnOff%* **)**

**ctvdSetSpeaker(** *wfOnOff%***)**

**ctwmSetSpeaker(** *wfOnOff%* **)**

**ctwdSetSpeaker(** *wfOnOff%* **)**

**Parameters** *wfOnOff*

0 turns the speaker off, 1 turns it on.

**Remarks** Due to hardware differences, this function has no effect on the Sound Blaster 16. It works as intended on the 8-bit sound cards.

On the Sound Blaster 16, there is no need to turn the DAC speaker off during recording (anyway, you can't). However, you could still call this function with an argument of 0 immediately before starting any recording, so as to preserve source-level compatibility across all cards.

**Return Value** None

See Also None

ASM Interface ctvmSetSpeaker: CT-VOICE.DRV Function 4

ctvdSetSpeaker: CTVDSK.DRV Function 4

ctwmSetSpeaker: CTWMEM.DRV Function 4

ctwdSetSpeaker: CTWDSK.DRV Function 10

**Stop** functions

**Action** Stops the active digitized sound input or output process.

**Syntax C WORD ctvmStop( WORD** *wIOHandle***)**

**WORD ctvdStop( WORD** *wIOHandle***)**

**WORD ctwmStop( WORD** *wIOHandle***)**

**WORD ctwdStop( WORD** *wIOHandle***)**

**Pascal ctvmStop(** *wIOHandle***:word ) :word**

**ctvdStop(** *wIOHandle***:word ) :word**

**ctwmStop(** *wIOHandle***:word ) :word**

**ctwdStop(** *wIOHandle***:word ) :word**

**Basic ctvmStop%(** *wIOHandle%***)**

**ctvdStop%(** *wIOHandle%***)**

**ctwmStop%(** *wIOHandle%***)**

**ctwdStop%(** *wIOHandle%***)**

**Parameters** *wIOHandle*

Digitized sound I/O handle.

**Remarks** The I/O handle's digitized sound status word is reset to zero.

**Return Value** Zero if successful. Non-zero otherwise.

See Also Input functions

**Output** functions

ASM Interface ctvmStop: CT-VOICE.DRV Function 37

ctvdStop: CTVDSK.DRV Function 34

ctwmStop: CTWMEM.DRV Function 12

ctwdStop: CTWDSK.DRV Function 15

**Terminate** functions

**Action** Terminates the driver.

S**yntax C void ctvmTerminate( void )**

**void ctvdTerminate( void )**

**void ctwmTerminate( void )**

**void ctwdTerminate( void )**

**Pascal ctvmTerminate**

**ctvdTerminate**

**ctwmTerminate**

**ctwdTerminate**

**Basic ctvmTerminate( )**

**ctvdTerminate( )**

**ctwmTerminate( )**

**ctwdTerminate( )**

**Parameters** None

**Remarks** This function must be called once before the program exits.

Any currently active digitized sound process is stopped.

**ct*?*dTerminate** will also restore those interrupts intercepted during a successful call to **ct*?*dInit**.

**Return Value** None

See Also Init functions

ASM Interface ctvmTerminate: CT-VOICE.DRV Function 9

ctvdTerminate: CTVDSK.DRV Function 9

ctwmTerminate: CTWMEM.DRV Function 3

ctwdTerminate: **CTWDSK.DRV** Function 3

CT-VOICE.DRV Assembly Interface

The entries in this section tell you what values to load into the respective registers in order to invoke the driver services. The details are in the High-Level Language interface descriptions.

3 Initialize Driver

**Action** Initializes the driver and the sound card.

Entry BX = 3

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmInit**

4 Set Speaker

**Action** Turns the DAC speaker on or off.

Entry BX = 4

**AX** = *wfOnOff*

**Exit** None

**Details In** CT-VOICE.DRV HLL Interface **ctvmSetSpeaker**

9 Terminate Driver

**Action** Terminates the driver.

Entry BX = 9

**Exit** None

**Details In** CT-VOICE.DRV HLL Interface **ctvmTerminate**

28 Get Environment Settings

**Action** Passes the BLASTER environment string to the driver for it to interpret as to which hardware settings are to be used.

Entry BX = 28

**ES:SI** = *lpszBlaster*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmGetEnvSettings**

29 Get Parameter

**Action** Gets information on the driver and sound card.

Entry BX = 29

**AX** = *wParamType*

**ES:DI** = *lpdwParam*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmGetParam**

30 Set DMA Buffer

**Action** Sets up the DMA buffer for an I/O handle.

Entry BX = 30

**AX** = *wIOHandle*

**ES:DI** = *dw32BitAddx*

**CX** = *w2KBHalfBufferSize*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmSetDMABuffer**

31 Set I/O Parameter

**Action** Sets the value of an I/O parameter.

Entry BX = 31

**AX** = *wIOHandle*

**DX** = *wParamType*

**DI:SI** = *dwParam*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmSetIOParam**

32 Get I/O Parameter

**Action** Gets the value of an I/O parameter.

Entry BX = 32

**AX** = *wIOHandle*

**DX** = *wParamType*

**ES:DI** = *lpdwParam*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmGetIOParam**

33 Input into Conventional Memory

**Action** Records digitized sound input to conventional memory.

Entry BX = 33

**AX** = *wIOHandle*

**ES:DI =** *lpBuf*

**DX:CX** = *dwBufLen*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmInputCM**

34 Input into Extended Memory

**Action** Records digitized sound input to extended memory.

Entry BX = 34

**AX** = *wIOHandle*

**DX** = *wXMBHandle*

**DI:SI =** *dwXMBOffset*

**CX** = *wKBBufferSize*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmInputXM**

35 Output from Conventional Memory

**Action** Starts digitized sound output from conventional memory.

Entry BX = 35

**AX** = *wIOHandle*

**ES:DI =** *lpBuf*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmOutputCM**

36 Output from Extended Memory

**Action** Starts digitized sound output from extended memory.

Entry BX = 36

**AX** = *wIOHandle*

**DX** = wXMBHandle

**DI:SI =** *dwXMBOffset*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmOutputXM**

37 Stop Digitized sound I/O

**Action** Stops the active digitized sound input or output process.

Entry BX = 37

**AX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmStop**

38 Pause Digitized sound Output

**Action** Pauses the active digitized sound output process.

Entry BX = 38

**AX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmPause**

39 Continue Digitized sound Output

**Action** Continues the paused digitized sound output process.

Entry BX = 39

**AX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmContinue**

40 Break Digitized sound Output Loop

**Action** Breaks out from a repeat loop in the current digitized sound output process.

Entry BX = 40

**AX** = *wIOHandle*

**CX** = *wBreakMode*

**Exit** **AX** = error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmBreakLoop**

41 Get Error Codes

**Action** Gets the most recent operation's driver and DOS error codes.

Entry BX = 41

**Exit** **AX** = driver error code

**DX** = DOS error code

**Details In** CT-VOICE.DRV HLL Interface **ctvmGetDrvError** and **ctvmGetExtError**

CTVDSK.DRV Assembly Interface

The entries in this section tell you what values to load into the respective registers in order to invoke the driver services. The details are in the High-Level Language interface descriptions.

4 Set Speaker

**Action** Turns the DAC speaker on or off.

Entry BX = 4

**AX** = *wfOnOff*

**Exit** None

**Details In** CTVDSK.DRV HLL Interface **ctvdSetSpeaker**

9 Terminate Driver

**Action** Terminates the driver.

Entry BX = 9

**Exit** None

**Details In** CTVDSK.DRV HLL Interface **ctvdTerminate**

14 Get Error Codes

**Action** Gets the most recent operation's driver and DOS error codes.

Entry BX = 14

**Exit** **AX** = driver error code

**DX** = DOS error code

**Details In** CTVDSK.DRV HLL Interface **ctvdGetDrvError** and **ctvdGetExtError**

26 Get Environment Settings

**Action** Passes the BLASTER environment string to the driver for it to interpret as to which hardware settings to use.

Entry BX = 26

**DX:AX** = *lpszBlaster*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdGetEnvSettings**

27 Get Parameter

**Action** Gets information on the driver and sound card.

Entry BX = 27

**CX** = *wParamType*

**DX:AX** = *lpdwParam*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdGetParam**

28 Set Disk Buffer

**Action** Sets up the disk buffer for an I/O handle.

Entry BX = 28

**SI** = *wIOHandle*

**DX:AX** = *lpBuffer*

**CX** = *w2KBHalfBufferSize*

**Exit AX = error code**

**Details In** CTVDSK.DRV HLL Interface **ctvdSetDiskBuffer**

29 Set DMA Buffer

**Action** Sets up the DMA buffer for an I/O handle.

Entry BX = 29

**SI** = *wIOHandle*

**DX:AX** = *dw32BitAddx*

**CX** = *w2KBHalfBufferSize*

**Exit AX = error code**

**Details In** CTVDSK.DRV HLL Interface **ctvdSetDMABuffer**

30 Set I/O Parameter

**Action** Sets the value of an I/O parameter.

Entry BX = 30

**SI** = *wIOHandle*

**CX** = *wParamType*

**DX:AX** = *dwParam*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdSetIOParam**

31 Get I/O Parameter

**Action** Gets the value of an I/O parameter.

Entry BX = 31

**SI** = *wIOHandle*

**CX** = *wParamType*

**DX:AX** = *lpdwParam*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdGetIOParam**

32 Input

**Action** Starts digitized sound input into a disk file.

Entry BX = 32

**DX** = *wIOHandle*

**AX** = *wFileHandle*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdInput**

33 Output

**Action** Starts digitized sound output from a disk file.

Entry BX = 33

**DX** = *wIOHandle*

**AX** = *wFileHandle*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdOutput**

34 Stop Digitized sound I/O

**Action** Stops the active digitized sound input or output process.

Entry BX = 34

**DX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdStop**

35 Pause Digitized sound Output

**Action** Pauses the active digitized sound output process.

Entry BX = 35

**DX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdPause**

36 Continue Digitized sound Output

**Action** Continues the paused digitized sound output process.

Entry BX = 36

**DX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdContinue**

37 Break Digitized sound Output Loop

**Action** Breaks out from a repeat loop in the current digitized sound output process.

Entry BX = 37

**DX** = *wIOHandle*

**AX** = *wBreakMode*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdBreakLoop**

38 Initialize Driver

**Action** Initializes the driver and the sound card.

Entry BX = 38

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdInit**

39 Output from an Offset

**Action** Starts digitized sound from an offset of a disk file.

Entry BX = 39

**SI** = *wIOHandle*

**DI** = *wFileHandle*

**DX:CX** = *dwOffset*

**DX** = *wSeekMode*

**Exit** **AX** = error code

**Details In** CTVDSK.DRV HLL Interface **ctvdOutputOffset**

CTWMEM.DRV Assembly Interface

The entries in this section tell you what values to load into the respective registers in order to invoke the driver services. The details are in the High-Level Language interface descriptions.

0 Get Environment Settings

**Action** Passes the BLASTER environment string to the driver for it to interpret as to which hardware settings to use.

Entry BX = 0

**ES:SI** = *lpszBlaster*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmGetEnvSettings**

1 Get Parameter

**Action** Gets information on the driver and sound card.

Entry BX = 1

**AX** = *wParamType*

**ES:DI** = *lpdwParam*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmGetParam**

2 Initialize Driver

**Action** Initializes the driver and the sound card.

Entry BX = 2

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmInit**

3 Terminate Driver

**Action** Terminates the driver.

Entry BX = 3

**Exit** None

**Details In** CTWMEM.DRV HLL Interface **ctwmTerminate**

4 Set Speaker

**Action** Turns the DAC speaker on or off.

Entry BX = 4

**AX** = *wfOnOff*

**Exit** None

**Details In** CTWMEM.DRV HLL Interface **ctwmSetSpeaker**

5 Set DMA Buffer

**Action** Sets up the DMA buffer for an I/O handle.

Entry BX = 5

**AX** = *wIOHandle*

**ES:DI** = dw32BitAddx

**CX** = *w2KBHalfBufferSize*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmSetDMABuffer**

6 Set I/O Parameter

**Action** Sets the value of an I/O parameter.

Entry BX = 6

**AX** = *wIOHandle*

**DX** = *wParamType*

**DI:SI** = *dwParam*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmSetIOParam**

7 Get I/O Parameter

**Action** Gets the value of an I/O parameter.

Entry BX = 7

**AX** = *wIOHandle*

**DX** = *wParamType*

**ES:DI** = *lpdwParam*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmGetIOParam**

8 Input into Conventional Memory

**Action** Records digitized sound input in conventional memory.

Entry BX = 8

**AX** = *wIOHandle*

**ES:DI =** *lpBuf*

**DX:CX** = *dwBufLen*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmInputCM**

9 Input into Extended Memory

**Action** Records digitized sound input to extended memory.

Entry BX = 9

**AX** = *wIOHandle*

**DX** = *wXMBHandle*

**DI:SI =** dwXMBOffset

**CX** = *wKBBufferSize*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmInputXM**

10 Output from Conventional Memory

**Action** Plays digitized sound output from conventional memory.

Entry BX = 10

**AX** = *wIOHandle*

**ES:DI =** *lpBuf*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmOutputCM**

11 Output from Extended Memory

**Action** Plays digitized sound output from extended memory.

Entry BX = 11

**AX** = *wIOHandle*

**DX** = *wXMBHandle*

**DI:SI =** *dwXMBOffset*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmOutputXM**

12 Stop Digitized sound I/O

**Action** Stops the active digitized sound input or output process.

Entry BX = 12

**AX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmStop**

13 Pause Digitized sound Output

**Action** Pauses the active digitized sound output process.

Entry BX = 13

**AX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmPause**

14 Continue Digitized sound Output

**Action** Continues the paused digitized sound output process.

Entry BX = 14

**AX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTWMEM.DRV HLL Interface **ctwmContinue**

15 Get Error Codes

**Action** Gets the most recent operation's driver and DOS error codes.

Entry BX = 15

**Exit** **AX** = driver error code

**DX** = DOS error code

**Details In** CTWMEM.DRV HLL Interface **ctwmGetDrvError** and **ctwmGetExtError**

CTWDSK.DRV Assembly Interface

The entries in this section tell you what values to load into the respective registers in order to invoke the driver services. The details are in the High-Level Language interface descriptions.

0 Get Environment Settings

**Action** Passes the BLASTER environment string to the driver for it to interpret as to which hardware settings to use.

Entry BX = 0

**DX:AX** = *lpszBlaster*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdGetEnvSettings**

1 Get Parameter

**Action** Gets information on the driver and sound card.

Entry BX = 1

**CX** = *wParamType*

**DX:AX** = *lpdwParam*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdGetParam**

2 Set Disk Buffer

**Action** Sets up the disk buffer for an I/O handle.

Entry BX = 2

**SI** = *wIOHandle*

**DX:AX** = *lpBuffer*

**CX** = *w2KBHalfBufferSize*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdSetDiskBuffer**

3 Initialize Driver

**Action** Initializes the driver and the sound card.

Entry BX = 3

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdInit**

4 Terminate Driver

**Action** Terminates the driver.

Entry BX = 4

**Exit** None

**Details In** CTWDSK.DRV HLL Interface **ctwdTerminate**

5 Set DMA Buffer

**Action** Sets up the DMA buffer for an I/O handle.

Entry BX = 5

**SI** = *wIOHandle*

**DX:AX** = *dw32BitAddx*

**CX** = *w2KBHalfBufferSize*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdSetDMABuffer**

6 Get Error Codes

**Action** Gets the most recent operation's driver and DOS error codes.

Entry BX = 6

**Exit** **AX** = driver error code

**DX** = DOS error code

**Details In** CTWDSK.DRV HLL Interface **ctwdGetDrvError** and **ctwdGetExtError**

8 Set I/O Parameter

**Action** Sets the value of an I/O parameter.

Entry BX = 8

**SI** = *wIOHandle*

**CX** = *wParamType*

**DX:AX** = *dwParam*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdSetIOParam**

9 Get I/O Parameter

**Action** Gets the value of an I/O parameter.

Entry BX = 9

**SI** = *wIOHandle*

**CX** = *wParamType*

**DX:AX** = *lpdwParam*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdGetIOParam**

10 Set Speaker

**Action** Turns the DAC speaker on or off.

Entry BX = 10

**AX** = *wfOnOff*

**Exit** None

**Details In** CTWDSK.DRV HLL Interface **ctwdSetSpeaker**

11 Output

**Action** Starts digitized sound output from a disk file.

Entry BX = 11

**DX** = *wIOHandle*

**AX** = *wFileHandle*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdOutput**

12 Input

**Action** Starts digitized sound input into a disk file.

Entry BX = 12

**DX** = *wIOHandle*

**AX** = *wFileHandle*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdInput**

13 Pause Digitized sound Output

**Action** Pauses the active digitized sound output process.

Entry BX = 13

**DX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdPause**

14 Continue Digitized sound Output

**Action** Continues the paused digitized sound output process.

Entry BX = 14

**DX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdContinue**

15 Stop Digitized sound I/O

**Action** Stops the active digitized sound input or output process.

Entry BX = 15

**DX** = *wIOHandle*

**Exit** **AX** = error code

**Details In** CTWDSK.DRV HLL Interface **ctwdStop**

Chapter 2

High-Level Auxiliary Driver

This chapter documents the interfaces to the Creative high-level loadable auxiliary driver (AUXDRV.DRV).

It is divided into two parts. The first part covers high-level language interfaces and the second covers the register base interfaces. Cross-references to the register base interface are provided in the high-level language interfaces.

The interfaces require a minimum driver version number of 4.00.

High-Level Language Interface

This section describes the high-level language interface to the AUXDRV driver in alphabetical order.

Function Prefix

High-level function names begin with the following prefix:

|  |  |
| --- | --- |
| **Prefix** | **Driver** |
| ctad | AUXDRV |

Include Files

The followings are the required include files for the AUXDRV driver:

|  |  |  |  |
| --- | --- | --- | --- |
| **Driver** | **C Language** | **Turbo Pascal** | **Microsoft Basic** |
| AUXDRV | SBKAUX.H | SBKAUX.INC | SBKAUX.BI |

Functions by Category

The AUXDRV driver functions may be divided into the following categories:

|  |  |
| --- | --- |
| **Category** | **Functions** |
|  |  |
| Initialization/Termination | **ctadGetEnvSettings** |
|  | **ctadInit** |
|  | **ctadTerminate** |
|  |  |
|  |  |
| Volume control | **ctadGetVolume** |
|  | **ctadSetVolume** |
|  |  |
|  |  |
| Fade or Pan effects | **ctadFade** |
|  | **ctadPan** |
|  |  |
|  |  |
|  | **ctadClrSource** |
|  | **ctadPauseCtrl** |
|  | **ctadStartCtrl** |
|  | **ctadStopCtrl** |
|  |  |
|  |  |
|  | **ctadGetPanPosition** |
|  |  |
|  |  |
|  | **ctadSetFadeStAddx** |
|  | **ctadSetPanStAddx** |
|  |  |
|  |  |
| Input and Output Mixing | **ctadGetMixerSwitch** |
|  | **ctadSetMixerSwitch** |
|  |  |
|  |  |
| Gain control | **ctadGetMixerGain** |
|  | **ctadSetMixerGain** |
|  |  |
|  |  |
| Tone control | **ctadGetToneLevel** |
|  | **ctadSetToneLevel** |
|  |  |
|  |  |
| Microphone AGC | **ctadGetAGC** |
|  | **ctadSetAGC** |
|  |  |
|  |  |
| Miscellaneous | **ctadGetDrvVer** |
|  | **ctadResetMixer** |
|  |  |
|  |  |

**ctadClrSource**

**Action** Clears a stereo volume source already set up for Fade or Pan effect, or which is currently active in either of these processes.

**Syntax C WORD ctadClrSource( WORD** *wVolSource***, WORD** *wEffect* **)**

**Pascal ctadClrSource(** *wVolSource***,** *wEffect* **:word ) :word**

**Basic ctadClrSource%(** *wVolSource%***,** *wEffect%* **)**

**Parameters** *wVolSource*

Stereo volume source. See **ctadSetVolume** for the manifest constants.

wEffect

0 for Fade; 1 for Pan.

**Remarks** **ctadStopCtrl** provides the shotgun approach, terminating both the fade and pan effects on all volume sources. This function provides a sniper rifle approach, allowing you to kill off an effect of a volume source.

**Return Value** Zero if successful. Non-zero if the specified source is not active.

See Also ctadPauseCtrl, ctadStartCtrl, ctadStopCtrl

ASM Interface AUXDRV.DRV Function 13

**ctadFade**

**Action** Sets up Fade parameters for the specified stereo volume source.

**Syntax C WORD ctadFade( WORD** *wVolSource***, WORD** *wFinalVol***,**

**WORD** *wCycleTime***, WORD** *wFadeMode***,**

**WORD** *wRepeatCount* **)**

**Pascal ctadFade(** *wVolSource***,** *wFinalVol***,** *wCycleTime***,** *wFadeMode***,**

*wRepeatCount* **:word ) :word**

**Basic ctadFade%(** *wVolSource%***,** *wFinalVol%***,** *wCycleTime%***,**

*wFadeMode%***,** *wRepeatCount%* **)**

**Parameters** *wVolSource*

Stereo volume source. See **ctadSetVolume** for the manifest constants.

wFinalVol

High and low bytes denote the left and right final volume levels respectively. The acceptable range for each is from 0 to 255.

wCycleTime

Time taken, units of milliseconds, to complete one fade cycle. The acceptable range is from 1 to 65535.

wFadeMode

0 to stop the fade process at the final volume level specified by *wFinalVol*.

1 to make a loop back to the original level before stopping.

wRepeatCount

Number of cycles to repeat. The acceptable range is from 1 to 65535.

**Remarks** The fading process will only actually be carried out when **ctadStartCtrl** is successfully invoked.

**Return Value** Zero if successful. Non-zero if the specified source has been set up for pan effect, or if it is currently active in a fade or pan process.

See Also ctadStartCtrl, ctadSetFadeStAddx

ASM Interface AUXDRV.DRV Function 8

**ctadGetAGC**

**Action** Reads the Automatic Gain Control status.

**Syntax C WORD ctadGetAGC( void )**

**Pascal ctadGetAGC :word**

**Basic ctadGetAGC%( )**

**Parameters** None

**Remarks** None

**Return Value** 0 indicates the AGC is off; 1 that it is on.

See Also ctadSetAGC

ASM Interface AUXDRV.DRV Function 26

**ctadGetDrvVer**

**Action** Gets the version number of the auxiliary driver.

**Syntax C WORD ctadGetDrvVer( void )**

**Pascal ctadGetDrvVer :word**

**Basic ctadGetDrvVer%( )**

**Parameters** None

**Remarks** None

**Return Value** Major version number in the high byte. Minor version number in the low byte.

See Also None

ASM Interface AUXDRV.DRV Function 0

**ctadGetEnvSettings**

**Action** Passes the BLASTER environment string to the driver for it to interpret the hardware settings to use.

**Syntax C WORD ctadGetEnvSettings( const char far \****lpszBlaster***)**

**Pascal ctadGetEnvSettings(** *lpszBlaster***:pointer) :word**

**Basic ctadGetEnvSettings%(** *lpszBlaster&***)**

**Parameters** *lpszBlaster*

Far pointer to the BLASTER environment string, without the "BLASTER=" prefix.

**Remarks** This function must be called first, even before **ctadInit**.

**Return Value** Zero if successful. Otherwise, a bit-or'ed combination of the values listed in the table below is returned:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 0x0001 | *lpszBlaster* is NULL, or points to an empty string |
| 0x0002 | Base I/O address not specified, or is out of range |

See Also ctadInit

ASM Interface AUXDRV.DRV Function 29

**ctadGetMixerGain**

**Action** Reads the gain of either the input or output mixer.

**Syntax C WORD ctadGetMixerGain( WORD** *wMixer***)**

**Pascal ctadGetMixerGain(** *wMixer***:word) :word**

**Basic ctadGetMixerGain%(** *wMixer%***)**

**Parameters** *wMixer*

0 for the input mixer; 1 for the output mixer.

**Remarks** None

**Return Value** High and low bytes hold the left and right channel gains respectively. The expected range for each is from 0 to 3.

See Also ctadSetMixerGain

ASM Interface AUXDRV.DRV Function 22

**ctadGetMixerSwitch**

**Action** Reads the settings of the input or output mixer switches.

**Syntax C DWORD ctadGetMixerSwitch( WORD** *wMixer***)**

**Pascal ctadGetMixerSwitch(** *wMixer***:word) :longint**

**Basic ctadGetMixerSwitch&(** *wMixer%***)**

**Parameters** *wMixer*

0 for the input mixer; 1 for the output mixer.

**Remarks** None

**Return Value** High and low words hold the left and right channel switches respectively. Refer to the table in the entry for **ctadSetMixerSwitch** for the corresponding switch settings.

See Also ctadSetMixerSwitch

ASM Interface AUXDRV.DRV Function 28

**ctadGetPanPosition**

**Action** Gets the Pan position of the specified stereo volume source.

**Syntax C WORD ctadGetPanPosition( WORD** *wVolSource***)**

**Pascal ctadGetPanPosition(** *wVolSource***:word ) :word**

**Basic ctadGetPanPosition%(** *wVolSource%***)**

**Parameters** *wVolSource*

Stereo volume source. See **ctadSetVolume** for the manifest constants.

**Remarks** None

**Return Value** If the function is successful, the pan position is returned. Otherwise, 0xFFFF is returned if the specified source has not gone through any panning process.

See Also None

ASM Interface AUXDRV.DRV Function 15

**ctadGetToneLevel**

**Action** Reads the current Treble or Bass tone setting.

**Syntax C WORD ctadGetToneLevel( WORD *wTone* )**

**Pascal ctadGetToneLevel(** *wTone* **:word ) :word**

**Basic ctadGetToneLevel%(** *wTone%* **)**

**Parameters** *wTone*

0 for treble; 1 for bass.

**Remarks** None

**Return Value** High and low bytes hold the left and right channel tone levels respectively. The expected range for each is from 0 to 255.

See Also ctadSetToneLevel

ASM Interface AUXDRV.DRV Function 24

**ctadGetVolume**

**Action** Reads the volume level of the specified source.

**Syntax C WORD ctadGetVolume( WORD** *wVolSource***)**

**Pascal ctadGetVolume(** *wVolSource***:word ) :word**

**Basic ctadGetVolume%(** *wVolSource%* **)**

**Parameters** *wVolSource*

Volume source as for **ctadSetVolume**.

**Remarks** None

**Return Value** For mono sources (microphone or PC speaker), the entire word holds the volume level. For stereo sources, the high byte and low bytes hold the left and right volume levels respectively. Each volume level is from 0 to 255.

See Also ctadSetVolume

ASM Interface AUXDRV.DRV Function 7

**ctadInit**

**Action** Initializes the driver.

**Syntax C void ctadInit( void )**

**Pascal ctadInit**

**Basic ctadInit( )**

**Parameters** None

**Remarks** **ctadGetEnvSettings** must be called before this function, which in turn must be called before all the other auxiliary driver functions can be called.

**Return Value** None

See Also ctadGetEnvSettings, ctadTerminate

ASM Interface AUXDRV.DRV Function 4

**ctadPan**

**Action** Sets up Pan parameters for the specified stereo volume source.

**Syntax C WORD ctadPan( WORD** *wVolSource***, WORD** *wInitialPos***,**

**WORD** *wFinalPos***, WORD** *wCycleTime***,**

**WORD** *wPanMode***, WORD** *wRepeatCount* **)**

**Pascal ctadPan(** *wVolSource***,** *wInitialPos***,** *wFinalPos***,** *wCycleTime***,**

*wPanMode***,** *wRepeatCount* **:word ) :word**

**Basic ctadPan%(** *wVolSource%***,** *wInitialPos%***,** *wFinalPos%***,**

*wCycleTime%***,** *wPanMode%***,** *wRepeatCount%* **)**

**Parameters** *wVolSource*

Stereo volume source. See **ctadSetVolume** for the manifest constants.

wInitialPos

Any position between the extreme left (position 0) and extreme right (position 255) at which to start the panning.

wFinalPos

Any position between the extreme left (position 0) and extreme right (position 255) at which to end the panning.

wCycleTime

Specifies the time, in milliseconds, taken to complete one pan cycle. The acceptable range is from 1 to 65535.

wPanMode

0 to stop panning process at the final position.

1 to make a loop back to the initial position before stopping.

wRepeatCount

Number of repeat cycles. The acceptable range is from 1 to 65535.

**Remarks** The panning process will only actually be carried out when **ctadStartCtrl** is successfully invoked.

**Return Value** Zero if successful. Non-zero if the specified source has already been set up for fade effect, or if it is currently active in a fading or panning process.

See Also ctadSetPanStAddx, ctadStartCtrl

ASM Interface AUXDRV.DRV Function 9

**ctadPauseCtrl**

**Action** Pauses the active Fade and Pan processes.

**Syntax C WORD ctadPauseCtrl( void )**

**Pascal ctadPauseCtrl : word**

**Basic ctadPauseCtrl%( )**

**Parameters** None

**Remarks** To resume the fading or panning process, invoke **ctadStartCtrl** again.

The fade and pan status words remain unchanged.

**Return Value** Zero if successful. Non-zero if there is no active fading or panning process.

See Also ctadClrSource, ctadStartCtrl, ctadStopCtrl

ASM Interface AUXDRV.DRV Function 12

**ctadResetMixer**

**Action** Resets the mixer chip to its default state.

**Syntax C void ctadResetMixer( void )**

**Pascal ctadResetMixer**

**Basic ctadResetMixer%( )**

**Parameters** None

**Remarks** None

**Return Value** None

See Also None

ASM Interface AUXDRV.DRV Function 17

**ctadSetAGC**

**Action** Turns on or off the AGC of the microphone input.

**Syntax C WORD ctadSetAGC( WORD** *wfOnOff* **)**

**Pascal ctadSetAGC(** *wfOnOff* **:word) :word**

**Basic ctadSetAGC%(** *wfOnOff%* **)**

**Parameters** *wfOnOff*

0 to turn off, 1 to turn on.

**Remarks** None.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctadGetAGC

ASM Interface AUXDRV.DRV Function 25

**ctadSetFadeStAddx**

**Action** Sets the address of the Fade status word.

**Syntax C void ctadSetFadeStAddx( WORD far \****lpwFadeStatus* **)**

**Pascal ctadSetFadeStAddx( var** *lpwFadeStatus* **:word )**

**Basic ctadSetFadeStAddx( SEG** *lpwFadeStatus%* **)**

**Parameters** *lpwFadeStatus*

Far pointer to the Fade status word.

**Remarks** This function must be called to provide the driver with the address of the fade status word. This status word is used to monitor the fade process.

Refer to the Programmer's Guide for the format of the status word.

When this function is called, the status word content is reset to zero.

When the fade process is activated with **ctadStartCtrl**, only those sources that have been properly set up with **ctadFade** will undergo fading and have their corresponding status bits set to 1. The bits remain at 1 until the end of the fading process, or until the sources are cleared.

When the fading process is paused, the status word remains unchanged.

**Return Value** None.

See Also ctadFade

**ctadClrSource, ctadPauseCtrl, ctadStartCtrl, ctadStopCtrl**

**ctadSetPanStAddx**

ASM Interface AUXDRV.DRV Function 2

**ctadSetMixerGain**

**Action** Sets the gain of the input or output mixer.

**Syntax C WORD ctadSetMixerGain( WORD***wMixer***, WORD** *wGain***)**

**Pascal ctadSetMixerGain(** *wMixer***,** *wGain***:word ) :word**

**Basic ctadSetMixerGain%(** *wMixer%***,** *wGain%***)**

**Parameters** *wMixer*

0 for input mixer; 1 for output mixer.

wGain

High and low bytes specify the left and right channel gains. The acceptable range for each is from 0 to 3.

**Remarks** None

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctadGetMixerGain

ASM Interface AUXDRV.DRV Function 21

**ctadSetMixerSwitch**

**Action** Sets the input or output mixer switches.

**Syntax C WORD ctadSetMixerSwitch( WORD** *wMixer***,DWORD** *dwSwitches* **)**

**Pascal ctadSetMixerSwitch(** *wMixer* **:word,** *dwSwitches* **:longint ) :word**

**Basic ctadSetMixerSwitch%(** *wMixer%***,***dwSwitches&* **)**

**Parameters** *wMixer*

0 for the input mixer; 1 for the output mixer.

dwSwitches

High and low words specify the left and right channel switches respectively.

**Remarks** *dwSwitches* can be any bit-or'ed combination of the following constants:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **MIXERSWI\_MIC** | Microphone |
| **MIXERSWI\_CD\_R** | CD Audio right channel |
| **MIXERSWI\_CD\_L** | CD Audio left channel |
| **MIXERSWI\_LINE\_R** | Line-In right channel |
| **MIXERSWI\_LINE\_L** | Line-In left channel |
| **MIXERSWI\_MIDI\_R** | MIDI right channel |
| **MIXERSWI\_MIDI\_L** | MIDI left channel |

For the output mixer, no separate left and right channel control is available. In this case, only the low word of *dwSwitches* is used and it is limited to a combination of **MIXERSWI\_MIC**, **MIXERSWI\_CD\_L**, **MIXERSWI\_CD\_R**, **MIXERSWI\_LINE\_R** and **MIXERSWI\_LINE\_L**.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctadGetMixerSwitch

ASM Interface AUXDRV.DRV Function 27

**ctadSetPanStAddx**

**Action** Sets the address of the Pan status word.

**Syntax C void ctadSetPanStAddx( WORD far \****lpwPanStatus* **)**

**Pascal ctadSetPanStAddx( var** *lpwPanStatus* **:word)**

**Basic ctadSetPanStAddx( SEG** *lpwPanStatus%* **)**

**Parameters** *lpwPanStatus*

Far pointer to the Pan status word.

**Remarks** Details can be found under **ctadSetFadeStAddx**. Just substitute ‘pan’ for ‘fade’ in the descriptions.

**Return Value** None.

See Also ctadPan

**ctadClrSource, ctadPauseCtrl, ctadStartCtrl, ctadStopCtrl**

**ctadSetFadeStAddx**

ASM Interface AUXDRV.DRV Function 3

**ctadSetToneLevel**

**Action** Sets the Treble or Bass tone level.

**Syntax** **C WORD ctadSetToneLevel( WORD** *wTone***, WORD** *wLevel* **)**

**Pascal ctadSetToneLevel(** *wTone***,** *wLevel* **:word ) :word**

**Basic ctadSetToneLevel%(** *wTone%***,** *wLevel%* **)**

**Parameters** *wTone*

0 for treble; 1 for bass.

wLevel

High and low bytes specify the left and right channel tone levels respectively. The acceptable range for each is from 0 to 255.

**Remarks** None

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctadGetToneLevel

ASM Interface AUXDRV.DRV Function 23

**ctadSetVolume**

**Action** Sets the volume level of the specified source.

**Syntax C WORD ctadSetVolume( WORD** *wVolSource***, WORD** *wLevel***)**

**Pascal ctadSetVolume(** *wVolSource***,** *wLevel***:word ) :word**

**Basic ctadSetVolume%(** *wVolSource%***,** *wLevel%***)**

**Parameters** *wVolSource*

Volume source.

wLevel

High and low bytes specify the left and right volume levels to be set respectively.

**Remarks** *wVolSource* must be one of the following constants:

|  |  |  |
| --- | --- | --- |
| **Constant** | **Meaning** | **Mono/Stereo** |
| **MIXERVOL\_MASTER** | Overall Master | Stereo |
| **MIXERVOL\_VOICE** | Digitized sound | Stereo |
| **MIXERVOL\_MIDI** | MIDI | Stereo |
| **MIXERVOL\_CD** | CD Audio | Stereo |
| **MIXERVOL\_LINE** | Line-In | Stereo |
| **MIXERVOL\_MIC** | Microphone | Mono |
| **MIXERVOL\_PCSPEAKER** | PC Speaker | Mono |

Stereo sources have separate left and right channel volume control. For mono sources, the entire word holds the volume level. Both stereo and mono volume sources support volume control levels from 0 to 255. For mono sources, only low byte will be used.

**Return Value** Zero if successful. Non-zero if the specified source has been set up for fade or pan effect, or if it is currently active in a fading or panning process.

See Also ctadGetVolume

ASM Interface AUXDRV.DRV Function 6

**ctadStartCtrl**

**Action** Starts the Fading and Panning processes (which are already set up).

**Syntax C WORD ctadStartCtrl( void )**

**Pascal ctadStartCtrl :word**

**Basic ctadStartCtrl%( )**

**Parameters** None

**Remarks** The volume sources activated for the fade and pan effects will be indicated in the fade and pan status words.

**Return Value** Zero if successful. Non-zero if no source has been set up for a fade or pan effect.

See Also ctadFade, ctadPan

**ctadClrSource, ctadPauseCtrl, ctadStopCtrl**

ASM Interface AUXDRV.DRV Function 10

**ctadStopCtrl**

**Action** Terminates the active Fading and Panning processes.

**Syntax C WORD ctadStopCtrl( void )**

**Pascal ctadStopCtrl :word**

**Basic ctadStopCtrl%( )**

**Parameters** None

**Remarks** The fade and pan status words are reset to zero.

**Return Value** Zero if successful. Non-zero if no fade or pan process is active.

See Also ctadPauseCtrl, ctadStartCtrl

ASM Interface AUXDRV.DRV Function 11

**ctadTerminate**

**Action** Terminates the driver.

**Syntax C void ctadTerminate( void )**

**Pascal ctadTerminate**

**Basic ctadTerminate( )**

**Parameters** None

**Remarks** This function must be called once before the program exits.

**Return Value** None

See Also ctadInit

ASM Interface AUXDRV.DRV Function 5

Assembly Interface

The entries in this section tell you what values to load into the respective registers in order to invoke the driver services. The details are in the High-Level Language interface descriptions.

0 Get Version Number

**Action** Gets the version number of the auxiliary driver.

Entry BX = 0

**Exit AX** = version number

**Details In** AUXDRV.DRV HLL Interface **ctadGetDrvVer**

2 Set Address of Fade Status Word

**Action** Sets the address of the Fade status word.

Entry BX = 2

**ES:DI** = *lpwFadeStatus*

**Exit** None

**Details In** AUXDRV.DRV HLL Interface **ctadSetFadeStAddx**

3 Set Address of Pan Status Word

**Action** Sets the address of the Pan status word.

Entry BX = 3

**ES:DI** = *lpwPanStatus*

**Exit** None

**Details In** AUXDRV.DRV HLL Interface **ctadSetPanStAddx**

4 Initialize Driver

**Action** Initializes the driver.

Entry BX = 4

**Exit** None

**Details In** AUXDRV.DRV HLL Interface **ctadInit**

5 Terminate Driver

**Action** Terminates the driver.

Entry BX = 5

**Exit** None

**Details In** AUXDRV.DRV HLL Interface **ctadTerminate**

6 Set Volume

**Action** Sets the volume level of the specified source.

Entry BX = 6

**AX** = *wVolSource*

**DX** = *wLevel*

**Exit** **AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadSetVolume**

7 Get Volume

**Action** Reads the volume level of the specified source.

Entry BX = 7

**AX** = *wVolSource*

**Exit** **AX** =*wLevel*

**Details In** AUXDRV.DRV HLL Interface **ctadGetVolume**

8 Set up Fade

**Action** Sets up Fade parameters for the specified stereo volume source.

Entry BX = 8

**AL** = *VolSource*

**AH** = *FadeMode*

**CX** = *wCycleTime*

**DX** = *wRepeatCount*

**DI** = *wFinalLevel*

**Exit** **AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadFade**

9 Set up Pan

**Action** Sets up Pan parameters for the specified stereo volume source.

Entry BX = 9

**AL** = *VolSource*

**AH** = *PanMode*

**CX** = *wCycleTime*

**DX** = *wRepeatCount*

**SI** = *wInitialPos*

**DI** = *wFinalPos*

**Exit** **AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadPan**

10 Start Fade and Pan

**Action** Starts the Fading and Panning processes (which are already set up).

Entry BX = 10

**Exit** **AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadStartCtrl**

11 Stop Fade and Pan

**Action** Terminates the active Fading and Panning processes.

Entry BX = 11

**Exit** **AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadStopCtrl**

12 Pause Fade and Pan

**Action** Pauses the active Fading and Panning processes.

Entry BX = 12

**Exit** **AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadPauseCtrl**

13 Clear Source

**Action** Clears a stereo volume source already set up for Fade or Pan effect, or which is currently active in either of these processes.

Entry BX = 13

**AX** = *wVolSource*

**DX** = *wEffect*

**Exit AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadClrSource**

15 Get Pan Position

**Action** Gets the Pan position of the specified stereo volume source.

Entry BX = 15

**AX** = *wVolSource*

**Exit AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadGetPanPosition**

17 Reset Mixer

**Action** Resets the mixer chip to its default state.

Entry BX = 17

**Exit** None

**Details In** AUXDRV.DRV HLL Interface **ctadResetMixer**

21 Set Gain

**Action** Sets the gain of the input or output mixer.

Entry BX = 21

**AX** = *wMixer*

**DX** = *wGain*

**Exit AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadSetMixerGain**

22 Get Gain

**Action** Reads the gain of either the input or output mixer.

Entry BX = 22

**AX** = *wMixer*

**Exit AX** = *wGain*

**Details In** AUXDRV.DRV HLL Interface **ctadGetMixerGain**

23 Set Tone

**Action** Sets the Treble or Bass tone level.

Entry BX = 23

**AX** = *wTone*

**DX** = *wLevel*

**Exit** **AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadSetToneLevel**

24 Get Tone

**Action** Reads the current Treble or Bass tone setting.

Entry BX = 24

**AX** = *wTone*

**Exit AX** = *wLevel*

**Details In** AUXDRV.DRV HLL Interface **ctadGetToneLevel**

25 Set AGC

**Action** Turns on or off the AGC (automatic gain control) of the microphone input.

Entry BX = 25

**AX** = *wfOnOff*

**Exit AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadSetAGC**

26 Get AGC

**Action** Reads the AGC status.

Entry BX = 26

**Exit AX** = *wfOnOff*

**Details In** AUXDRV.DRV HLL Interface **ctadGetAGC**

27 Set Mixer Switches

**Action** Sets the input or output mixer switches.

Entry BX = 27

**CX** = *wMixer*

**DX:AX** = *dwSwitches*

**Exit AX** = error code

**Details In** AUXDRV.DRV HLL Interface **ctadSetMixerSwitch**

28 Get Mixer Switches

**Action** Reads the input or output mixer switches.

Entry BX = 28

**CX** = *wMixer*

**Exit DX:AX** = *dwSwitches*

**Details In** AUXDRV.DRV HLL Interface **ctadGetMixerSwitch**

29 Get Environment Settings

**Action** Passes the BLASTER environment string to the driver for it to interpret the hardware settings to use.

Entry BX = 29

**ES:DI** = *lpszBlaster*

**Exit AX** = error code

**Details In** AUXDRV.DRV HLL Interface ctadGetEnvSettings

Chapter 3

Creative Multimedia System Driver

This chapter documents the interfaces to Creative Multimedia System driver, CTMMSYS.SYS.

CTMMSYS.SYS is a DOS device driver that mediates access to Creative's sound devices. This driver is installed with an entry in CONFIG.SYS. The syntax of the entry is:

DEVICE=<*full path to CTMMSYS.SYS*>

In the following document, we use two dummy keywords IN and OUT as the status of a variable to help you identify whether a variable is sent by the application on entry to the driver, or output by the driver on return to the application. These dummy keywords have been defined in the header file as follow:

#define IN

#define OUT

An application passes information relating to a variable with status IN to the driver. The driver uses a variable with status OUT to return information back to the application.

Device Driver Entry-Point

An application interfaces with CTMMSYS via the entry-point of the CTMMSYS driver. The entry-point of CTMMSYS is expected to be invoked with a far call, with function arguments pushed onto the stack in the Pascal calling convention, i.e. from left to right.

To help you invoke the CTMMSYS services, the following **typedef** is supplied in the header file CTMMSYS.H:

**typedef MMSTATUS (FAR PASCAL \*MMSYSPROC*)*(**

IN **HMMDEVICE** *hDev*,

IN **MMDEVICE** *MmDevice*,

IN **WORD**  *wMsg*,

IN OUT **DWORD** *dwParam1*,

IN OUT **DWORD** *dwParam2*);

**Parameters** *hDev*

Specifies the handle of the target device.

*MmDevice*

Specifies the type of device.

*wMsg*

Specifies the message being sent to the driver.

*dwParam1*

Specifies a message-dependent parameter.

*dwParam2*

Specifies a message-dependent parameter

**Remarks** *hDe*v is the handle assigned by the driver when the device was opened.

The *MmDevice* types available are:

|  |  |
| --- | --- |
| **Type** | **Description** |
| **MMDEVICE\_SOUNDOUT** | playback device |
| **MMDEVICE\_SOUNDIN** | recording device |
| **MMDEVICE\_CSP** | signal processing device |
| **MMDEVICE\_AUX** | auxiliary device |

Note that **MMDEVICE\_SOUNDOUT** or **MMDEVICE\_SOUNDIN** should be used with messages for sound devices; **MMDEVICE\_AUX** should be used with messages for auxiliary audio devices, and **MMDEVICE\_CSP** should be used for signal processing device.

All unused arguments must be set to **PARAM\_UNUSED**.

**Return Value** The driver returns a message dependent **MMSTATUS\_**error code.

There are two general error codes that are consistent across the full spectrum of messages.

**MMSTATUS\_SUCCESS** is returned if the operation is successful.

**MMSTATUS\_UNSUPPORTED\_MSG** is returned if the specified message is not supported by the driver.

This new type **MMSYSPROC** should be used to define storage for a variable which, after having been assigned the entry-point address to CTMMSYS, may be used to invoke the driver.

Callback Function

CTMMSYS device driver needs to notify applications when certain events occur, such as when a sound data buffer has been played, or has been recorded. When the application opens a device, it specifies a callback function for use by the driver.

**SOUNDCALLBACK** is pointer to a function type defined as follows:

typedef void

(FAR PASCAL \*SOUNDCALLBACK)(

HMMDEVICE hDev,

WORD wMsg,

DWORD dwCallbackData,

DWORD dwParam1,

DWORD dwParam2);

where

**hDev** specifies the handle identifying the device;

**wMsg**specifies a word storage where the driver returns a **SOM\_BUFFERDONE** or **SIM\_BUFFERDONE** message to the application:

**SOM\_BUFFERDONE** indicates that a sound output buffer has been processed;

**SIM\_BUFFERDONE** indicates that a sound input buffer has been processed;

**dwCallbackData** specifies the 32 bits of user data supplied by the application when the device was opened;

**dwParam1** is a variable of type **LPSOUNDBUFFER**(refer to the section on **Sound Device Data Structure**) used to store the information of the returned buffer;

**dwParam2** is unused.

Device Driver Messages

The following section lists the messages supported by the CTMMSYS device driver.

Auxiliary Audio Device Messages

The auxiliary audio device messages can be divided into basic and optional categories. Basic messages are those messages supported by the auxiliary audio device regardless of the Sound Blaster card used. Optional messages are those messages supported with regard to the mixer capabilities of the Sound Blaster card used.

Basic Messages

The auxiliary audio device supports the following basic messages.

|  |  |
| --- | --- |
| **Message** | **Meaning** |
| **AUXDM\_QUERY\_NumDevs** | Query number of device supported |
| **AUXDM\_QUERY\_Capabilities** | Query the capabilities of a device |
| **AUXDM\_CONFIGURATION\_Query** | Query BLASTER environment variable |
| **AUXDM\_OPEN** | Open a device |
| **AUXDM\_CLOSE** | Close a device |

Optional Messages

The auxiliary audio device supports the following messages optionally, depending on the capabilities of the target hardware.

|  |  |
| --- | --- |
| **Message** | **Meaning** |
| **AUXDM\_VOLUME\_QueryCaps** | Query volume control capabilities |
| **AUXDM\_VOLUME\_Get** | Get volume |
| **AUXDM\_VOLUME\_Set** | Set volume |
| **AUXDM\_MIXING\_QueryCaps** | Query mixing control capabilities |
| **AUXDM\_MIXING\_Get** | Get mixer switches |
| **AUXDM\_MIXING\_Set** | Set mixer switches |
| **AUXDM\_FILTER\_QueryCaps** | Query filter control capabilities |
| **AUXDM\_FILTER\_Get** | Get filter status |
| **AUXDM\_FILTER\_Set** | Set filter on or off |

|  |  |
| --- | --- |
| **Message** | **Meaning** |
| **AUXDM\_TONE\_QueryCaps** | Query tone control capabilities |
| **AUXDM\_TONE\_Get** | Get tone level |
| **AUXDM\_TONE\_Set** | Set tone level |
| **AUXDM\_GAIN\_QueryCaps** | Query gain control capabilities |
| **AUXDM\_GAIN\_Get** | Get gain constant |
| **AUXDM\_GAIN\_Set** | Set gain |
| **AUXDM\_AGC\_QueryCaps** | Query AGC control capabilities |
| **AUXDM\_AGC\_Get** | Get AGC status |
| **AUXDM\_AGC\_Set** | Set AGC on or off |
| **AUXDM\_MISC\_Reset** | Reset mixer |

Sound Device Messages

The playback and recording sound devices support the following messages:

|  |  |
| --- | --- |
| **Message** | **Meaning** |
| **S*x*DM\_QUERY\_NumDevs** | Query total no. of a specified device type |
| **S*x*DM\_QUERY\_Capabilities** | Query capabilities of a specified device |
| **S*x*DM\_QUERY\_SamplingRange** | Query max. and min. sampling rates |
| **S*x*DM\_QUERY\_TransferBuffer** | Query information of transfer buffer needed |
| **S*x*DM\_CONFIGURATION\_Query** | Query BLASTER environment variable |
| **S*x*DM\_OPEN** | Open specified device and allocate handle |
| **S*x*DM\_CLOSE** | Close specified device and deallocate handle |
| **S*x*DM\_STATE\_Query** | Query state of a specified device |
| **S*x*DM\_STATE\_Set** | Set state of a specified device |
| **S*x*DM\_BUFFERQUEUE\_Query** | Query status of buffer queue |
| **S*x*DM\_BUFFERQUEUE\_Add** | Add new buffer to queue |
| **S*x*DM\_POSITION\_Query** | Query current playback or recording position |
| **S*x*DM\_MISC\_SetSpeaker** | Set speaker on or off |

The **'*x*'** in the messages is to be replaced with either **'I'** for input or **'O**' for output.

Signal Processing Device Messages

The signal processing devices support the following messages:

|  |  |
| --- | --- |
| **Message** | **Meaning** |
| **CSPDM\_QUERY\_NumDevs** | Query number of device supported |
| **CSPDM\_QUERY\_Capabilities** | Query capabilities of a specified device |
| **CSPDM\_CONFIGURATION\_Query** | Query BLASTER environment variable |
| **CSPDM\_OPEN** | Open specified device and allocate handle |
| **CSPDM\_CLOSE** | Close specified device and deallocate handle |
| **CSPDM\_STATE\_Set** | Set state of a specified device |
| **CSPDM\_CODE\_Download** | Download micro-code to signal processor |

Device Driver Data Structures

Each message sent to the device driver entry-point comes with two DWORD parameters. These parameters contain information passed between the application and the driver and are stored in data structures defined in the three categories. These categories are Common, Auxiliary Audio Device, and Sound Device and Signal Processing Device.

Common Data Structure

A common data structure is used by both an auxiliary audio device and a sound device. It is listed as follow:

DEVCONFIG

A structure filled with driver information providing details to the BLASTER environment string.

Auxiliary Audio Device Data Structures

Auxiliary audio device uses the data structures listed as follows:

AUXCAPS

A structure filled with driver information providing an application with details on the capabilities of an auxiliary audio device.

AUXOPEN

A structure filled with application information providing details on opening an auxiliary audio device.

AUXSETTINGS

A structure filled with driver and application information providing details on the settings of an audio source.

AUXVOLUMECAPS

A structure filled with driver information providing details on volume control.

AUXMIXINGCAPS

A structure filled with driver information providing details on mixing control.

AUXFILTERCAPS

A structure filled with driver information providing details on filter control.

AUXTONECAPS

A structure filled with driver information providing details on tone control.

AUXGAINCAPS

A structure filled with driver information providing details on gain control.

AUXAGCCAPS

A structure filled with driver information providing details on AGC control.

Sound Device Data Structures

Sound devices use the data structures listed as follows:

MEMORYDESC

A structure filled with driver and application information providing details on the memory buffer.

MMTIME

A structure filled with driver and application information providing details on the current playback or recording position.

SOUNDBUFFER

A structure filled with driver and application information providing details on the buffer to add to the queue of a specified device.

SOUNDBUFQ

A structure filled with driver information providing details on the status of the buffer queue of a specified device.

SOUNDCAPS

A structure filled with driver information describing the capabilities of a sound device.

SOUNDFORMAT

A structure filled with application information describing the format of data.

SOUNDOPEN

A structure filled with driver and application information providing details needed by the driver when devices are opened.

SOUNDQYXFERBUF

A structure filled with driver information providing details on the transfer buffer.

SOUNDSAMPLINGRANGE

A structure filled with driver and application information providing the minimum and maximum sampling rates of a sound device.

SOUNDXFERBUFDESC

A structure filled with application information providing the TRANSFER buffer information.

Signal Processing Device Data Structures

Signal processing device uses the data structures listed as follows:

CSPCAPS

A structure filled with driver information describing the capabilities of a signal processing device.

CSPCODEDOWNLOAD

A structure filled with application information providing the buffer containing the signal processor's micro-code.

CSPOPEN

A structure filled with driver and application information providing details needed by the driver when device is opened.

Device Driver Error Messages

The following table lists the error codes that will be returned by CTMMSYS:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **MMSTATUS\_SUCCESS** | No error |
| **MMSTATUS\_DRIVER\_BUSY** | Driver is busy |
| **MMSTATUS\_ERROR** | Unspecified error |
| **MMSTATUS\_ALLOCATED** | Requested device already allocated |
| **MMSTATUS\_UNALLOCATED** | Device not allocated |
| **MMSTATUS\_UNSUPPORTED\_MSG** | Invalid message send to driver |
| **MMSTATUS\_BAD\_HANDLE** | Invalid device handle supplied |
| **MMSTATUS\_BAD\_DEVICEID** | Invalid device ID supplied |
| **MMSTATUS\_BAD\_FLAG** | Invalid flag status supplied |
| **MMSTATUS\_BAD\_PARAMETER** | Invalid parameter supplied |
| **MMSTATUS\_REDUNDANT\_ACTION** | Unnecessary action |
| **MMSTATUS\_BUFFER\_TOO\_SMALL** | Buffer size too small |
| **MMSTATUS\_NOT\_ENABLED** | Driver is not enabled |
| **MMSTATUS\_BAD\_FORMAT** | Unsupported data format supplied |
| **MMSTATUS\_STILL\_ACTIVE** | Device still in use |
| **MMSTATUS\_NO\_MEMORY** | Memory related error |

Device Driver Message Reference

Applications communicate with the CTMMSYS driver through messages sent to the driver. The device driver will then dispatch to the respective routines.

This section contains an alphabetical list of all messages that can be received by the driver. The message reference is subdivided into the following sections:

Auxiliary Audio Device Message Reference

Sound Device Message Reference

Signal Processing Device Message Reference

A message consists of a handle, the device type, a message number and two DWORD parameters. Message numbers are identified by predefined message names. The two DWORD parameters contain message-dependent values. Recall that device types available are:

|  |  |
| --- | --- |
| **Type** | **Description** |
| **MMDEVICE\_SOUNDOUT** | playback device |
| **MMDEVICE\_SOUNDIN** | recording device |
| **MMDEVICE\_AUX** | auxiliary device |
| **MMDEVICE\_AUX** | signal processing device |

Note that **MMDEVICE\_SOUNDOUT** or **MMDEVICE\_SOUNDIN** should be used with messages for sound devices; **MMDEVICE\_AUX** should be used with messages for auxiliary audio devices, and **MMDEVICE\_CSP** should be used with messages for signal processing devices.

Auxiliary Audio Device Message Reference

**AUXDM\_AGC\_Get**

**Action** Gets the Automatic Gain Control (AGC) status.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up AUXSETTINGS.*dwItem* with the constant **AUX\_SOURCE\_MIC** to obtain microphone input AGC status.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, so it is set to zero. The AGC status is reported in the fields AUXSETTINGS.*dwLeft* as **AUX\_AGC\_ON** or **AUX\_AGC\_OFF**.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_AGC\_Set

**AUXDM\_AGC\_QueryCaps**

**Action** Queries the capabilities of AGC control.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXAGCCAPS data structure.

dwParam2

Unused.

**Remarks** Upon return, AUXAGCCAPS.*dwSource* contains a value which is a bit-or'ed combination of the sources supporting AGC control. Please refer to **AUXDM\_VOLUME\_Get** for the source constants.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also AUXDM\_AGC\_Get

**AUXDM\_AGC\_Set**

**AUXDM\_VOLUME\_Get**

**AUXDM\_AGC\_Set**

**Action** Sets AGC on or off.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up AUXSETTINGS.*dwItem* with the constant **AUX\_SOURCE\_MIC** to set microphone input AGC, and AUXSETTINGS.*dwLeft* with setting **AUX\_AGC\_ON** or **AUX\_AGC\_OFF**.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, so it is set to zero.

If the AGC is being turn off, a constant gain will be used.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_AGC\_Get

**AUXDM\_CLOSE**

**Action** Closes and deallocates an auxiliary audio device.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Unused.

dwParam2

Unused.

**Remarks** None

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also AUXDM\_OPEN

**AUXDM\_CONFIGURATION\_Query**

**Action** Ascertains whether the configuration of the card matches that of the BLASTER environment variable.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a DEVCONFIG data structure.

dwParam2

Unused.

**Remarks** DEVCONFIG.*wDeviceID* can only accept any number in the range of one less than the number of auxiliary device available. DEVCONFIG.*dwFlags* should be set to zero.

Upon completion, the BLASTER environment string will be returned in the field DEVCONFIG.*szzConfiguration*, unless a bad device ID is given. But the validity of returned string depends on the returned value. If the returned value is **MMSTATUS\_SUCCESS**, the settings have tested okay; otherwise, the returned value is **MMSTATUS\_NOT\_ENABLED**.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

See Also SxDM\_CONFIGURATION\_Query

**AUXDM\_FILTER\_Get**

**Action** Obtains the input or output low pass filter status and confirms whether it is on or off.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application fills up AUXSETTINGS.*dwItem* with constant **AUX\_FILTER\_INPUT** or **AUX\_FILTER\_OUTPUT** depending on the input or output filter status.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, so it is set to zero. The filter status will be returned in the AUXSETTINGS.*dwLeft*. This field will be **AUX\_FILTER\_OFF** or **AUX\_FILTER\_ON** for output filter status, or **AUX\_FILTER\_OFF**, **AUX\_FILTER\_HIGH** or **AUX\_FILTER\_LOW** for input filter status.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_FILTER\_Set

**AUXDM\_FILTER\_QueryCaps**

**Action** Queries the capabilities of filter control.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXFILTERCAPS data structure.

dwParam2

Unused.

**Remarks** Upon return, AUXFILTERCAPS.*dwFlags* contains information on the filter control, which is a bit-or'ed combination of the following:

**AUXFILTERCAPS\_INPUT\_MONO**

**AUXFILTERCAPS\_OUTPUT\_MONO**

**AUXFILTERCAPS\_INPUT\_STEREO**

**AUXFILTERCAPS\_OUTPUT\_STEREO**

The filter control indicates whether the device is in mono or stereo mode, and whether it is used for input or output (or both).

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also AUXDM\_FILTER\_Get

**AUXDM\_FILTER\_Set**

**AUXDM\_FILTER\_Set**

**Action** Sets input or output filter off or on.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up the fields AUXSETTINGS.*dwItem* with **AUX\_FILTER\_INPUT** or **AUX\_FILTER\_OUTPUT**, and AUXSETTINGS.*dwLeft* with **AUX\_FILTER\_ON**, **AUX\_FILTER\_OFF**, **AUX\_FILTER\_LOW** or **AUX\_FILTER\_HIGH** accordingly.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, so it is set to zero.

To turn input filter status on, use the constants **AUX\_FILTER\_LOW** or **AUX\_FILTER\_HIGH** instead of **AUX\_FILTER\_ON** because the input filter provides two different levels of cut-off frequencies.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_FILTER\_Get

**AUXDM\_GAIN\_Get**

**Action** Obtains the input or output gain constant.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up AUXSETTINGS.*dwItem* with constant **AUX\_GAIN\_INPUT** or **AUX\_GAIN\_OUTPUT** depending on input or output gain constant to be retrieved.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, so it is set to zero. The left and right channels gain constant will be returned in the fields AUXSETTINGS.*dwLeft* and AUXSETTINGS.*dwRight* respectively.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_GAIN\_Set

**AUXDM\_GAIN\_QueryCaps**

**Action** Queries the capabilities of gain control.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXGAINCAPS data structure.

dwParam2

Unused.

**Remarks** Upon return, AUXGAINCAPS.*dwFlags* contains information on the gain control, which is a bit-or'ed combination of the following:

**AUXGAINCAPS\_INPUT\_MONO**

**AUXGAINCAPS\_OUTPUT\_MONO**

**AUXGAINCAPS\_INPUT\_STEREO**

**AUXGAINCAPS\_OUTPUT\_STEREO**

The gain control indicates whether the device is in mono or stereo mode, and whether it is used for input or output (or both).

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also AUXDM\_GAIN\_Get

**AUXDM\_GAIN\_Set**

**AUXDM\_GAIN\_Set**

**Action** Sets input or output gain.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up the AUXSETTINGS.*dwItem* field with **AUX\_GAIN\_INPUT** or **AUX\_GAIN\_OUTPUT**. The requested left and right channels gain constant is set in the fields AUXSETTINGS.*dwLeft* and AUXSETTINGS.*dwRight* respectively.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, so it is set to zero. The supported gain constant ranges from 0 to 3.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_GAIN\_Get

**AUXDM\_MISC\_Reset**

**Action** Resets the mixer to its default state.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Unused.

dwParam2

Unused.

**Remarks** None

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also None

**AUXDM\_MIXING\_Get**

**Action** Obtains the setting of the input or output mixer switches.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up AUXSETTINGS.*dwItem* with either **AUX\_MIXING\_INPUT** or **AUX\_MIXING\_OUTPUT** constant depending on either input or output mixer switches is to be retrieved.

dwParam2

Unused.

**Remarks** To get a complete mixer setting, invoke **MMSYSPROC** to process this message twice with different flag values. Set AUXSETTINGS.*dwFlags* to **AUX\_MIXING\_LEFT** and **AUX\_MIXING\_RIGHT** to obtain left and right channels setting of the audio sources.

If the operation is successful, the mixer setting is a bit-or'ed combination of the constants **AUX\_SOURCE\_MIDI**, **AUX\_SOURCE\_CD**, **AUX\_SOURCE\_LINEIN** and **AUX\_SOURCE\_MIC** which will be placed in the AUXSETTINGS.*dwLeft* and AUXSETTINGS.*dwRight* fields, for the left and right channels mixing path respectively.

Please be reminded that, if there is no separate left and right channels control on the mixing path, the mixer settings can only be obtained from AUXSETTINGS.*dwLeft* field only.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_MIXING\_Set

**AUXDM\_MIXING\_QueryCaps**

**Action** Queries the capabilities of mixing control.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXMIXINGCAPS data structure.

dwParam2

Unused.

**Remarks** Upon return, AUXMIXINGCAPS.*dwFlags* contains information on the mixing control, which is a bit-or'ed combination of the following:

**AUXMIXINGCAPS\_INPUT\_MONO**

**AUXMIXINGCAPS\_OUTPUT\_MONO**

**AUXMIXINGCAPS\_INPUT\_STEREO**

**AUXMIXINGCAPS\_OUTPUT\_STEREO**

**AUXMIXINGCAPS\_INPUT\_INDIVIDUAL\_LR**

**AUXMIXINGCAPS\_OUTPUT\_INDIVIDUAL\_LR**

**AUXMIXINGCAPS\_INPUT\_MULTIPLE**

**AUXMIXINGCAPS\_OUTPUT\_MULTIPLE**

A mixing path is available to indicate whether the device is in mono or stereo mode, and whether it is used for input or output (or both). If it is stereo mode, it provides a separate left and right channels path control. It also indicates whether the device supports multiple input or output mixing.

AUXMIXINGCAPS.*dwInputSource* and AUXMIXINGCAPS.*dwOutputSource* indicate the available mixing sources. It is a bit-or'ed combination of the followings:

**AUX\_SOURCE\_MIDI**

**AUX\_SOURCE\_CD**

**AUX\_SOURCE\_LINEIN**

**AUX\_SOURCE\_MIC**

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also AUXDM\_MIXING\_Get

**AUXDM\_MIXING\_Set**

**AUXDM\_MIXING\_Set**

**Action** Sets input or output mixer switches.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up the fields:

AUXSETTINGS.*dwItem* with **AUX\_MIXING\_INPUT** or **AUX\_MIXING\_OUTPUT**.

AUXSETTINGS.*dwFlags* with **AUX\_MIXING\_LEFT** or **AUX\_MIXING\_RIGHT**.

AUXSETTINGS.*dwLeft* and AUXSETINGS.*dwRight* should be filled with a value which is a bit-or'ed combination of the constants **AUX\_SOURCE\_MIDI**, **AUX\_SOURCE\_CD**, **AUX\_SOURCE\_LINEIN** and **AUX\_SOURCE\_MIC**.

dwParam2

Unused.

**Remarks** A complete setup for the left and right channels of the sources will need a separate call with AUXSETTINGS.*dwFlags* set as **AUX\_MIXING\_LEFT** and **AUX\_MIXING\_RIGHT** accordingly.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_MIXING\_Get

**AUXDM\_OPEN**

**Action** Opens and allocates an auxiliary device.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to an AUXOPEN data structure.

dwParam2

Unused.

**Remarks** AUXOPEN.*wDeviceID* can only accept any number in the range of one less than the number of device available. For example, if only one auxiliary device is available, set AUXOPEN.*wDeviceID* to zero. AUXOPEN.*dwFlags* is always set to zero.

Upon return, AUXOPEN.*hDev* will contain the handle to the target device, which is needed by all subsequent calls to the auxiliary device.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_ALLOCATED**

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

See Also AUXDM\_CLOSE

**AUXDM\_QUERY\_Capabilities**

**Action** Queries the capabilities of an auxiliary device.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to an AUXCAPS data structure.

dwParam2

Unused.

**Remarks** AUXCAPS.*wDeviceID* can only accept any number in the range of one less than the number of device available. For example, if only one auxiliary device available, set AUXCAPS.*wDeviceID* to zero.

Upon return, AUXCAPS will contain all necessary information regarding the target device. For example, the hardware ID, the audio sources supported, and the available controls and so on are situated in the AUXCAPS fields.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

See Also SxDM\_QUERY\_Capabilities

**AUXDM\_QUERY\_NumDevs**

**Action** Queries the number of auxiliary device support.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a word *wNumDevs*.

dwParam2

Unused.

**Remarks** If no auxiliary devices are installed, the contents of *wNumDevs* will be set to zero. Otherwise, it will be set to any number depending on number of devices supported.

On knowing the number of devices supported, the subsequent basic messages should be called with a device ID ranging from 0 to one less than the *wNumDevs*.

Return Value MMSTATUS\_SUCCESS

See Also None

**AUXDM\_TONE\_Get**

**Action** Gets the tone level.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up AUXSETTINGS.*dwItem* with constant **AUX\_TONE\_TREBLE** or **AUX\_TONE\_BASS** depending on whether the treble or bass tone level is to be retrieved.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, it is set to zero. The tone level for the left and right channels will be returned in the fields AUXSETTINGS.*dwLeft* and AUXSETTINGS.*dwRight* respectively.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_TONE\_Set

**AUXDM\_TONE\_QueryCaps**

**Action** Queries the capabilities of tone control.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXTONECAPS data structure.

dwParam2

Unused.

**Remarks** Upon return, AUXTONECAPS.*dwTone* specifies the available tone control. It is a bit-or'ed combination of:

**AUX\_TONE\_TREBLE**

**AUX\_TONE\_BASS**

The field AUXTONECAPS.*dwStereo* indicates whether the available tone control is in mono or stereo mode. If the tone control is in stereo mode, the corresponding bit (**AUX\_TONE\_XXX**) will be set to one.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also AUXDM\_TONE\_Get and AUXDM\_TONE\_Set

**AUXDM\_TONE\_Set**

**Action** Sets treble or bass tone level.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up the AUXSETTINGS.*dwItem* field with **AUX\_TONE\_TREBLE** or **AUX\_TONE\_BASS**. The requested left and right channels tone level is set in the fields AUXSETTINGS.*dwLeft* and AUXSETTINGS.*dwRight* respectively.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, it is set to zero.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_TONE\_Get

**AUXDM\_VOLUME\_Get**

**Action** Obtains volume level of a specified source.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an **AUXSETTINGS** data structure. The application sets up AUXSETTINGS.*dwItem* with a specified source constant.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, so it is set to zero.

The available constants for AUXSETTINGS.*dwItem* are:

**AUX\_SOURCE\_MASTER**

**AUX\_SOURCE\_VOICE**

**AUX\_SOURCE\_MIDI**

**AUX\_SOURCE\_CD**

**AUX\_SOURCE\_LINEIN**

**AUX\_SOURCE\_MIC**

**AUX\_SOURCE\_PCSPEAKER**

The left and right channels volume level of a stereo source will be returned in the AUXSETTINGS.*dwLeft* and AUXSETTINGS.*dwRight* fields respectively. For a mono source, the volume level will be returned in the AUXSETTINGS.*dwLeft* field only.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_VOLUME\_Set

**AUXDM\_VOLUME\_QueryCaps**

**Action** Queries the capabilities of volume control.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXVOLUMECAPS data structure.

dwParam2

Unused.

**Remarks** Upon return, AUXTONECAPS.*dwStereo* specifies whether the supported sources are mono or stereo. For a stereo source, the corresponding bit (according to the bit setting of **AUX\_SOURCE\_XXX**) will be set to one.

**AUX\_SOURCE\_MASTER**

**AUX\_SOURCE\_VOICE**

**AUX\_SOURCE\_MIDI**

**AUX\_SOURCE\_CD**

**AUX\_SOURCE\_LINEIN**

**AUX\_SOURCE\_MIC**

**AUX\_SOURCE\_PCSPEAKER**

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also AUXDM\_VOLUME\_Get

**AUXDM\_VOLUME\_Set**

**AUXDM\_VOLUME\_Set**

**Action** Sets volume level of a specified source.

**Parameters** *hDev*

Specifies the handle to the target device.

dwParam1

Specifies a far pointer to an AUXSETTINGS data structure. The application sets up the AUXSETTINGS.*dwItem* fields with a specified source constant. The requested left and right channels volume level is set in the fields AUXSETTINGS.*dwLeft* and AUXSETTINGS.*dwRight* respectively.

dwParam2

Unused.

**Remarks** AUXSETTINGS.*dwFlags* is not used, so it is set to zero. To setup a volume level for a mono source, just specify the requested volume level in the field AUXSETTINGS.*dwLeft*.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_FLAG**

**MMSTATUS\_BAD\_PARAMETER**

See Also AUXDM\_VOLUME\_Get

Sound Device Message Reference

**S*x*DM\_BUFFERQUEUE\_Add**

**Action** Adds a buffer of data for playback or storage of recorded data to the buffer queue of a specified device.

**Parameters** *hDev*

Specifies a device handle obtained from previous **S*x*DM\_OPEN** call.

dwParam1

Specifies a far pointer to a SOUNDBUFFER structure containing the application supplied address and size of the buffer, and the information returned from the driver.

dwParam2

Unused.

**Remarks** SOUNDBUFFER.*dwFlags* must be set to zero on entry. The driver will set the **SOUNDBUFFER\_INQUEUE** bit and place the data buffer in its playback queue or recording queue. Once the data buffer has been processed, the driver will set the **SOUNDBUFFER\_DONE** bit and clear the **SOUNDBUFFER\_INQUEUE** bit.

SOUNDBUFFER.*lpNext* and SOUNDBUFFER.*dwReserved* are not to be used or altered by the application.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_PARAMETER**

**MMSTATUS\_NO\_MEMORY**

**MMSTATUS\_NO\_MEMORY** is returned if driver runs out of memory for internal support structures.

See Also S*x*DM\_BUFFERQUEUE\_Query

**S*x*DM\_BUFFERQUEUE\_Query**

**Action** Queries the status of the buffer queue of a specified device.

**Parameters** *hDev*

Specifies a device handle obtained from previous **S*x*DM\_OPEN** call.

dwParam1

Specifies a far pointer to a SOUNDBUFQ structure containing the driver returned information.

dwParam2

Unused.

**Remarks** Use this function to check if the driver has run out of buffers to playback data or buffers to record data.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also S*x*DM\_BUFFERQUEUE\_Add

**S*x*DM\_CLOSE**

**Action** Closes a specified device and deallocates the handle.

**Parameters** *hDev*

Specifies a device handle obtained from previous **S*x*DM\_OPEN** call.

dwParam1

Unused.

dwParam2

Unused.

**Remarks** When the device is closed, the device handle will no longer be valid.

If there are data buffers that have not been processed by the driver, the driver will fail or the close operation and return a **MMSTATUS\_STILL\_ACTIVE** error.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_STILL\_ACTIVE**

See Also S*x*DM\_OPEN

**S*x*DM\_CONFIGURATION\_Query**

**Action** Queries the BLASTER environment variable.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a DEVCONFIG structure containing the driver returned BLASTER environment.

dwParam2

Unused.

**Remarks** If the return value is **MMSTATUS\_SUCCESS**, then DEVCONFIG.*szzConfiguration*, minimally, points to

'BLASTER=A:220 I:5 D:1 H:5',0,0

Note that to cater for future expansion where multiple configuration strings may exist, each individual configuration string is terminated with a null character; the entire assembly of configuration strings is terminated with two null characters.

If the return value is **MMSTATUS\_BAD\_DEVICEID** or **MMSTATUS\_NOT\_ENABLED**, then DEVCONFIG.*szzConfiguration* points to {0,0}, i.e. the two null characters.

DEVCONFIG.*dwFlags* must be set to zero on entry.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

See Also None

**S*x*DM\_MISC\_SetSpeaker**

**Action** Sets the speaker on or off for a specified device.

**Parameters** *hDev*

Specifies a device handle obtained from previous **S*x*DM\_OPEN** call.

dwParam1

Specifies the status of the speaker supplied by an application. It must be set to one of the following values:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **SPEAKER\_OFF** | turn off speaker |
| **SPEAKER\_ON** | turn on speaker |

wParam2

Unused.

**Remarks** This message has no effect on Sound Blaster 16.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_PARAMETER**

See Also None

**S*x*DM\_OPEN**

**Action** Opens a specified device and allocates a device handle for use by the application.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a SOUNDOPEN structure containing the information supplied by the application such as the format of the sound data, the call-back address, and the driver returned handle value. All subsequent calls to the driver must use this handle value for the *hDev* argument.

dwParam2

Unused.

**Remarks** This function must be called in order to get the device handle SOUNDOPEN.*hDev* to be used by other messages.

On entry, the SOUNDFORMAT variable must be initialized, and SOUNDOPEN.*lpFormat* must be set to the address of the SOUNDFORMAT variable.

Set SOUNDOPEN.*lpXferBufDesc* to NULL when the call to **S*x*DM\_QUERY\_TransferBuffer** indicates that a transfer buffer is not needed.

The transfer buffer passed to the driver will be used only for the lifetime of the open handle. When the device is closed, the handle is no longer valid and the transfer buffer is considered by the driver to have been returned to the application.

When the device is no longer needed, the message **S*x*DM\_CLOSE** must be sent to free the device.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_ALLOCATED**

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

**MMSTATUS\_BAD\_FORMAT**

**MMSTATUS\_NO\_MEMORY**

**MMSTATUS\_NO\_MEMORY** is returned if a transfer buffer is required but was not provided, or the transfer buffer provided did not satisfy the required conditions.

See Also S*x*DM\_CLOSE

**S*x*DM\_POSITION\_Query**

**Action** Queries the current playback or recording position of a specified device.

**Parameters** *hDev*

Specifies a device handle obtained from previous **S*x*DM\_OPEN** call.

dwParam1

Specifies a far pointer to a MMTIME structure containing unit of measurement used (supplied by the application), and number of units returned by the driver.

wParam2

Unused.

**Remarks** The position is relative to the start of the current playback or the current recording.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_PARAMETER**

See Also None

**S*x*DM\_QUERY\_Capabilities**

**Action** Queries the capabilities of a specified device.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a SOUNDCAPSstructure containing the information on the capabilities of the specified device returned by the driver.

dwParam2

Unused.

**Remarks** None

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

See Also None

**S*x*DM\_QUERY\_NumDevs**

**Action** Queries the total number of a specified device type available

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a word storage containing the total number of the specified device type returned by the driver.

dwParam2

Unused.

**Remarks** If no device of the specified type is present, then the total number of devices of the specified type would be set to zero.

If the devices of the specified type are present, then the device ID would range from zero to one less than the total number of devices of this type.

**Return Value MMSTATUS\_SUCCESS**

See Also None

**S*x*DM\_QUERY\_SamplingRange**

**Action** Queries the minimum and maximum sampling rates of a specified device.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a SOUNDSAMPLINGRANGE structure containing the minimum and maximum sampling rates of the specified device returned by the driver.

dwParam2

Unused.

**Remarks** None

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_BAD\_PARAMETER**

**MMSTATUS\_NOT\_ENABLED**

See Also None

**S*x*DM\_QUERY\_TransferBuffer**

**Action** Queries the information of the transfer buffer needed for a specified device.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a SOUNDQYXFERBUFstructure containing the information of the transfer buffer used for the specified device returned by the driver.

dwParam2

Unused.

**Remarks** If dwcbMinSize and dwcbMaxSize are both zero, then no transfer buffer is needed.

The valid size (in bytes) of the transfer buffer must satisfy the following condition:

size = dwcbMinSize + n\*dwcbGranularity dwcbMaxSize

where n is an integer 0.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_DEVICEID**

See Also S*x*DM\_BUFFERQUEUE\_Query

**S*x*DM\_BUFFERQUEUE\_Add**

**S*x*DM\_STATE\_Query**

**Action** Queries the state of a specified device.

**Parameters** *hDev*

Specifies a device handle obtained from a previous **S*x*DM\_OPEN** call.

dwParam1

Specifies a far pointer to a SOUNDSTATE storage containing the driver returned state of the device.

The storage can take one of the following values:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **SOUNDSTATE\_IDLE** | data transfer has not started or has ended. |
| **SOUNDSTATE\_ACTIVE** | data transfer is on going. |
| **SOUNDSTATE\_PAUSED** | data transfer has temporarily stopped. |

dwParam2

Unused.

**Remarks** None

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also S*x*DM\_STATE\_Set

**S*x*DM\_STATE\_Set**

**Action** Sets the state of a specified device.

**Parameters** *hDev*

Specifies a device handle obtained from a previous **S*x*DM\_OPEN** call.

dwParam1

Specifies a SOUNDSTATE variable containing the application supplied information to set the device in a specified state.

The possible values for *dwParam1* are:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **SOUNDSTATE\_START** | start data transfer |
| **SOUNDSTATE\_STOP** | pause data transfer |
| **SOUNDSTATE\_RESET** | end data transfer |

dwParam2

Unused.

**Remarks** Use this function to start, stop, or reset the sound input or output.

For **SOUNDSTATE\_RESET**, the driver is playing, record, or pause state will immediately terminate the state and mark all data buffers in the buffer queue as done. The driver will then notify the application by using the callback function to send a done message for each data buffer.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_PARAMETER**

**MMSTATUS\_REDUNDANT\_ACTION**

See Also S*x*DM\_STATE\_Query

Signal Processing Device Message Reference

**CSPDM\_CLOSE**

**Action** Closes a specified device and deallocates the handle.

**Parameters** *hDev*

Specifies a device handle obtained from previous **CSPDM\_OPEN** call.

dwParam1

Unused.

dwParam2

Unused.

**Remarks** When the device is closed, the device handle will no longer be valid.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

See Also CSPDM\_OPEN

**CSPDM\_CONFIGURATION\_Query**

**Action** Queries the BLASTER environment variable.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a DEVCONFIG structure containing the driver returned BLASTER environment.

dwParam2

Unused.

**Remarks** If the return value is **MMSTATUS\_SUCCESS**, then DEVCONFIG.*szzConfiguration*, minimally, points to

'BLASTER=A:220',0,0

Note that to cater for future expansion where multiple configuration strings may exist, each individual configuration string is terminated with a null character; the entire assembly of configuration strings is terminated with two null characters.

If the return value is **MMSTATUS\_BAD\_DEVICEID** or **MMSTATUS\_NOT\_ENABLED**, then DEVCONFIG.*szzConfiguration* points to {0,0}, i.e. the two null characters.

DEVCONFIG.*dwFlags* must be set to zero on entry.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

See Also None

**CSPDM\_CODE\_Download**

**Action** Downloads code into a specified device.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a CSPCODEDOWNLOAD structure.

dwParam2

Unused.

**Remarks** CSPCODEDOWNLOAD.*lpCode* specifies the far pointer to the buffer holding the code to be downloaded. CSPCODEDOWNLOAD.*dwcbCode* specifies the length of the code in number of bytes. CSPCODEDOWNLOAD.*dwFlags* indicates the type of code of the following:

|  |  |
| --- | --- |
| **Constant** | **Description** |
| **CSPCODEDOWNLOAD\_INITCODE** | compression and decompression code |

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_ERROR**

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_PARAMETER**

See Also None

**CSPDM\_OPEN**

**Action** Opens a specified device and allocates a device handle for use by the application.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a CSPOPEN structure.

dwParam2

Unused.

**Remarks** CSPOPEN.*wDeviceID* can only accept any number in the range of one less than the number of the device available. For example, if only one signal processing device is available, set CSPOPEN.*wDeviceID* to zero. Currently, DEVCONFIG.*dwFlags* is set to zero.

When the device is no longer needed, the message **CSPDM\_CLOSE** must be sent to free the device.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_ALLOCATED**

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

See Also CSPDM\_CLOSE

**CSPDM\_QUERY\_Capabilities**

**Action** Queries the capabilities of a specified device.

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a CSPCAPSstructure.

dwParam2

Unused.

**Remarks** Upon return, CSPCAPS will contain all necessary information regarding the target device.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_DEVICEID**

**MMSTATUS\_NOT\_ENABLED**

See Also None

**CSPDM\_QUERY\_NumDevs**

**Action** Queries the total number of a specified device type available

**Parameters** *hDev*

Unused.

dwParam1

Specifies a far pointer to a word storage containing the total number of the specified device type returned by the driver.

dwParam2

Unused.

**Remarks** If no device of the specified type is present, then the total number of devices of the specified type would be set to zero.

If the devices of the specified type are present, then the device ID would range from zero to one less than the total number of devices of this type.

**Return Value MMSTATUS\_SUCCESS**

See Also None

**CSPDM\_STATE\_Set**

**Action** Sets the state of a specified device.

**Parameters** *hDev*

Specifies a device handle obtained from a previous **CSPDM\_OPEN** call.

dwParam1

Specifies a CSPSTATE variable containing the application supplied information to set the device in a specified state.

The possible values for *dwParam1* are:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **CSPSTATE\_ACTIVE** | set signal processor to active mode |
| **CSPSTATE\_STANDBY** | set signal processor to stand-by mode |
| **CSPSTATE\_INACTIVE** | set signal processor to inactive mode |

dwParam2

Unused.

**Remarks** Use this function to set the signal processor to active, stand-by or inactive modes..

**CSPSTATE\_STANDBY** is used to pause the signal processor. It can be set back to active mode using **CSPSTATE\_ACTIVE**.

**Return Value MMSTATUS\_SUCCESS** if the operation is successful. Otherwise, the following error code may occur:

**MMSTATUS\_BAD\_HANDLE**

**MMSTATUS\_BAD\_PARAMETER**

**MMSTATUS\_ERROR**

See Also CSPDM\_CODE\_Download

Device Driver Data Structure Reference

This section lists the data structures used by the CTMMSYS driver in alphabetical order. For each data structure, the structure definition is given, followed by a description of each field.

The data structure reference is subdivided into the following sections:

Common Data Structure Reference

Auxiliary Audio Device Data Structure Reference

Sound Device Data Structure Reference

Signal Processing Device Data Structure Reference

Common Data Structure Reference

DEVCONFIG

The DEVCONFIG structure gives the information of the BLASTER environment variable.

typedef struct

{

IN WORD wDeviceID;

IN OUT DWORD dwFlags;

OUT LPCSTR szzConfiguration;

} DEVCONFIG,

FAR \*LPDEVCONFIG;

**Fields** **wDeviceID**

Specifies an ID for the device.

dwFlags

Unused

szzConfiguration

Specifies the BLASTER environment string. Minimally, it points to

'BLASTER=A:220 I:5 D:1 H:5',0,0

Note that to cater for future expansion where multiple configuration strings may exist, each individual configuration string is terminated with a null character; the entire assembly of configuration strings is terminated with two null characters.

See Also None

Auxiliary Audio Device Data Structure Reference

AUXAGCCAPS

The AUXAGCCAPS structure describes the capabilities of the AGC control.

typedef struct

{

OUT DWORD dwFlags;

OUT DWORD dwSource;

} AUXAGCCAPS,

FAR \*LPAUXAGCCAPS;

**Fields** **dwFlags**

Unused.

dwSource

Specifies the AGC sources. The value will be a bit-or'ed combination of the following:

|  |
| --- |
| **AUX\_SOURCE\_MASTER** |
| **AUX\_SOURCE\_VOICE** |
| **AUX\_SOURCE\_MIDI** |
| **AUX\_SOURCE\_CD** |
| **AUX\_SOURCE\_LINEIN** |
| **AUX\_SOURCE\_MIC** |
| **AUX\_SOURCE\_PCSPEAKER** |

**See Also** None

AUXCAPS

The AUXCAPS structure describes the capabilities of an auxiliary device.

typedef struct

{

IN WORD wDeviceID;

OUT DWORD dwDriverVersion;

OUT WORD wProduct;

OUT char szProductName[MMCAPS\_PRODUCTNAME\_MAXLEN];

OUT DWORD dwSupport;

OUT DWORD dwSource;

} AUXCAPS,

FAR \*LPAUXCAPS;

**Fields** **wDeviceID**

Specifies the target device ID. The ID value must be within the range zero to one less than the number of devices supported.

dwDriverVersion

Specifies the driver version number and built number. The high word containing the major and minor version number in high byte and low byte respectively. The low word containing the driver built number.

wProduct

Specifies the member the hardware device belongs to. The high byte and low byte represents the family and the member number respectively. Possible values are:

|  |
| --- |
| **MMPRODUCT\_SB** |
| **MMPRODUCT\_SB\_2** |
| **MMPRODUCT\_SBPRO** |
| **MMPRODUCT\_SB16** |

szProductName

A full string for the product name.

dwSource

Specifies the audio sources supported by the target device. The value is a bit-or'ed combination of the following:

|  |
| --- |
| **AUX\_SOURCE\_MASTER** |
| **AUX\_SOURCE\_VOICE** |
| **AUX\_SOURCE\_MIDI** |
| **AUX\_SOURCE\_CD** |
| **AUX\_SOURCE\_LINEIN** |
| **AUX\_SOURCE\_MIC** |
| **AUX\_SOURCE\_PCSPEAKER** |

dwSupport

Specifies the available controls by the target device.

|  |
| --- |
| **AUXCAPS\_SUPPORT\_VOLUME** |
| **AUXCAPS\_SUPPORT\_MIXING** |
| **AUXCAPS\_SUPPORT\_FILTER** |
| **AUXCAPS\_SUPPORT\_TONE** |
| **AUXCAPS\_SUPPORT\_GAIN** |
| **AUXCAPS\_SUPPORT\_AGC** |

See Also SxDM\_QUERY\_Capabilities

AUXFILTERCAPS

The AUXFILTERCAPS structure describes the capabilities of the filter control.

typedef struct

{

OUT DWORD dwFlags;

DWORD dwReserved;

} AUXFILTERCAPS,

FAR \*LPAUXFILTERCAPS;

**Fields** **dwFlags**

Specifies the input and output filter control mode. The value will be a bit-or'ed combination of the following:

|  |
| --- |
| **AUXFILTERCAPS\_INPUT\_MONO** |
| **AUXFILTERCAPS\_OUTPUT\_MONO** |
| **AUXFILTERCAPS\_INPUT\_STEREO** |
| **AUXFILTERCAPS\_OUTPUT\_STEREO** |

**See Also** None

AUXGAINCAPS

The AUXGAINCAPS structure describes the capabilities of the gain control.

typedef struct

{

OUT DWORD dwFlags;

DWORD dwReserved;

} AUXGAINCAPS,

FAR \*LPAUXGAINCAPS;

**Fields** **dwFlags**

Specifies the input and output gain control mode. The value will be a bit-or'ed combination of the following:

|  |
| --- |
| **AUXGAINCAPS\_INPUT\_MONO** |
| **AUXGAINCAPS\_OUTPUT\_MONO** |
| **AUXGAINCAPS\_INPUT\_STEREO** |
| **AUXGAINCAPS\_OUTPUT\_STEREO** |

**See Also** None

AUXMIXINGCAPS

The AUXMIXINGCAPS structure describes the capabilities of the mixing control.

typedef struct

{

OUT DWORD dwFlags;

OUT DWORD dwInputSource;

OUT DWORD dwOutputSource;

} AUXMIXINGCAPS,

FAR \*LPAUXMIXINGCAPS;

**Fields** **dwFlags**

Specifies the input and output mixing path control mode. The value will be a bit-or'ed combination of the following:

|  |
| --- |
| **AUXMIXINGCAPS\_INPUT\_MONO** |
| **AUXMIXINGCAPS\_OUTPUT\_MONO** |
| **AUXMIXINGCAPS\_INPUT\_STEREO** |
| **AUXMIXINGCAPS\_OUTPUT\_STEREO** |
| **AUXMIXINGCAPS\_INPUT\_INDIVIDUAL\_LR** |
| **AUXMIXINGCAPS\_OUTPUT\_INDIVIDUAL\_LR** |
| **AUXMIXINGCAPS\_INPUT\_MULTIPLE** |
| **AUXMIXINGCAPS\_OUTPUT\_MULTIPLE** |

dwInputSource, dwOutputSource

Specifies the mixing sources supported. The value will be a bit-or'ed combination of the following:

|  |
| --- |
| **AUX\_SOURCE\_MASTER** |
| **AUX\_SOURCE\_VOICE** |
| **AUX\_SOURCE\_MIDI** |
| **AUX\_SOURCE\_CD** |
| **AUX\_SOURCE\_LINEIN** |
| **AUX\_SOURCE\_MIC** |
| **AUX\_SOURCE\_PCSPEAKER** |

**See Also** None

AUXOPEN

The AUXOPEN structure contains information needed when auxiliary device is opened with **AUXDM\_OPEN** message.

typedef struct

{

IN WORD wDeviceID;

OUT HMMDEVICE hDev;

IN DWORD dwFlags;

} AUXOPEN,

FAR \*LPAUXOPEN;

**Fields** **wDeviceID**

Specifies the target device ID. The ID value must fall within the range zero and one less than the number of devices supported.

hDev

Specifies the application's handle to the auxiliary device, assigned by CTMMSYS. Use this handle when invoking the target device with other messages.

dwFlags

Reserved.

See Also None

AUXSETTINGS

The AUXSETTINGS structure contains information needed by the driver to perform the required settings.

typedef struct

{

IN DWORD dwItem;

IN OUT DWORD dwFlags;

IN OUT DWORD dwLeft;

IN OUT DWORD dwRight;

} AUXSETTINGS,

FAR \*LPAUXSETTINGS;

**Fields** **dwItem**

Specifies the required setting.

dwFlags

Specifies the left or right channel of audio source.

dwLeft

Specifies the left channel data.

dwRight

Specifies the right channel data.

See Also None

AUXTONECAPS

The AUXTONECAPS structure describes the capabilities of the tone control.

typedef struct

{

OUT DWORD dwFlags;

OUT DWORD dwTone;

OUT DWORD dwStereo;

} AUXTONECAPS,

FAR \*LPAUXTONECAPS;

**Fields** **dwFlags**

Unused.

dwTone

Specifies the tone control sources supported. The value is a bit-or'ed combination of the following:

|  |
| --- |
| **AUX\_TONE\_TREBLE** |
| **AUX\_TONE\_BASS** |

dwStereo

Specifies the tone control mode. If it is stereo mode, the value is a bit‑or'ed combination of the following:

|  |
| --- |
| **AUX\_TONE\_TREBLE** |
| **AUX\_TONE\_BASS** |

**See Also** None

AUXVOLUMECAPS

The AUXVOLUMECAPS structure describes the capabilities of the volume control.

typedef struct

{

OUT DWORD dwFlags;

OUT DWORD dwStereo;

} AUXVOLUMECAPS,

FAR \*LPAUXVOLUMECAPS;

**Fields** **dwFlags**

Unused.

dwStereo

Specifies the mode of supported source. If it is a stereo source, the value is a bit‑or'ed combination of the following:

|  |
| --- |
| **AUX\_SOURCE\_MASTER** |
| **AUX\_SOURCE\_VOICE** |
| **AUX\_SOURCE\_MIDI** |
| **AUX\_SOURCE\_CD** |
| **AUX\_SOURCE\_LINEIN** |
| **AUX\_SOURCE\_MIC** |
| **AUX\_SOURCE\_PCSPEAKER** |

**See Also** None

Sound Device Data Structure Reference

MEMORYDESC

typedef struct

{

WORD wType;

union

{

LPBYTE lpMem;

struct

{

WORD wHandle;

DWORD dwOffset;

} xms;

} u;

DWORD dwReserved;

} MEMORYDESC,

FAR \*LPMEMORYDESC;

**Fields** **wType**

Specifies the type of memory. It is a bit-or of the following values:

**MEMORYDESC\_MEM** for conventional memory;

**MEMORYDESC\_XMS** for extended memory.

lpMem

Specifies the far pointer to the conventional memory when conventional memory is used.

wHandle, dwOffset

Specifies the handle and offset address to the extended memory when extended memory is used.

dwReserved

Reserved.

See Also None

MMTIME

The MMTIME structure gives the current playback or recording position.

typedef struct

{

IN OUT WORD wType;

OUT union

{

DWORD dwMillisecs;

DWORD dwSamples;

DWORD dwBytes;

} u;

} MMTIME,

FAR \*LPMMTIME;

**Fields** **wType**

Specifies the unit of measurement. It is set to one of the following values:

**MMTIME\_MILLISECS**

**MMTIME\_SAMPLES**

**MMTIME\_BYTES**

dwMillisecs

Specifies the duration (in millisecond) of the current playback or recording.

dwSamples

Specifies the number of samples of data that have been currently played back or recorded.

dwBytes

Specifies the number of bytes of data that have been currently played back or recorded.

See Also None

SOUNDBUFFER

The SOUNDBUFFER structure presents the information of the buffer to be added to the queue of a specified device.

typedef struct \_SOUNDBUFFER

{

IN MEMORYDESC Buffer;

IN DWORD dwcbBufferSize;

OUT DWORD dwcbRecorded;

IN DWORD dwUserData;

IN OUT DWORD dwFlags;

struct \_SOUNDBUFFER FAR lpNext;

DWORD dwReserved;

} SOUNDBUFFER,

FAR \*LPSOUNDBUFFER;

**Fields** **Buffer**

Specifies the information of the buffer.

dwcbBufferSize

Specifies the size of the buffer in bytes. It must be a multiple of SOUNDFORMAT.*wBlockAlign*.

If the buffer is in extended memory, the minimum value must be 2. This is because extended memory services won't handle odd length moves.

dwcbRecorded

Specifies the number of bytes of data that have been recorded for the buffer.

dwUserData

Specifies 32 bits of user data.

dwFlags

Specifies flags giving status of the buffer. It will be set as follows:

**SOUNDBUFFER\_DONE**

Set by the device driver to indicate that the data buffer has been processed, and is being returned to the application.

**SOUNDBUFFER\_INQUEUE**

Set by the device driver to indicate that the data buffer is queued for playback or recording.

lpNext

Reserved for use by the device driver to point to the next SOUNDBUFFER structure in the queue.

dwReserved

Reserved for use by the device driver.

See Also SOUNDBUFQ

**SOUNDFORMAT**

SOUNDBUFQ

The SOUNDBUFQ structure presents the status of the buffer queue of a specified device.

typedef struct

{

OUT DWORD dwFlags;

OUT DWORD dwcbQueuedData;

} SOUNDBUFQ,

FAR \*LPSOUNDBUFQ;

**Fields** **dwFlags**

Specifies whether more buffers are required. It will be set to **SOUNDBUFQ\_NEEDBUFFERS** if the device has no more buffers to playback data or to store recorded data. Otherwise, it will be set to 0.

dwcbQueueData

Specifies either the total number of bytes remaining in the output buffer queue for playback, or in the input buffer queue available for recording.

See Also SOUNDBUFFER

SOUNDCAPS

The SOUNDCAPS structure describes the capabilities of a sound device.

typedef struct

{

IN WORD wDeviceID;

OUT DWORD dwDriverVersion;

OUT WORD wProduct;

OUT char szProductName[MMCAPS\_PRODUCTNAME\_MAXLEN];

OUT WORD wChannels;

OUT DWORD dwFlags;

} SOUNDCAPS,

FAR \*LPSOUNDCAPS;

**Fields** **wDeviceID**

Specifies an ID for the sound device.

dwDriverVersion

Specifies the version number and build number of the device driver for the sound device where

Word1 = version number

where high-byte = major, and low-byte = minor

Word0 = build number

wProduct

Specifies the member of the product. Note that a new member is created if the programming of the DSP is different.

It will be set to one of the following values:

**MMPRODUCT\_SB**

**MMPRODUCT\_SB\_2**

**MMPRODUCT\_SBPRO**

**MMPRODUCT\_SB16**

szProductName

Specifies the product name in a NULL-terminated string. For example, it can be set to "Creative Sound Blaster 16".

wChannels

Specifies whether the sound device is a mono or a stereo device. It will be set to one of the following values:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 1 | mono device |
| 2 | stereo device |

dwFlags

Unused.

See Also None

SOUNDFORMAT

The SOUNDFORMAT structure describes the format of data.

typedef struct

{

IN WORD wFormatTag;

IN WORD wFormatFamily;

IN WORD wChannels;

IN DWORD dwSamplesPerSec;

IN WORD wBlockAlign;

IN WORD wBitsPerSample;

IN WORD wcbExtraSize;

} SOUNDFORMAT,

FAR \*LPSOUNDFORMAT;

**Fields** **wFormatTag**

Specifies the format type.

If wFormatFamily = **SOUNDFORMAT\_FAMILY\_WAVE** then wFormatTag can be as follows:

**WAVE\_FORMAT\_PCM**

**WAVE\_FORMAT\_ALAW**

**WAVE\_FORMAT\_MULAW**

**WAVE\_FORMAT\_CREATIVE\_ADPCM**

If wFormatFamily = **SOUNDFORMAT\_FAMILY\_CREATIVE** then wFormatTag can be as follows:

**CREATIVE\_FORMAT\_ADPCM**

wFormatFamily

Specifies the format family. The family types are as follows:

**SOUNDFORMAT\_FAMILY\_WAVE** for data from .WAV family

**SOUNDFORMAT\_FAMILY\_CREATIVE** for data from Creative family

wChannels

Specifies the number of channels in the data. It is set as follows:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 1 | mono data |
| 2 | stereo data |

dwSamplesPerSec

Specifies the sampling rate in samples per second.

wBlockAlign

Specifies the block alignment in bytes. The block alignment is the minimum atomic unit of data. For PCM data, the block alignment is the number of bytes used by a single sample, including data for both channels if the data is stereo. For example, the block alignment for 16-bit stereo PCM is 4 bytes (2 channels, 2 bytes per sample).

The driver will validate this field and return error if the value is wrong.

dwBitsPerSample

Specifies the number of bytes per sample of data.

wcbExtraSize

Specifies the number bytes of extra information in the extended wave format header.

See Also SOUNDOPEN

SOUNDOPEN

The SOUNDOPEN structure contains information needed by the driver when devices are opened.

typedef struct

{

IN WORD wDeviceID;

OUT HMMDEVICE hDev;

IN LPSOUNDFORMAT lpFormat;

IN DWORD dwFlags;

IN SOUNDCALLBACK Callback;

IN DWORD dwCallbackData;

IN LPSOUNDXFERBUFDESC lpXferBufDesc;

} SOUNDOPEN,

FAR \*LPSOUNDOPEN;

**Fields** **wDeviceID**

Specifies an ID for the sound device.

hDev

Specifies the handle to the sound device.

lpFormat

Specifies a far pointer to a SOUNDFORMAT structure, indicating the data format requested by the application.

dwFlags

Specifies option flags for opening the device.

**SOUNDOPEN\_QUERYFORMAT**

If this flag is set on entry, the application can find out whether the driver supports a specified sound data format in the SOUNDFORMAT structure. **The driver will not open the device in this case**, but will return **MMSTATUS\_SUCCESS** if it supports the requested format, or **MMSTATUS\_BAD\_FORMAT** if it does not. To open the device, set the flag to zero.

Callback

Specifies the address of a callback function. The driver uses this information to notify the application via the callback function.

dwCallbackData

Specifies 32 bits of user data by the application. This information is returned to the application whenever the driver notifies the application using the callback function.

lpXferBufDesc

Specifies a far pointer to a SOUNDXFERBUFDESC structure, containing the information of the buffer.

See Also SOUNDFORMAT, SOUNDXFERBUFDESC

SOUNDQYXFERBUF

The SOUNDQYXFERBUF structure presents the information of the transfer buffer.

typedef struct

{

IN WORD wDeviceID;

OUT WORD wMemoryDescType;

OUT WORD wBlockAlign;

OUT DWORD dwcbMinSize;

OUT DWORD dwcbMaxSize;

OUT DWORD dwcbGranularity;

OUT DWORD dwFlags;

} SOUNDQYXFERBUF,

FAR \*LPSOUNDQYXFERBUF;

**Fields** **wDeviceID**

Specifies an ID for the sound device.

wMemoryDescType

Specifies whether the transfer buffer is to be in the conventional or extended memory or both. It will be set to the following values:

**MEMORYDESC\_MEM** for conventional memory;

**MEMORYDESC\_XMS** for extended memory.

wBlockAlign

Specifies the block alignment for the transfer buffer. For example, if the block alignment is 2, then the transfer buffer is required to start only at an even address. If the block alignment is 16, then the transfer buffer is required to start only at a paragraph. The transfer buffer can start at any arbitrary address if the block alignment is one.

dwcbMinSize

Specifies the minimum size of the transfer buffer in bytes.

dwcbMaxSize

Specifies the maximum size of the transfer buffer in bytes.

dwcbGranularity

Specifies the step size for the increment of the transfer buffer size from the minimum value.

dwFlags

Specifies whether the transfer buffer can cross a 64K or 128K page boundary. It will be set to a bit‑or'ed combination of the following values:

**SOUNDQYXFERBUF\_CANNOTCROSS64KBPAGE**

**SOUNDQYXFERBUF\_CANNOTCROSS128KBPAGE**

See Also SOUNDXFERBUFDESC

SOUNDSAMPLINGRANGE

The SOUNDSAMPLINGRANGE structure presents the minimum and maximum sampling rates of a sound device.

typedef struct

{

IN WORD wDeviceID;

IN WORD wChannels;

OUT DWORD dwMinSamplesPerSec;

OUT DWORD dwMaxSamplesPerSec;

} SOUNDSAMPLINGRANGE,

FAR \*LPSOUNDSAMPLINGRANGE;

**Fields** **wDeviceID**

Specifies an ID for the sound device.

**wChannels**

Specifies whether the sampling rates are for mono or stereo data transfer. It is set to one of the following values:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 1 | mono data transfer |
| 2 | stereo data transfer |

dwMinSamplesPerSec

Specifies the minimum sampling rate of the device.

dwMinSamplesPerSec

Specifies the maximum sampling rate of the device.

See Also **None**

SOUNDXFERBUFDESC

The **SOUNDXFERBUFDESC** structure presents the information of the transfer buffer.

typedef struct

{

IN MEMORYDESC Buffer;

IN DWORD dwcbBufferSize;

} SOUNDXFERBUFDESC,

FAR \*LPSOUNDXFERBUFDESC;

**Fields** **Buffer**

Specifies the information of the transfer buffer.

dwcbBufferSize

Specifies the size of the transfer buffer in bytes.

See Also SOUNDOPEN, SOUNDQYXFERBUF

Signal Processing Device Structure Reference

CSPCAPS

The CSPCAPS structure describes the capabilities of a signal processing device.

typedef struct

{

IN WORD wDeviceID;

OUT DWORD dwDriverVersion;

OUT WORD wProduct;

OUT char szProductName[MMCAPS\_PRODUCTNAME\_MAXLEN];

OUT DWORD dwFlags;

DWORD dwReserved;

} CSPCAPS,

FAR \*LPCSPCAPS;

**Fields** **wDeviceID**

Specifies an ID for the signal processing device.

dwDriverVersion

Specifies the version number and build number of the device driver for the signal processing device where

Word1 = version number

where high-byte = major, and low-byte = minor

Word0 = build number

wProduct

Specifies the member of the product. It will be set to one of the following values:

**MMPRODUCT\_SB**

**MMPRODUCT\_SB\_2**

**MMPRODUCT\_SBPRO**

**MMPRODUCT\_SB16**

szProductName

Specifies the product name in a NULL-terminated string. For example, it can be set to "Creative Sound Blaster 16".

dwFlags

Unused.

dwReserved

Unused.

See Also None

CSPCODEDOWNLOAD

The CSPCODEDOWNLOAD structure describes the information of code to be downloaded to a signal processor.

typedef struct

{

IN LPBYTE lpCode;

IN DWORD dwcbCode;

IN DWORD dwFlags;

DWORD dwReserved;

} CSPCODEDOWNLOAD,

FAR \*LPCSPCODEDOWNLOAD;

**Fields** **lpCode**

Specifies the far pointer to the buffer holding the code to be downloaded.

dwcbCode

Specifies the length of the code in number of bytes.

dwFlags

Specifies the type of code to be downloaded. It is set as follows:

|  |  |
| --- | --- |
| **Constant** | **Description** |
| **CSPCODEDOWNLOAD\_INITCODE** | compression and decompression code |

dwReserved

Unused.

See Also None

CSPOPEN

The CSPOPEN structure contains information needed by the driver when devices are opened.

typedef struct

{

IN WORD wDeviceID;

OUT HMMDEVICE hDev;

IN LPSOUNDFORMAT lpSoundFormat;

IN DWORD dwFlags;

} CSPOPEN,

FAR \*LPCSPOPEN;

**Fields** **wDeviceID**

Specifies an ID for the signal processing device.

hDev

Specifies the handle to the signal processing device.

lpSoundFormat

Specifies a far pointer to a SOUNDFORMAT structure, indicating the data format requested by the application.

dwFlags

Unused.

See Also None

Chapter 4

MIDI Driver

This chapter documents the interfaces to the Creative loadable MIDI driver (CTMIDI.DRV).

It is divided into two parts. The first part covers high-level language interfaces and the second covers the register base interfaces. Cross-references to the register base interface are provided in the high-level language interfaces.

High-Level Language Interface

This section describes the high-level language interface to the CTMIDI.DRV driver in alphabetical order.

Function Prefix

High-level function names begin with the following prefix:

|  |  |
| --- | --- |
| **Prefix** | **Driver** |
| ctmd | CTMIDI |

Include Files

The followings are the required include files for the CTMIDI driver:

|  |  |  |
| --- | --- | --- |
| **C Language** | **Turbo Pascal** | **Microsoft Basic** |
| SBKMIDI.H | SBKMIDI.INC | SBKMIDI.BI |

Functions by Category

The CTMIDI driver functions may be divided into the following categories:

|  |  |
| --- | --- |
| **Category** | **Function** |
|  |  |
| Initialization/Termination | **ctmdGetEnvSettings** |
|  | **ctmdGetMidiEnvSettings** |
|  | **ctmdInit** |
|  | **ctmdResetMidiDriver** |
|  | **ctmdTerminate** |
|  |  |
|  |  |
| Setup | **ctmdSetChannelMapper** |
|  | **ctmdSetInputStatusAddx** |
|  | **ctmdSetMapperType** |
|  | **ctmdSetMidiInputBuffer** |
|  | **ctmdSetMidiInputCallBackFunct** |
|  | **ctmdSetOutputStatusAddx** |
|  | **ctmdSetTimeStampMode** |
|  |  |
|  |  |
| Query | **ctmdGetMapperType** |
|  |  |
|  |  |
| Control | **ctmdPauseMidiMusic** |
|  | **ctmdPlayMidiMusic** |
|  | **ctmdPrepareMidiStart** |
|  | **ctmdResumeMidiMusic** |
|  | **ctmdSetMusicTempo** |
|  | **ctmdSetMusicTranspose** |
|  | **ctmdStartMidiInput** |
|  | **ctmdStopMidiInput** |
|  | **ctmdStopMidiMusic** |
|  |  |
|  |  |
| Additional Control | **ctmdSendShortMessage** |
|  | **ctmdSendLongMessage** |
|  |  |
|  |  |
| Miscellaneous | **ctmdGetDrvVer** |
|  |  |
|  |  |

Manifest Constants

The following constants and their meaning have been defined in the include file to help you use the CTMIDI driver functions.

Synthesizer Type

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **INTERN\_SYNTH** | Internal Sound Blaster Synthesizer Chip |
| **EXTERN\_SYNTH** | External MIDI device |

Mapper Type

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **GENERAL\_MIDI\_MAPPER** | General Mapper Format |
| **EXTENDED\_MIDI\_MAPPER** | External Mapper Format |
| **BASIC\_MIDI\_MAPPER** | Basic Mapper Format |
| **USER\_DEFINED\_MAPPER** | User-defined Mapper Format |

Time Stamp Mode

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **DIFFERENTIATE\_MODE** | Differentiate mode |
| **ELAPSED\_MODE** | Elapsed mode |

**ctmdGetDrvVer**

**Action** Gets the version number of the MIDI driver.

**Syntax** **C** **WORD ctmdGetDrvVer( void )**

**Pascal** **ctmdGetDrvVer :word**

**Basic** **ctmdGetDrvVer%( )**

**Parameters** None

**Remarks** None

**Return Value** Major version number in the high byte. Minor version number in the low byte.

See Also None

ASM Interface CTMIDI.DRV Function 0

**ctmdGetEnvSettings**

**Action** Passes the BLASTER environment string to the driver for it to interpret which hardware settings to use.

**Syntax C WORD ctmdGetEnvSettings( const char far** *\*lpzBlaster* **)**

**Pascal** **ctmdGetEnvSettings(** *lpszBlaster***: pointer ) :word**

**Basic ctmdGetEnvSettings%(***lpszBlaster&* **)**

**Parameters** *lpszBlaster*

Far pointer to the BLASTER environment string, without the "BLASTER**=**" prefix

**Remarks** The driver will determine:

1. the Base I/O Address,

or 2. IRQ, in the case of MIDI input is to be activated.

or 3. the MPU-401 Base I/O Address (Sound Blaster 16 only).

Since we have enforced the concept of using the environment variables to signify information of our Sound Blaster cards, thus this function must be called before **ctmdInit** as the driver presumed no default hardware settings.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdGetMidiEnvSettings

**ctmdInit**

ASM Interface CTMIDI.DRV Function 2

**ctmdGetMapperType**

**Action** Queries the current mapper type.

**Syntax C WORD ctmdGetMapperType( void )**

**Pascal ctmdGetMapperType :word**

**Basic ctmdGetMapperType%( )**

**Parameters** None

**Remarks** None

**Return Value** Current mapper type. One of the followings is returned.

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **GENERAL\_MIDI\_MAPPER** | General Mapper Format |
| **EXTENDED\_MIDI\_MAPPER** | External Mapper Format |
| **BASIC\_MIDI\_MAPPER** | Basic Mapper Format |
| **USER\_DEFINED\_MAPPER** | User-defined Mapper Format |

See Also ctmdGetMidiEnvSettings

**ctmdSetMapperType**

ASM Interface CTMIDI.DRV Function 13

**ctmdGetMidiEnvSettings**

**Action** Passes the MIDI environment string to the driver for it to interpret which synthesizer type and channel mapper to use.

**Syntax** **C** **WORD ctmdGetMidiEnvSettings( const char far *\*****lpszMidi* **)**

**Pascal ctmdGetMidiEnvSettings(** *lpszMidi*: **pointer ) :word**

**Basic ctmdGetMidiEnvSettings%(** *lpszMidi&* **)**

**Parameters** *lpszMidi*

Far pointer to the MIDI environment string (without the "MIDI**=**" prefix).

**Remarks**  This function is optional if application prefer to use the default settings for synthesizer type and mapper. By default, the driver will use the internal music synthesizer and extended mapper for playback.

To allow the driver to use settings other than default, this function must be invoked before **ctmdInit**. Application can change to other mapper type even when the MIDI music is playing. This is achieved by invoking **ctmdSetMapperType**.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdGetEnvSettings

**ctmdInit**

ASM Interface CTMIDI.DRV Function 1

**ctmdInit**

**Action** Initializes the driver.

**Syntax C WORD ctmdInit( void )**

**Pascal ctmdInit :word**

**Basic ctmdInit%( )**

**Parameters** None

**Remarks**  This is the necessary step to activate the driver before any MIDI activity can take place. This function is responsible for hooking the necessary interrupt(s).

**Return Value** Zero if successful.

See Also ctmdTerminate

ASM Interface CTMIDI.DRV Function 3

**ctmdPauseMidiMusic**

**Action** Pauses playing MIDI music.

**Syntax C WORD ctmdPauseMidiMusic ( void )**

**Pascal ctmdPauseMidiMusic :word**

**Basic ctmdPauseMidiMusic% ( )**

**Parameters** None

**Remarks**  To resume, call **ctmdResumeMidiMusic**.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdPrepareMidiStart

**ctmdPlayMidiMusic**

**ctmdStopMidiMusic**

**ctmdResumeMidiMusic**

ASM Interface CTMIDI.DRV Function 11

**ctmdPlayMidiMusic**

**Action** Starts playing the MIDI file.

**Syntax** **C WORD ctmdPlayMidiMusic( void )**

**Pascal ctmdPlayMidiMusic :word**

**Basic ctmdPlayMidiMusic%( )**

**Parameters** None

**Remarks** This function must be called after **ctmdPrepareMidiStart** to start playing.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdPrepareMidiStart

**ctmdStopMidiMusic**

**ctmdPauseMidiMusic**

**ctmdResumeMidiMusic**

ASM Interface CTMIDI.DRV Function 9

**ctmdPrepareMidiStart**

**Action** Pre-processes the MIDI file to play.

**Syntax C WORD ctmdPrepareMidiStart( const BYTE** *\*lpszMidiBuffer* **)**

**Pascal ctmdPrepareMidiStart(** *lpszMidiBuffer* **: pointer) :word**

**Basic ctmdPrepareMidiStart%(** *lpszMidiBuffer&* **)**

**Parameters** *lpszMidiBuffer*

Far pointer to the first byte of the buffer where the MIDI file is loaded.

**Remarks** This function has to be called after **ctmdInit** and before **ctmdPlayMidiMusic**. It processes the file parameter such as resolution of the MIDI (ticks per quarter note), number of tracks etc.

The buffer to process must be in a standard MIDI file format which is preceded by the 4 byte '**MThd**' MIDI file ID.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdPlayMidiMusic

**ctmdStopMidiMusic**

**ctmdPauseMidiMusic**

**ctmdResumeMidiMusic**

ASM Interface CTMIDI.DRV Function 8

**ctmdResetMidiDriver**

**Action** Resets MIDI driver.

**Syntax C WORD ctmdResetMidiDriver( void )**

**Pascal ctmdResetMidiDriver :word**

**Basic ctmdResetMidiDriver%( )**

**Parameters** None

**Remarks** This function restores MIDI driver parameters to their default state. These include timer speed and mapper type.

Return Value Zero if successful. Non-zero otherwise.

See Also None

ASM Interface CTMIDI.DRV Function 5

**ctmdResumeMidiMusic**

**Action** Resumes the paused MIDI music.

**Syntax C WORD ctmdResumeMidiMusic( void )**

**Pascal ctmdResumeMidiMusic :word**

**Basic ctmdResumeMidiMusic%( )**

**Parameters** None

**Remarks**  None

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdPrepareMidiStart

**ctmdPlayMidiMusic**

**ctmdStopMidiMusic**

**ctmdPauseMidiMusic**

ASM Interface CTMIDI.DRV Function 12

**ctmdSendLongMessage**

**Action** Sends a buffer of MIDI data to the external MIDI synthesizer via the MIDI port.

**Syntax C WORD ctmdSendLongMessage (char far *\*****lpMsg***,**

**WORD** *wMsgLen* **)**

**Pascal ctmdSendLongMessage(** *lpMsg* **:pointer;** *wMsgLen* **word) :word**

**Basic ctmdSendLongMessage%(** *lpMsg&***,** *wMsgLen%* **)**

**Parameters** *lpMsg*

Far pointer to the starting of the MIDI message.

wMsgLen

The length of the message to be sent in bytes.

**Remarks**  Use this function to send multiple MIDI events, (including system exclusive messages).

For buffer that contains multiple MIDI events, each MIDI event must be separated by a delta time (as on the standard MIDI file format). Presently, delta time on the **ctmdSendLongMessage** will be ignored by the MIDI driver. Also, MIDI driver will use the current MIDI tempo and transpose to send the MIDI events.

On Sound Blaster 16, MPU-401 MIDI Port will be used for MIDI data transfer if external synthesizer is selected. For other Sound Blaster cards, SB-MIDI Port will be used.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdSendShortMessage

ASM Interface CTMIDI.DRV Function 18

**ctmdSendShortMessage**

**Action** Sends a MIDI event to the external MIDI synthesizer via the MIDI port.

**Syntax C WORD ctmdSendShortMessage( WORD** *wMidiStatus***,**

**WORD** *wMidiData1***, WORD** *wMidiData2* **)**

**Pascal ctmdSendShortMessage(** *wMidiStatus***,** *wMidiData1***,**

*wMidiData2***:word ) :word**

**Basic ctmdSendShortMessage%(** *wMidiStatus%***,**

*wMidiData1%***,** *wMidiData2%* **)**

**Parameters** *wMidiStatus*

MIDI status byte.

wMidiData1

First MIDI data byte.

wMidiData2

Second MIDI data byte.

**Remarks** Use this function to send short MIDI event. Use **ctmdSendLongMessage** to send system exclusive messages.

On Sound Blaster 16, MPU-401 MIDI Port will be used for MIDI data transfer if external synthesizer is selected. For other Sound Blaster cards, SB-MIDI Port will be used.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdSendLongMessage

ASM Interface CTMIDI.DRV Function 17

**ctmdSetChannelMapper**

**Action** Sets the channel mapping defined by the application.

**Syntax C WORD ctmdSetChannelMapper( const char far** *\*lpszMapper* **)**

**Pascal ctmdSetChannelMapper(** *lpszMapper***: pointer ) :word**

**Basic ctmdSetChannelMapper%(** *lpszMapper&* **)**

**Parameters** *lpszMapper*

A far pointer to the 16 bytes channel mapper array, no null terminator is required.

**Remarks**  Each bytes of the 16 bytes channel mapper array corresponds to the MIDI channel starting from channel 1 and ending at channel 16. This values will be interpreted by the driver as the logical channel to the output. For example, if the value in array 0 is 5, then any MIDI channel 0 in the MIDI file will be mapped to channel 5 instead.

A value of -1 in the entry will suppress the respective channel.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdSetMapperType

**ctmdGetMapperType**

ASM Interface CTMIDI.DRV Function 6

**ctmdSetInputStatusAddx**

**Action** Sets the input status word address defined by the application.

**Syntax C WORD ctmdSetInputStatusAddx( const WORD far *\*****lpwStatus* **)**

**Pascal ctmdSetInputStatusAddx(** *lpwStatus* **: pointer) :word**

**Basic ctmdSetInputStatusAddx%(** *lpwStatus&* **)**

**Parameters** *lpwStatus*

Far pointer to the application defined status word.

**Remarks** Driver will modify the status word to reflect the actual MIDI input activity during MIDI recording. Application can monitor the word but should not modify its content. Modifying this word by the application will adversely affect the playing MIDI music as well as the MIDI input events.

Application should not set the same address for both playing and recording.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdSetOutputStatusAddx

ASM Interface CTMIDI.DRV Function 30

**ctmdSetMapperType**

**Action** Sets the mapper type.

**Syntax C WORD ctmdSetMapperType( WORD** *wMapper* **)**

**Pascal ctmdSetMapperType(** *wMapper* **: word ) :word**

**Basic ctmdSetMapperType%(** *wMapper%* **)**

**Parameters** *wMapper*

A word that specify the type of mapper used for playback. The followings are values that represent the type of mapper.

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **GENERAL\_MIDI\_MAPPER** | General Mapper Format |
| **EXTENDED\_MIDI\_MAPPER** | External Mapper Format |
| **BASIC\_MIDI\_MAPPER** | Basic Mapper Format |
| **USER\_DEFINED\_MAPPER** | User-defined Mapper Format |

User-defined Mapper Format will come into action only after the application program does a **ctmdSetChannelMapper** call to set to its user-defined mapper.

**Remarks** None

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdGetMidiEnvSettings

**ctmdGetMapperType**

ASM Interface CTMIDI.DRV Function 14

**ctmdSetMidiInputBuffer**

**Action** Sets the address and size of the buffer for MIDI input.

**Syntax C WORD ctmdSetMidiInputBuffer( DWORD far *\*****lpBuf***,**

**DWORD** *dwBufSize* )

**Pascal ctmdSetMidiInputBuffer (***lpBuf* **:pointer;** *dwBufSize* **:long) :word**

**Basic ctmdSetMidiInputBuffer%(** *lpBuf&***,** *dwBufSize&* **)**

**Parameters** *lpBuf*

Far pointer to the buffer for storing the incoming MIDI code.

dwBufSize

Size of the buffer in unit of double word.

**Remarks** A far pointer which points to the buffer, and its size are passed as the first and second parameters respectively. Application must use a buffer of size in multiples of four bytes, since the driver stores every incoming MIDI code in the form of four consecutive bytes with the first byte being the MIDI code and the following three bytes store the time stamp in milli-second.

The first four bytes is not used for storing MIDI code; instead the driver stores the count of the MIDI codes received so far. The second parameter specifies the size of the buffer in double word units (4 bytes) including the first double word (4 bytes) even though it is not used for storing MIDI code.

**Return Value** Zero if successful. Non-zero otherwise.

See Also None

ASM Interface CTMIDI.DRV Function 32

**ctmdSetMidiCallBackFunct**

**Action** Sets the callback function address.

**Syntax C WORD ctmdSetMidiCallBackFunct( WORD far *\*****lpFunct***,**

**DWORD** *dwToken* **)**

**Pascal ctmdSetMidiCallBackFunct(** *lpFunct* **:pointer;** *dwToken* **:long) :word**

**Basic ctmdSetMidiCallBackFunct%(** *lpFunct&***,** *dwToken&* **)**

**Parameters** *lpFunct*

Far pointer to the callback function.

dwToken

A double-word value to be returned to the application program for every MIDI code received. The value of this double word is insignificant to the driver, but used by the callback function.

**Remarks** This is provided by the driver as an optional feature. The driver will call the callback function when there is a in-bound MIDI data. It is extremely useful for those application that need real-time monitoring of the MIDI code input event to allow them to decide what action to take upon receiving the MIDI code. For example, to display the MIDI input event immediately after the MIDI code is received.

In the case of non-timing-critical operation, there is no need to take advantage of this feature. Instead, it can monitor the buffer to determine how many MIDI codes are being received and decide what to do with the MIDI code.

For an application that uses the buffer method, it must call the function **ctmdSetMidiInputBuffer** to make the buffer pointer as well as its size known to the driver .

The callback function is fixed to be a function that accepts two parameters from the driver. The first parameter is a far pointer to the double word containing the MIDI code and the Time-Stamp. The second parameter is a double-word user-defined data that returns to the application. To allow the callback mechanism to function properly, certain rules and criteria must be followed.

The call back function provided by the application program must use the PASCAL calling convention; (i.e. the callee is responsible to clear the stack parameters upon exits). For a detailed description on the callback function, refer to **Programmer's Guide**.

**Return Value** Zero if successful. Non-zero otherwise.

See Also None

ASM Interface CTMIDI.DRV Function 33

**ctmdSetMusicTempo**

**Action** Sets the tempo multiplier of the playing MIDI music.

**Syntax C WORD ctmdSetMusicTempo( int** *nMusicTempo* **)**

**Pascal ctmdSetMusicTempo(** *nMusicTempo* **:integer) :word**

**Basic ctmdSetMusicTempo%(** *nMusicTempo%* **)**

**Parameters** *nMusicTempo*

The desired tempo multiplier value to be set.

**Remarks**  The tempo multiplier value ranges from -20 to +20, (0 being the normal tempo). Every step of the tempo multiplier value change will increase/decrease the music speed by 10%. For example, tempo multiplier value of +10 will increase the music speed by 100% (i.e. twice the normal speed).

**Return Value** Zero if successful. Non-zero otherwise.

See Also None

ASM Interface CTMIDI.DRV Function 15

**ctmdSetMusicTranspose**

**Action** Sets the transpose of the playing MIDI music.

**Syntax C WORD ctmdSetMusicTranspose( int** *nMusicTranspose* **)**

**Pascal ctmdSetMusicTranspose(** *nMusicTranspose* **:integer) :word**

**Basic ctmdSetMusicTranspose%(** *nMusicTranspose%* **)**

**Parameters** *nMusicTranspose*

The desired transpose value to be set.

**Remarks**  The transpose value ranges from -12 to +12, 0 being the normal transpose. Every step of the transpose value change will increase/decrease the tune by one semitone.

**Return Value** Zero if successful. Non-zero otherwise.

See Also None

ASM Interface CTMIDI.DRV Function 16

**ctmdSetOutputStatusAddx**

**Action** Sets the output status word address defined by the application.

**Syntax** **C WORD ctmdSetOutputStatusAddx( const WORD far *\*****lpwStatus* **)**

**Pascal ctmdSetOutputStatusAddx(***lpwStatus* **: pointer) :word**

**Basic ctmdSetOutputStatusAddx%(** *lpwStatus&* **)**

**Parameters** *lpwStatus*

Far pointer to the application defined status word.

**Remarks** Driver will modify the status word to reflect the actual MIDI playback activity during MIDI playback. Application can monitor the word but should not modify its content. Modifying this word by the application will adversely affect the playing of MIDI music.

An application should not use the same address for both playing and recording.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdSetInputStatusAddx

ASM Interface CTMIDI.DRV Function 7

**ctmdSetTimeStampMode**

**Action** Sets the time stamp mode for MIDI recording.

**Syntax C WORD ctmdSetTimeStampMode( WORD** *wTimeStampMode* **)**

**Pascal ctmdSetTimeStampMode(** *wTimeStampMode* **:word ) :word**

**Basic ctmdSetTimeStampMode%(** *wTimeStampMode%* **)**

**Parameters** *wTimeStampMode*

A word to specified the mode to be used. It should be one of the followings:

|  |  |
| --- | --- |
| **Constant** | **Meaning** |
| **DIFFERENTIATE\_MODE** | Differentiate mode |
| **ELAPSED\_MODE** | Elapsed mode |

**Remarks** DIFFERENTIATE mode is the time differential between the current and previous MIDI code received.

ELAPSED mode is the accumulated running time from the start of the first MIDI code received.

**Return Value** Zero if successful. Non-zero otherwise.

See Also None

ASM Interface CTMIDI.DRV Function 31

**ctmdStartMidiInput**

**Action** Starts the MIDI input operation.

**Syntax C WORD ctmdStartMidiInput( void )**

**Pascal ctmdStartMidiInput :word**

**Basic ctmdStartMidiInput%( )**

**Parameters** None

**Remarks**  None

Return Value Zero if successful. Non-zero otherwise.

See Also ctmdStopMidiInput

ASM Interface CTMIDI.DRV Function 34

**ctmdStopMidiInput**

**Action** Stops the MIDI input operation.

**Syntax C WORD ctmdStopMidiInput( void )**

**Pascal ctmdStopMidiInput :word**

**Basic ctmdStopMidiInput%( )**

**Parameters** None

**Remarks**  None

Return Value Zero if successful. Non-zero otherwise.

See Also ctmdStartMidiInput

ASM Interface CTMIDI.DRV Function 35

**ctmdStopMidiMusic**

**Action** Stops the playing of MIDI music.

**Syntax C WORD ctmdStopMidiMusic( void )**

**Pascal ctmdStopMidiMusic :word**

**Basic ctmdStopMidiMusic%( )**

**Parameters** None

**Remarks** None

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdPrepareMidiStart

**ctmdPlayMidiMusic**

**ctmdPauseMidiMusic**

**ctmdResumeMidiMusic**

ASM Interface CTMIDI.DRV Function 10

**ctmdTerminate**

**Action** Terminates the driver.

**Syntax C WORD ctmdTerminate( void )**

**Pascal ctmdTerminate :word**

**Basic ctmdTerminate%( )**

**Parameters** None

**Remarks**  This function releases those interrupt(s) hooked by the driver. It must be called before the program exits.

**Return Value** Zero if successful. Non-zero otherwise.

See Also ctmdInit

ASM Interface CTMIDI.DRV Function 4

Assembly Interface

The entries in this section tell you what values to load into each register, in order to invoke the driver services. The details are in the High-Level Language interface descriptions.

0 Get Driver Version

**Action** Gets the version number of the MIDI driver.

Entry BX = 0

**Exit** **AX** = version number

**Details In** CTMIDI.DRV HLL Interface **ctmdGetDrvVer**

1 Get MIDI Environment Settings

**Action** Passes the MIDI environment string to the driver to interpret which synthesizer and channel mapper to use.

Entry BX = 1

DX:AX = lpszMidi

**Exit** AX = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdGetMidiEnvSettings**

2 Get Environment Settings

**Action** Passes the BLASTER environment string to the driver to interpret which hardware settings to use.

Entry BX = 2

**DX:AX** = *lpszBlaster*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdGetEnvSettings**

3 Initialize Driver

**Action** Initializes the driver.

Entry BX = 3

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdInit**

4 Terminate Driver

**Action** Terminates the driver.

Entry BX = 4

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdTerminate**

5 Reset MIDI Driver

**Action** Resets MIDI driver.

Entry BX = 5

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdResetMidiDriver**

6 Set The Channel Mapper

**Action** Sets the channel mapping defined by the application.

Entry BX = 6

**DX:AX** = *lpszMapper*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetChannelMapper**

7 Set Output Status Word Address

**Action** Sets the output status word address defined by the application.

Entry BX = 7

**DX:AX** = *lpwStatus*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetOutputStatusAddx**

8 Prepare MIDI Start

**Action** Pre-processes the MIDI file to be played.

Entry BX = 8

**DX:AX** = *lpszMidiBuffer*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdPrepareMidiStart**

9 Play MIDI Music

**Action** Starts playing the MIDI file.

Entry BX = 9

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdPlayMidiMusic**

10 Stop MIDI Music

**Action** Stops playing the MIDI file.

Entry BX = 10

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdStopMidiMusic**

11 Pause MIDI Music

**Action** Pauses the playing MIDI music.

Entry BX = 11

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdPauseMidiMusic**

12 Resume MIDI Music

**Action** Resumes the paused MIDI music.

Entry BX = 12

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdResumeMidiMusic**

13 Get Mapper Type

**Action** Queries the current mapper type.

**Entry** **BX** = 13

**Exit** **AX** = mapper type

**Details In** CTMIDI.DRV HLL Interface **ctmdGetMapperType**

14 Set Mapper Type

**Action** Sets the mapper type.

Entry BX = 14

**AX** = *wMapper*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetMapperType**

15 Set Music Tempo Multiplier

**Action** Sets the tempo multiplier of the playing MIDI music.

Entry BX = 15

**AX** = *nMusicTempo*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetMusicTempo**

16 Set Music Transpose

**Action** Sets the transpose of the playing MIDI music.

Entry BX = 16

**AX** = *nMusicTranspose*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetMusicTranspose**

17 Send Short Message

**Action** Sends a MIDI event to the external MIDI synthesizer via the MIDI port.

Entry BX = 17

**AX** = *wMidiStatus*

**DX** = *wMidiData1*

**CX** = *wMidiData2*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSendShortMessage**

18 Send Long Message

**Action** Sends a buffer of MIDI data to the external MIDI synthesizer via the MIDI port.

Entry BX = 18

**DX:AX** = *lpMsg*

**CX** = *wMsgLen*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSendLongMessage**

30 Set Input Status Word Address

**Action** Sets the input status word address defined by the application.

Entry BX = 30

**DX:AX** = *lpwStatus*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetInputStatusAddx**

31 Set Time Stamp Mode

**Action** Sets the time stamp mode for MIDI recording.

Entry BX = 31

**AX** = *wTimeStampMode*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetTimeStampMode**

32 Set MIDI Input Buffer

**Action** Sets the address and size of the buffer for MIDI input.

Entry BX = 32

**DX:AX** = *lpBuf*

**DI:CX** = *dwBufSize*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetMidiInputBuffer**

33 Set MIDI Callback Function

**Action** Sets the callback function address.

Entry BX = 33

**DX:AX** = *lpFunct*

**DI:CX** = *dwToken*

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdSetMidiCallBackFunct**

34 Start MIDI Input

**Action** Starts the MIDI input operation.

Entry BX = 34

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdStartMidiInput**

35 Stop MIDI Input

**Action** Stops the MIDI input operation.

Entry BX = 35

**Exit** **AX** = error code

**Details In** CTMIDI.DRV HLL Interface **ctmdStopMidiInput**

Chapter 5

CD-ROM Audio

This chapter documents the high-level language library functions that perform audio operations on the Creative CD-ROM drive.

High-Level Language Interface

This section describes the high-level language interface of the CD-ROM audio functions in alphabetical order.

Function Prefix

High-level function names begin with the following prefix:

|  |  |
| --- | --- |
| **Prefix** | **Description** |
| sbcd | CD-ROM audio functions |

Include Files

The followings are the required include files for the CD-ROM audio functions:

|  |  |  |
| --- | --- | --- |
| **C Language** | **Turbo Pascal** | **Microsoft Basic** |
| SBKCD.H | SBKCD.INC | SBKCD.BI |

Functions by Category

The CD-ROM audio functions may be divided into the following categories:

|  |  |
| --- | --- |
| **Category** | **Function** |
|  |  |
| Initialization | **sbcdInit** |
|  |  |
|  |  |
| Audio playback control | **sbcdPlay** |
|  | **sbcdStop** |
|  | **sbcdFastForward** |
|  | **sbcdRewind** |
|  | **sbcdNextTrack** |
|  | **sbcdPrevTrack** |
|  | **sbcdPause** |
|  | **sbcdContinue** |
|  |  |
|  |  |
| CD information | **sbcdGetVolume** |
|  | **sbcdGetDiscInfo** |
|  | **sbcdReadTOC** |
|  |  |
|  |  |
| CD-ROM drive-related operations | **sbcdSelectDrive** |
|  | **sbcdEject** |
|  | **sbcdCloseTray** |
|  | **sbcdLockDoor** |
|  | **sbcdGetAudioStatus** |
|  | **sbcdGetDeviceStatus** |
|  | **sbcdGetLocInfo** |
|  | **sbcdMediaChanged** |
|  |  |

Values Passed To / Returned By Functions

Values passed to or returned by some of the audio functions are in Binary-Coded Decimal (BCD) format. These will be stated explicitly.

Error Codes

All the CD-ROM audio functions that performed successfully return zero. A non-zero error code is returned otherwise. The followings are lists of the error codes returned by the CD-ROM audio functions:

|  |  |
| --- | --- |
| **Error Code (in Hex)** | **Description** |
| 10 | Write-protect violation |
| 11 | Unknown unit |
| 12 | Drive not ready |
| 13 | Unknown command |
| 14 | CRC error |
| 15 | Bad drive request structure length |
| 16 | Seek error |
| 17 | Unknown media |
| 18 | Sector not found |
| 19 | Reserved |
| 1A | Write fault |
| 1B | Read fault |
| 1C | General failure |
| 1D | Reserved |
| 1E | Reserved |
| 1F | Invalid disc change |
| 20 | Invalid track number |
| 21 | Invalid parameter input |
| 22 | Not a CD-ROM drive |
| 23 | Drive not in play mode |
| 24 | Drive not in pause mode |

**sbcdCloseTray**

**Action** Closes the CD-ROM drive tray.

**Syntax C int sbcdCloseTray( void )**

**Pascal sbcdCloseTray :integer**

**Basic sbcdCloseTray%( )**

**Parameters** None

**Remarks** No effect if the CD-ROM drive does not support software tray operations.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdEject, sbcdLockDoor

**sbcdContinue**

**Action** Resumes a previously paused audio playback.

**Syntax** **C** **int sbcdContinue( void )**

**Pascal sbcdContinue :integer**

**Basic sbcdContinue%( )**

**Parameters** None

**Remarks** No effect if the CD-ROM drive is not in pause mode.

**Return Value** Zero if successful. Non-zero if the drive is not in pause mode.

See Also sbcdPause

**sbcdEject**

**Action** Opens the drive tray.

**Syntax** **C** **int sbcdEject( void )**

**Pascal sbcdEject :integer**

**Basic sbcdEject%( )**

**Parameters** None

**Remarks** No effect on CD-ROM drives not supporting software tray operations.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdCloseTray, sbcdLockDoor

**sbcdFastForward**

**Action** Forwards the disc in the CD-ROM drive for a specified time.

**Syntax** **C** **int sbcdFastForward( WORD** *wSec* **)**

**Pascal sbcdFastForward(** *wSec* **:word ) :integer**

**Basic sbcdFastForward%( BYVAL** *wSec%* **)**

**Parameters** *wSec*

Amount of time (in of seconds) to forward the CD.

**Remarks** No effect if the drive is not in play mode.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdRewind, sbcdNextTrack, sbcdPrevTrack

**sbcdGetAudioStatus**

**Action** Returns the CD-ROM drive audio status. This status indicates whether the CD-ROM drive is in pause mode.

**Syntax** **C** **int sbcdGetAudioStatus( int far \****lpStatus* **)**

**Pascal sbcdGetAudioStatus( var** *lpStatus* **:integer )** **:integer**

**Basic sbcdGetAudioStatus%( SEG** *lpStatus%* **)**

**Parameters** *lpStatus*

A far pointer to an integer variable for audio status returned:

|  |  |
| --- | --- |
| 0 | - drive is not in pause mode |
| 1 | - drive is in pause mode |

**Remarks** None

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdPause, sbcdContinue

**sbcdGetDeviceStatus**

**Action** Returns the status of the CD-ROM drive.

**Syntax** **C** **int sbcdGetDeviceStatus( DWORD far \****lpdwStatus* **)**

**Pascal sbcdGetDeviceStatus( var** *lpdwStatus* **:longint )** **:integer**

**Basic sbcdGetDeviceStatus%( SEG** *lpdwStatus&* **)**

**Parameters** *lpdwStatus*

A far pointer to a long integer variable for device status returned:

(Bit 0 is the least significant bit)

|  |  |  |
| --- | --- | --- |
| **Bit** | **Value** | **Description** |
| Bit 0 | 0 | Door is closed |
|  | 1 | Door is open |
| Bit 1 | 0 | Door is locked |
|  | 1 | Door is unlocked |
| Bit 2-10 |  | Reserved |
| Bit 11 | 0 | Disc is in drive |
|  | 1 | Disc is not in drive |
| Bit 12-31 |  | Reserved |

**Remarks** None

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also None

**sbcdGetDiscInfo**

**Action** Returns the highest, lowest track number, and the Red Book address of the lead-out track on a CD.

**Syntax** **C** **int sbcdGetDiscInfo( DISK\_INFO far \****lpBuffer* **)**

**Pascal sbcdGetDiscInfo( var** *lpBuffer* **:DISK\_INFO ) :integer**

**Basic sbcdGetDiscInfo%( SEG** *lpBuffer* **AS DISKxINFO )**

**Parameters** *lpBuffer*

A far pointer to a buffer of data type DISK\_INFO. The data structure of DISK\_INFO is:

BYTE bLoTNo; // lowest track number

BYTE bHiTNo; // highest track number

DWORD dwLeadOut; // lead-out track address

The bLoTNo and bHiTNo bytes give the lowest and highest track number (in binary) of a disc respectively. The dwLeadOut field returns the Red Book address of the lead-out track.

**Remarks** None

**Return Value** Zero if successful. Otherwise, a non-zero is returned.

See Also sbcdReadTOC

**sbcdGetLocInfo**

**Action** Returns the current location of the CD. Both track relative time and absolute time are returned.

**Syntax** **C** **int sbcdGetLocInfo( QCHAN\_INFO far \****lpBuffer* **)**

**Pascal sbcdGetLocInfo( var** *lpBuffer* **:QCHANINFO\_INFO ) :integer**

**Basic sbcdGetLocInfo%( SEG** *lpBuffer* **AS QCHANxINFO )**

**Parameters** *lpBuffer*

A far pointer to a buffer of data type QCHAN\_INFO. The data structure of QCHAN\_INFO is:

BYTE bTNo; // current track number

BYTE bReserved

BYTE bMin; // minute }

BYTE bSec; // second } running time within

BYTE bFrame; // frame } a track

BYTE bReserved;

BYTE bPMin; // minute }

BYTE bPSec; // second } running time on the

BYTE bPFrame; // frame } disk

The field bTNo gives the current track number in BCD format. The values in bMin, bSec, bFrame, bPMin, bPSec and bPFrame are in binary. bMin-bSec-bFrame gives the track-relative time while bPMin-bPSec-bPFrame gives the absolute time on a disc.

**Remarks** Valid information is returned regardless of whether audio is being played. Calling the function does not affect the state of the CD-ROM drive.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also None

**sbcdGetVolume**

**Action** Returns the volume size, in sectors, of a disc.

**Syntax** **C** **int sbcdGetVolume( DWORD far \****lpdwVolumeSize* **)**

**Pascal sbcdGetVolume( var** *lpdwVolumeSize* **:longint ) :integer**

**Basic sbcdGetVolume%( SEG** *lpdwVolumeSize*&**)**

**Parameters** *lpdwVolumeSize*

Contains the returned volume size, in sectors.

**Remarks** Volume size in sectors can be converted to the Red Book format, if needed. Refer to **Terminology** for the conversion equation.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also None

**sbcdInit**

**Action** Initializes the CD-ROM drive.

**Syntax** **C** **int sbcdInit( int far \****lpNumDrive* **)**

**Pascal sbcdInit( var** *lpNumDrive* **:integer ) :integer**

**Basic sbcdInit%( SEG** *lpNumDrive%* **)**

**Parameters** *lpNumDrive*

Stores the number of CD-ROM drives returned.

**Remarks** During initialization, the existence of the CD-ROM driver is checked and the total number of connected CD-ROM drives is detected. The function also sets the first CD-ROM drive to be the active drive.

When application starts, it should make this function call to determine if the CD-ROM driver and MSCDEX have been installed.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also None

**sbcdLockDoor**

**Action** Locks or unlocks the CD-ROM drive tray.

**Syntax** **C** **int sbcdLockDoor( BYTE** *bFunction* **)**

**Pascal sbcdLockDoor(** *bFunction* **:byte ) :integer**

**Basic sbcdLockDoor%( BYVAL** *bFunction%* **)**

**Parameters** *bFunction*

0 for unlock; 1 for lock.

**Remarks** A locked tray cannot be opened by pressing the "Eject" button or by calling the **sbcdEject** function.

A tray can be locked in the open or close position. If the **sbcdLockDoor(1)** function is invoked when the tray is open, the tray will be locked when it is next closed.

This function has no effect on drives that do not support tray locking feature.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdEject, sbcdCloseTray

**sbcdMediaChanged**

**Action** Detects if the disc in the drive has changed.

**Syntax** **C** **int sbcdMediaChanged( int far \****lpChanged* **)**

**Pascal sbcdMediaChanged( var** *lpChanged* **:integer ) :integer**

**Basic sbcdMediaChanged%( SEG** *lpChanged%* **)**

**Parameters** *lpChanged*

Contains the returned value of the media status:

|  |  |
| --- | --- |
| **Value** | **Description** |
| 1 | Disc has not been changed |
| 0 | Does not know if disc has been changed |
| -1 (0FFFF) | Disc has been changed |

**Remarks** An application can use this function to ensure that the TOC that is previously read is still valid for the current disc.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdReadTOC

**sbcdNextTrack**

**Action** Stops playing the current track and proceeds to play the next track.

**Syntax** **C** **int sbcdNextTrack( void )**

**Pascal sbcdNextTrack :integer**

**Basic sbcdNextTrack%( )**

**Parameters** None

**Remarks** If this function is called when the current track is the last CD track, it starts playing the first track. To avoid this wrapping around, applications should note the currently track that is playing and not call this function when the last track is playing.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdPrevTrack, sbcdFastForward, sbcdRewind

**sbcdPause**

**Action** Pauses audio playback.

**Syntax** **C** **int sbcdPause( void )**

**Pascal sbcdPause :integer**

**Basic sbcdPause%( )**

**Parameters** None

**Remarks** This function has no effect if the CD-ROM drive is not in play mode or is already in pause mode.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdContinue

**sbcdPlay**

**Action** Performs audio playback.

**Syntax** **C** **int sbcdPlay( BYTE** *bTrackNo***, WORD** *wOffset***, WORD** *wDuration* **)**

**Pascal sbcdPlay(** *bTrackNo* **:byte;** *wOffset***,** *wDuration* **:word ) :integer**

**Basic sbcdPlay%( BYVAL** *bTrackNo%***, BYVAL** *wOffset%***,**

**BYVAL** *wDuration%* **)**

**Parameters** *bTrackNo*

The track to play. An application should ensure that the value of *bTrackNo* is within the lowest and highest track number on the disc. The lowest and highest track numbers can be obtained by calling the **sbcdGetDiscInfo** function.

wOffset

The offset (in sectors) from the beginning of the track to start audio playback from. To start playing from the beginning of a track, specify a value of 0.

wDuration

The length (in seconds) for audio playback. To play till the end of a disc, specify a value of 0xFFFF.

**Remarks** None

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdGetDiscInfo, sbcdStop

**sbcdPrevTrack**

**Action** Stops playing the current track and proceeds to play the preceding track.

**Syntax** **C** **int sbcdPrevTrack( void )**

**Pascal sbcdPrevTrack :integer**

**Basic sbcdPrevTrack%( )**

**Parameters** None

**Remarks** None

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdNextTrack, sbcdFastForward, sbcdRewind

**sbcdReadTOC**

**Action** Returns the TOC of the current disc.

**Syntax** **C** **int sbcdReadTOC( DWORD far \****lpTOCBuffer* **)**

**Pascal sbcdReadTOC(** *lpTOCBuffer***:pointer** **) :integer**

**Basic sbcdReadTOC%( BYVAL** *lpTOCBuffer&* **)**

**Parameters** *lpTOCBuffer*

An array of long integers containing the Red Book addresses of all sound tracks, as well as the lead-out track, on the disc. (The first element of the array contains the first audio track address, the second element contains the second track address, etc.). The address of the lead-out track is stored in the array element following that containing the address of the last audio track.

**Remarks** Applications calling this function should ensure that the size of the buffer is sufficient for storing all track addresses. A typical array size of 100 elements is recommended. Alternatively, the number of tracks on a disc can be determined by calling the **sbcdGetDiscInfo** function. The size of the required buffer is thus, (highest track number + 1(for lead-out track) ) elements.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdGetDiscInfo

**sbcdRewind**

**Action** Rewinds the CD-ROM drive backward for a specified time.

**Syntax C int sbcdRewind( WORD** *wSec* **)**

**Pascal sbcdRewind(** *wSec* **:word ) :integer**

**Basic sbcdRewind%( BYVAL** *wSec%* **)**

**Parameters** *wSec*

The amount of time, in seconds, to rewind.

**Remarks** This function has no effect if the drive is not in play mode.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdFastForward, sbcdNextTrack, sbcdPrevTrack

**sbcdSelectDrive**

**Action** Selects a CD-ROM drive for subsequent operations.

**Syntax C int sbcdSelectDrive( BYTE** *bDriveNum* **)**

**Pascal sbcdSelectDrive(** *bDriveNum* **:byte ) :integer**

**Basic sbcdSelectDrive%( BYVAL** *bDriveNum%* **)**

**Parameters** *bDriveNum*

The desired CD-ROM drive number (0 for drive A, 1 for drive B, 2 for drive C, etc.).

**Remarks** If the function is not called or is called with an invalid drive number, all CD operations will be directed to the first CD-ROM drive. You may determine the number of CD-ROM drive connected by calling the **sbcdInit** function.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdInit

**sbcdStop**

**Action** Stops audio playback.

**Syntax C int sbcdStop( void )**

**Pascal sbcdStop :integer**

**Basic sbcdStop%( )**

**Parameters** None

**Remarks** None.

**Return Value** Zero if successful. Otherwise, an error code is returned.

See Also sbcdPlay

Appendix A

File Format

This appendix provides information about the Creative Voice File (.VOC) format and the Creative ADPCM wave type format registered with Microsoft.

The Creative Voice File allows you to:

embed ASCIIZ text and/or marker.

include information on compression techniques.

loop on a portion of the .VOC file.

use digitized sound data with multiple sampling rates within a file.

The Creative ADPCM wave type is used in the Multimedia Wave File to support the Creative ADPCM compression technique.

Creative Voice File (VOC) Format

The Creative Voice File is organized in two main blocks, the Header Block and Data Block.

The Header Block contains identifier, version number and pointer to the start of the Data Block. The Data Block is divided into sub-blocks of various types.

The CT-VOICE driver only processes the Data Block. It is important that you pass the address of the Data Block and not the entire .VOC File when calling this driver to perform digitized sound output.

Header Block

|  |  |
| --- | --- |
| **Offset (Hex)** | **Description** |
| 00H - 13H | File type description. |
|  | The following message is stored here:  "Creative Voice File", 1AH |
| 14H - 15H | Offset of the Data Block from the start of .VOC file. |
|  | This word points to the Data Block. It helps the application programs to locate the Data Block in case the size of Header Block is changed.  For this version, the value here is 1A Hex. |
| 16H - 17H | .VOC file format version number. |
|  | This version number allows your program to identify different organization formats of .VOC file in case of future enhancement.  The low and high byte are the minor and major version number respectively. Current version is 1.20 (0114H). |

|  |  |
| --- | --- |
| **Offset (Hex)** | **Description** |
| 18H - 19H | .VOC file identification code. |
|  | This code allows your program to check that this file is a .VOC file.  Its content is the complement of the file format version number, plus 1234 hex. For version of 1.20, it is complement(0114H) + 1234H = 111FH. |

Data Block

The Data Block is sub-divided into multiple sub-blocks of data.

The first byte of each sub-block is called the Block Type. It indicates the type of data contained in the sub-block.

The next three bytes is the 24-bit (3-byte) Block Length. It is the number of bytes in the sub-block excluding the Block Type and Block Length fields. The first byte is a lowest byte and the third byte is the highest byte of the length field respectively. All sub-blocks have the Block Type field followed immediately by the block length field except the Terminator sub-block.

Your program need not interpret all the Block Types. If unknown Block Type is encountered, it should ignored and advance to the next sub-block by using the Block Length.

The high-level digitized sound drivers handle these data blocks automatically for you. Therefore, you should use these drivers to perform digitized sound I/O operations.

Block Type 0

This is a 1-byte sub-block which terminates the entire Data Block. The Block Type identifier is 0. It indicates that there are no other sub-blocks after it. The high-level digitized sound drivers terminate digitized sound output when this Block Type is encountered.

This Block Type should be the last block of the .VOC file.

Block Type 1

This is a digitized sound data block. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 1 |
| BYTE | nBlockLen[3]; | // 3-byte block length |
| BYTE | bTimeConstant; |  |
| BYTE | bPackMethod; | // Packing Method |

The header is followed immediately by the digitized sound data.

Here is a discussion of various fields:

bBlockID

The Block Type identifier is 1.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

The value here will be the digitized sound data length plus 2.

bTimeConstant

This is a 1-byte field which indicates the **Time Constant** of the digitized sound data of this block. The Time Constant is defined as follows:

Time Constant = 65536 - (256 000 000/( *channels* \* *sampling rate*))

The *channels* parameter is 1 for mono and 2 for stereo.

Only the high byte of the result is stored here. For instance, for a 10000Hz mono digitized sound, the Time Constant is set to 9C hex using the following calculation:

|  |  |
| --- | --- |
| Time Constant | = 65536 - (256 000 000 / 10 000)  = 39936 (09C00H) |

bPackMethod

This is an 1 byte field which indicates the packing method used by the digitized sound data of this block. It is defined as:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 0 | 8-bit PCM |
| 1 | Creative 8-bit to 4-bit ADPCM |
| 2 | Creative 8-bit to 3-bit ADPCM |
| 3 | Creative 8-bit to 2-bit ADPCM |

Some points to note for Block Type 1:

1. If this block is preceded by Block Type 8 (discussed later), the digitized sound attributes on Block Type 8 should be used. The digitized sound attributes in this block should be ignored.

2. If this block is alone, the digitized sound channels should be defaulted to mono.

Block Type 2

This is a digitized sound continuation block. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 2 |
| BYTE | nBlockLen[3]; | // 3-byte block length |

The header is followed immediately by the digitized sound data.

Here is a discussion of various fields:

bBlockID

The Block Type identifier is 2.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

This block type will only be used when the digitized sound data size exceeds the 3-byte block length (16 megabytes).

Block Type 3

This block specifies the pause period for the digitized sound before next block of digitized sound data is transferred. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 3 |
| BYTE | nBlockLen[3]; | // 3-byte block length |
| WORD | wPausePeriod; |  |
| BYTE | bTimeConstant; |  |

Here is a discussion of various fields:

bBlockID

The Block Type identifier is 3.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

The value is 3.

wPausePeriod

This is a 2-byte field which specifies the pause period in units of sampling cycles. Total pause cycle is **wPausePeriod** plus 1.

bTimeConstant

This is a 1-byte field which indicates the **Time Constant** of the pause period. The Time Constant calculation is the same as described in Block Type 1.

Block Type 4

This is a special block that specifies a **Marker** in the digitized sound data. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 4 |
| BYTE | nBlockLen[3]; | // 3-byte block length |
| WORD | wMarker; | // marker value |

Here is a discussion of various fields:

bBlockID

The Block Type identifier is 4.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

The value is 2.

wMarker

This is a 2-byte field which specifies the marker value. The marker value can be any value between 1 to 0FFFE hex inclusive. The 0 and 0FFFF hex values are reserved by the digitized sound drivers.

During digitized sound output, the CT-VOICE and CTVDSK drivers update the digitized sound status word with this value when the marker is encountered. You program can check for the desired marker value to perform synchronization with the digitized sound output process.

Block Type 5

This block enables you to embed a null-terminated ASCII string in the .VOC file. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 5 |
| BYTE | nBlockLen[3]; | // 3-byte block length |
| BYTE | szString[]; | // Null-terminated string |

Here is a discussion of various fields:

bBlockID

The Block Type identifier is 5.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

The value is the length of the null-terminated ASCII string (null inclusive).

szString

This is variable length field which specifies a null-terminated ASCII string. The length of this field is the string length (null inclusive).

This field is for a program that requires ASCII information on the .VOC file such as name, type or remarks. You may choose to ignore this Block Type during the digitized sound block manipulation.

Block Type 6

This block indicates the beginning of a repeat loop. The data block between this block and the next End Repeat Block (Block Type 7) will be repeated. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 6 |
| BYTE | nBlockLen[3]; | // 3-byte block length |
| WORD | wRepeatTimes |  |

Here is a discussion of various fields:

bBlockID

The Block Type identifier is 6.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

The value is 2.

wRepeatTimes

This is a 2-byte field which specifies the number of times to repeat. It can be any value between 1 to 0FFFE hex inclusive. If this value is set to 0FFFF hex, an endless loop occurs.

Block Type 7

This block indicates the end of a repeat loop. It works in conjunction with Block Type 6. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 7 |
| BYTE | nBlockLen[3]; | // 3-byte block length |

Here is a discussion of various fields:

bBlockID

The Block Type identifier is 7.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

The value is 0.

Block Type 8

This is a special block that carries only the digitized sound attributes. It MUST precede Block Type 1. Usually, this block precedes the stereo or high speed digitized sound data. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 8 |
| BYTE | nBlockLen[3]; | // 3-byte block length |
| WORD | wTimeConstant; | // 2-byte Time Constant |
| BYTE | bPackMethod; | // Packing Method |
| BYTE | bVoiceMode; | // mono or stereo |

The header is followed immediately by Block Type 1.

Here is a discussion of various fields:

bBlockID

The Block Type identifier is 8.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

The value is 4.

wTimeConstant

This is a 2-byte field which indicates the **Time Constant** of the digitized sound data in the Block Type 1. The calculation of the Time Constant is the same as described on Block Type 1, but the whole word of the result is stored here.

For a 44 100Hz sampling rate mono digitized sound, the Time Constant is calculated as follows:

|  |  |
| --- | --- |
| Time Constant | = 65536 - (256 000 000 / 44 100)  = 59732 (0E95H) |

For a 22 050Hz sampling rate stereo digitized sound, the Time Constant is calculated as follows:

|  |  |
| --- | --- |
| Time Constant | = 65536 - (256 000 000 / (2 \* 22 050))  = 59732 (0E95H) |

bPackMethod

This is a 1-byte field which indicates the packing method used by the digitized sound data of this block. The meaning of the field is the same as the **bPackMethod** field in Block Type 1:

bVoiceMode

This is a 1-byte field which indicates mono or stereo digitized sound (0 for mono and 1 for stereo).

After this block, the digitized sound attributes carried by the following Block Type 1 is ignored.

Block Type 9

This is a digitized sound data block that supersedes Block Types 1 and 8. The Block Header is organized as follows:

|  |  |  |
| --- | --- | --- |
| BYTE | bBlockID; | // == 9 |
| BYTE | nBlockLen[3]; | // 3-byte block length |
| DWORD | dwSamplesPerSec; |  |
| BYTE | bBitsPerSample; |  |
| BYTE | bChannels; |  |
| WORD | wFormat; |  |
| BYTE | reserved[4]; | // pad with zero |

The header is followed immediately by the digitized sound data.

The four reserved bytes at the end are there for two reasons:

1. Pad the header up to a length of 16 bytes (a convenient size for manipulation).

2. Provide for future expansion.

Here is a brief discussion of the various fields:

bBlockID

The Block Type identifier is 9.

nBlockLen

Length of the block (in bytes), excluding the **bBlockID** and **nBlockLen** fields.

The value will be the digitized sound data length plus 12.

dwSamplesPerSec

This is the **actual**sampling frequency, not a Time Constant. There is no need to double the value when dealing with stereo I/O (unlike in Block Type 8).

bBitsPerSample

Actual number of bits per sample after compression (if any).

bChannels

This is 1 for mono or 2 for stereo.

wFormat

The currently supported formats are:

|  |  |
| --- | --- |
| **Value** | **Meaning** |
| 0x0000 | 8-bit unsigned PCM |
| 0x0001 | Creative 8-bit to 4-bit ADPCM |
| 0x0002 | Creative 8-bit to 3-bit ADPCM |
| 0x0003 | Creative 8-bit to 2-bit ADPCM |
| 0x0004 | 16-bit signed PCM |
| 0x0006 | CCITT a-Law |
| 0x0007 | CCITT -Law |
| 0x0200 | Creative 16-bit to 4-bit ADPCM |

Some other points to note:

1. This is a new Block Type introduced on .VOC file with version number 1.20 and above.

2. It is intended that this Block Type supersedes Block Types 1 and 8. That is, the new drivers will produce BlockType 9 blocks on recording.

Creative ADPCM Wave Type Format

**WAVE\_FORMAT\_CREATIVE\_ADPCM**, the name for a new .WAV format tag, 0x0200 has been registered with Microsoft. The wave format header is *typedef*’ed with

typedef struct creative\_adpcmwaveformat\_tag

{

WAVEFORMATEX ewf;

WORD wRevision;

} CREATIVEADPCMWAVEFORMAT;

An exploded view of the structure is laid out below:

|  |  |
| --- | --- |
| **Structure Members** | **Description** |
| **wFormatTag** | WAVE\_FORMAT\_CREATIVE\_ADPCM |
| **nChannels** | Number of channels.  1 for mono, 2 for stereo. |
| **nSamplesPerSec** | Sampling frequency of the data.  Should be restricted to 8000, 11025, 22050 and 44100 Hz. |
| **nAvgBytesPerSec** | Average data rate. |
| **nBlockAlign** | Block alignment.  1 for both mono and stereo data. |
| **wBitsPerSample** | Number of bits per sample.  The value is 4. |
| **cbExtraSize** | Number of bytes of extra information in the extended WAVE ‘fmt ’ header.  The value is 2. |
| **wRevision** | Revision of algorithm.  The value is zero for the current definition. |

This information should be in the latest release of the “Microsoft Multimedia Standards Update”. Where there are differences, this document is the final arbiter.

Index

?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

?, 1-2

A\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

AGC, *see* Automatic Gain Control

Automatic Gain Control

AUXAGCCAPS, 3-58

AUXDM\_AGC\_Get, 3-13

AUXDM\_AGC\_QueryCaps, 3-14

AUXDM\_AGC\_Set, 3-15

constant gain, 3-15

ctadGetAGC, 2-6, 2-35

ctadSetAGC, 2-19, 2-35

B\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bass, *see* Tone

BLASTER environment string, *see* Configuration

Block Align, 3-75, 3-79

Break Loop

break mode, 1-4

ct?vBreakLoop, 1-4, 1-49, 1-54

Buffer, *see* Disk Buffer, Transfer Buffer, Midi Buffer

Buffer Queue

SOUNDBUFFER, 3-70

SOUNDBUFQ, 3-72

SxDM\_BUFFERQUEUE\_Add, 3-37

SxDM\_BUFFERQUEUE\_Query, 3-38

C\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Callback, 3-4, 3-42, 4-21, 4-37

CD Information

DISK\_INFO, 5-11

error codes, 5-4

QCHAN\_INFO, 5-12

sbcdGetDiscInfo, 5-11

sbcdGetVolume, 5-13

sbcdReadTOC, 5-21

track number, 5-11

CD-ROM Drive Operations

sbcdCloseTray, 5-5

sbcdEject, 5-7

sbcdGetAudioStatus, 5-9

sbcdGetDeviceStatus, 5-10

sbcdGetLocInfo, 5-12

sbcdLockDoor, 5-15

sbcdMediaChanged, 5-16

sbcdSelectDrive, 5-23

Close Device

AUXDM\_CLOSE, 3-16

CSPDM\_CLOSE, 3-50

SxDM\_CLOSE, 3-39

Code Download

CSPCODEDOWNLOAD, 3-84

Configuration

AUXDM\_CONFIGURATION\_Query, 3‑17

CSPDM\_CONFIGURATION\_Query, 3-51

ct??GetEnvSettings, 1-8

ctadGetEnvSettings, 2-8, 2-36

DEVCONFIG, 3-57

SxDM\_CONFIGURATION\_Query, 3-40

Constants

CTMIDI manifest constants, 4-4

Continue

ct??Continue, 1-5, 1-48, 1-54, 1-60, 1-66

ctmdResumeMidiMusic, 4-14, 4-34

sbcdContinue, 5-6

Conventions, *see* Document Typographic Conventions

CTMMSYS.SYS

calling convention, Pascal, 3-2

data structure

auxiliary device, 3-8, 3-58 to 3-67

common, 3-8, 3-57

signal processing device, 3-10, 3-82 to 3-85

sound device, 3-9, 3-68 to 3-81

entry-point, 3-2

error messages, 3-3, 3-11

install, 3-1

message listing

auxiliary device, 3-5

signal processing device, 3-7

sound device, 3-6

messages

auxiliary device, 3-13 to 3-36

sound device, 3-50 to 3-56

D\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Device Types

auxiliary device, 3-2

AUXCAPS, 3-59

AUXDM\_CLOSE, 3-16

AUXDM\_OPEN, 3-28

AUXDM\_QUERY\_Capabilities, 3-29

AUXDM\_QUERY\_NumDevs, 3-30

AUXOPEN, 3-64

signal processing device, 3-2

CSPCAPS, 3-82

CSPCODEDOWNLOAD, 3-84

CSPDM\_CLOSE, 3-50

CSPDM\_CODE\_Download, 3-52

CSPDM\_OPEN, 3-53

CSPDM\_QUERY\_Capabilities, 3-54

CSPDM\_QUERY\_NumDevs, 3-55

CSPDM\_STATE\_Set, 3-56

CSPOPEN, 3-85

sound device, 3-2

SOUNDCAPS, 3-73

SOUNDFORMAT, 3-75

SOUNDOPEN, 3-77

SxDM\_CLOSE, 3-39

SxDM\_OPEN, 3-42

SxDM\_QUERY\_Capabilities, 3-44

SxDM\_QUERY\_NumDevs, 3-45

SxDM\_QUERY\_SamplingRange, 3‑46

SxDM\_STATE\_Query, 3-48

SxDM\_STATE\_Set, 3-49

Disk Buffer, 1-1

ct?dSetDiskBuffer, 1-29, 1-51, 1-63

size, 1-29 to 1-30

DMA Buffer, *see* Transfer Buffer

Document Typographic Conventions, viii

DOS Error Codes, *see* Errors

Double Buffering, *see* Disk Buffer

Driver parameters, *see* Information

Drivers

parameters

.VOC, 1-45, 1-51

.WAV, 1-18, 1-56, 1-62

Dummy Keywords

IN, 3-1

OUT, 3-1

E\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Embedded DMA Buffer, 1-14

Entry-point, CTMMSYS.SYS, 3-2

Environment Setting

ct??GetEnvSettings, 1-8, 1-45, 1-51, 1-56, 1-62

ctmdGetEnvSettings, 4-6, 4-31

ctmdGetMidiEnvSettings, 4-8, 4-31

Error Messages, *see* CTMMSYS.SYS

Errors

ct??GetDrvError, 1-6, 1-49, 1-50, 1-61, 1‑64

ct??GetExtError, 1-49, 1-50, 1-61, 1-64

ct?dGetExtError, 1-10

DOS error codes, 1-10

error codes, CD-ROM, 5-4

error codes, digitized sound, 1-7

F\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fade and Pan Effects

ctadClrSource, 2-4, 2-33

ctadFade, 2-5, 2-31

ctadGetPanPosition, 2-11, 2-33

ctadPan, 2-15, 2-31

ctadPauseCtrl, 2-17, 2-32

ctadSetFadeStAddx, 2-20, 2-29

ctadSetPanStAddx, 2-23, 2-29

ctadStartCtrl, 2-26, 2-32

ctadStopCtrl, 2-27, 2-32

Fade Effects, *see* Fade and Pan Effects

File Format

.VOC format

data block, A-3 to A-11

header block, A-2

Creative ADPCM Wave Type Format, A‑12

Filter Control

AUXDM\_FILTER\_Get, 3-18

AUXDM\_FILTER\_QueryCaps, 3-19

AUXDM\_FILTER\_Set, 3-20

AUXFILTERCAPS, 3-61

cut-off frequencies, 3-20

Format, Function Reference, ix

Function Listing

AUXDRV.DRV, 2-3

CD-ROM functions, 5-3

CTMIDI.DRV, 4-3

high level digitized sound drivers, 1-3

Function Prefixes

ctad, 2-2

ctmd, 4-2

ctvd, 1-2

ctvm, 1-2

ctwd, 1-2

ctwm, 1-2

sbcd, 5-2

G\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Gain Control, *see* Automatic Gain Control

AUXDM\_GAIN\_Get, 3-21

AUXDM\_GAIN\_QueryCaps, 3-22

AUXDM\_GAIN\_Set, 3-23

AUXGAINCAPS, 3-62

ctadGetMixerGain, 2-9, 2-34

ctadSetMixerGain, 2-21, 2-34

Granularity, 3-79

H\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Handle, 3-42, 3-53

I\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I/O Parameters

ct??GetIOParam, 1-11, 1-46, 1-52, 1-58, 1‑65

ct??SetIOParam, 1-33, 1-37, 1-46, 1-52, 1‑58, 1-64

IN, *see* Dummy Keywords

Include Files for

AUXDRV.DRV, 2-2

CD-ROM functions, 5-2

CT‑VOICE.DRV, 1-2

CTMIDI.DRV, 4-2

CTVDSK.DRV, 1-2

CTWDSK.DRV, 1-2

CTWMEM.DRV, 1-2

Information, card and driver

ct?GetParam, 1-13, 1-16

Initialization

ct??GetEnvSettings, 1-8, 1-45, 1-51, 1-56, 1-62

ct??Init, 1-19, 1-44, 1-55, 1-56, 1-63

ctadGetEnvSettings, 2-8, 2-36

ctadInit, 2-14, 2-30

ctmdGetEnvSettings, 4-6, 4-31

ctmdGetMidiEnvSettings, 4-8, 4-31

ctmdInit, 4-9, 4-32

ctmdPrepareMidiStart, 4-12, 4-33

sbcdInit, 5-14

Input Midi

ctmdStartMidiInput, 4-27, 4-38

Input to Disk

ct?dInput, 1-21, 1-53, 1-66

Input to Memory

ct?mInputCM, 1-21, 1-46, 1-58

ct?mInputXM, 1-21, 1-47, 1-59

Interrupts

INT 10h, video interrupt, 1-19

INT 13h, disk interrupt, 1-19

INT 28h, DOS idle interrupt, 1-19

INT 8h, timer interrupt, 1-19

L\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Loop, *see* Break

M\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Mapper

ctmdGetMapperType, 4-7, 4-34

ctmdSetChannelMapper, 4-17, 4-32

ctmdSetMapperType, 4-19, 4-35

Memory, *see* Buffer Queue, Transfer Buffer, Midi Buffer

MEMORYDESC, 3-68

Message Listing

CTMMSYS.SYS, 3-5 to 3-6

Microphone, *see* Automatic Gain Control

Midi Buffer, 4-12, 4-20

ctmdSetMidiInputBuffer, 4-20, 4-37

Midi File Format, 4-12

Mixer

AUXDM\_MIXING\_Get, 3-25

AUXDM\_MIXING\_QueryCaps, 3-26

AUXDM\_MIXING\_Set, 3-27

AUXMIXINGCAPS, 3-63

AUXSETTINGS, 3-65

ctadGetMixerSwitch, 2-10, 2-36

ctadSetMixerSwitch, 2-22, 2-36

MMDEVICE\_AUX, 3-2

MMDEVICE\_CSP, 3-2

MMDEVICE\_SOUNDIN, 3-2

MMDEVICE\_SOUNDOUT, 3-2

MMSYSPROC, 3-2

MSCDEX, 5-14

O\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Open

AUXDM\_OPEN, 3-28

AUXOPEN, 3-64

CSPDM\_CODE\_Download, 3-52

CSPDM\_OPEN, 3-53

CSPOPEN, 3-85

SOUNDOPEN, 3-77

SxDM\_OPEN, 3-42

OUT, *see* Dummy Keywords

Output CD

sbcdPlay, 5-19

Output from Disk

ct?dOutput, 1-24, 1-53, 1-65

ctvdOutputOffset, 1-27, 1-55

Output from Memory

ct?mOutputCM, 1-24, 1-47, 1-59

ct?mOutputXM, 1-24, 1-47, 1-59

Output Midi

ctmdPlayMidiMusic, 4-11, 4-33

ctmdSendLongMessage, 4-15, 4-36

ctmdSendShortMessage, 4-16, 4-36

P\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pan Effects, *see* Fade and Pan Effects

PARAM\_UNUSED, 3-3

Pause

ct??Pause, 1-28, 1-48, 1-54, 1-60, 1-66

ctmdPauseMidiMusic, 4-10, 4-34

sbcdPause, 5-18

Playback, *see* Output from Disk, Output from Memory

Playback Control, CD

sbcdContinue, 5-6

sbcdFastForward, 5-8

sbcdNextTrack, 5-17

sbcdPause, 5-18

sbcdPlay, 5-19

sbcdPrevTrack, 5-20

sbcdRewind, 5-22

sbcdStop, 5-24

Position

measurement unit, 3-69

MMTIME, 3-69

SxDM\_POSITION\_Query, 3-43

Prefixes, *see* Function Prefixes

R\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Record, *see* Input to Disk, Input to Memory

Red Book Address, 5-11, 5-21

Red Book Format, 5-13

Reset

ctadResetMixer, 2-18, 2-33

ctmdResetMidiDriver, 4-13, 4-32

mixer, 3-24

S\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sampling Range, 1-15, 1-17

SOUNDSAMPLINGRANGE, 3-80

SxDM\_QUERY\_SamplingRange, 3-46

Sound Format

SOUNDFORMAT, 3-75

SOUNDCALLBACK, 3-4

Speaker

ct??SetSpeaker, 1-40, 1-44, 1-50, 1-57, 1‑65

SxDM\_MISC\_SetSpeaker, 3-41

State

CSPDM\_STATE\_Set, 3-56

SxDM\_STATE\_Query, 3-48

SxDM\_STATE\_Set, 3-49

Status Word

digitized sound, 1-23, 1-25, 1-28, 1-42

Midi

ctmdSetInputStatusAddx, 4-18, 4-36

ctmdSetOutputStatusAddx, 4-25, 4-33

Stop

ct??Stop, 1-42, 1-48, 1-53, 1-60, 1-67

ctmdStopMidiInput, 4-28, 4-38

ctmdStopMidiMusic, 4-29, 4-34

sbcdStop, 5-24

T\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tempo, Midi

ctmdSetMusicTempo, 4-23, 4-35

Terminate

ct??Terminate, 1-43, 1-44, 1-50, 1-57, 1-63

ctadTerminate, 2-28, 2-30

ctmdTerminate, 4-30, 4-32

Time Stamp, Midi, 4-20

ctmdSetTimeStampMode, 4-26, 4-37

Tone

AUXDM\_TONE\_Get, 3-31

AUXDM\_TONE\_QueryCaps, 3-32

AUXDM\_TONE\_Set, 3-33

AUXTONECAPS, 3-66

ctadGetToneLevel, 2-12, 2-35

ctadSetToneLevel, 2-24, 2-34

Transfer Buffer

ct??SetDMABuffer, 1-31, 1-45, 1-52, 1-57, 1-64

granularity, 3-79

SOUNDQYXFERBUF, 3-79

SOUNDXFERBUFDESC, 3-81

SxDM\_OPEN, 3-42

SxDM\_QUERY\_TransferBuffer, 3-47

Transpose, Midi

ctmdSetMusicTranspose, 4-24, 4-35

Treble, *see* Tone

V\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Version

ctadGetDrvVer, 2-7, 2-29

ctmdGetDrvVer, 4-5, 4-31

Volume

AUXDM\_VOLUME\_Get, 3-34

AUXDM\_VOLUME\_QueryCaps, 3-35

AUXDM\_VOLUME\_Set, 3-36

AUXVOLUMECAPS, 3-67

ctadGetVolume, 2-13, 2-31

ctadSetVolume, 2-25, 2-30

sbcdGetVolume, 5-13