ahi

COLLABORATORS			
	TITLE : ahi		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ahi

1.1 ahi.guide

TABLE OF CONTENTS

1.2 ahi.device/--background--

ahi.device/--background--

PURPOSE

The 'ahi.device' was first created because the lack of standards when it comes to sound cards on the Amiga. Another reason was to make it easier to write multi-channel music programs.

This device is by no means the final and perfect solution. But hopefully, it can evolve into something useful until AT brings you The Real Thing (TM).

OVERVIEW

Please see the document "AHI Developer's Guide" for more information.

* Driver based

Each supported sound card is controlled by a library-based audio driver. For a 'dumb' sound card, a new driver could be written in a few hours. For a 'smart' sound card, it is possible to utilize an on-board DSP, for example, to maximize performance and sound quality. For sound cards with own DSP but little or no memory, it is possible to use the main CPU to mix channels and do the post-processing with the DSP. Drivers are available for most popular sound cards, as well as an 8SVX (mono) and AIFF/AIFC (mono & stereo) sample render driver.

* Fast, powerful mixing routines (yeah, right... haha)

The device's mixing routines mix 8- or 16-bit signed samples, both mono and stereo, located in Fast-RAM and outputs 16-bit mono or stereo (with stereo panning if desired) data, using any number of channels (as long as 'any' means less than 128). Tables can be used speed the mixing up (especially when using 8-bit samples). The samples can have any length (including odd) and can have any number of loops. There are also so-called HiFi mixing routines that can be used, that use linear interpolation and gives 32 bit output.

* Support for non-realtime mixing

By providing a timing feature, it is possible to create high-quality output even if the processing power is lacking, by saving the output to disk, for example as an IFF AIFF or 8SXV file.

* Audio database

Uses ID codes, much like Screenmode IDs, to select the many parameters that can be set. The functions to access the audio database are not too different from those in 'graphics.library'. The device also features a requester to get an ID code from the user.

* Both high- and low-level protocol

By acting both like a device and a library, AHI gives the programmer a choice between full control and simplicity. The device API allows

several programs to use the audio hardware at the same time, and
the AUDIO: dos-device driver makes playing and recording sound very
simple for both the programmer and user.

* Future Compatible

When AmigaOS gets device-independent audio worth it's name, it should
not be too difficult to write a driver for AHI, allowing applications
using 'ahi.device' to automatically use the new OS interface. At
least I hope it wont.

1.3 ahi.device/AHI_AllocAudioA

ahi.device/AHI_AllocAudioA

NAME
AHI_AllocAudioA -- allocates and initializes the audio hardware
AHI_AllocAudio -- varargs stub for AHI_AllocAudioA()

SYNOPSIS
```
audioctrl = AHI_AllocAudioA( tags );
D0                           A1

struct AHIAudioCtrl *AHI_AllocAudioA( struct TagItem * );

audioctrl = AHI_AllocAudio( tag1, ... );

struct AHIAudioCtrl *AHI_AllocAudio( Tag, ... );
```

FUNCTION
Allocates and initializes the audio hardware, selects the best
mixing routine (if necessary) according to the supplied tags.
To start playing you first need to call AHI_ControlAudioA().

INPUTS
tags - A pointer to a tag list.

TAGS

AHIA_AudioID (ULONG) - The audio mode to use. Default is
AHI_DEFAULT_ID. (AHI_DEFAULT_ID is the ID the user has selected
in the preferences program. It's a good value to use the first
time she starts your application.)

AHIA_MixFreq (ULONG) - Desired mixing frequency. The actual
mixing rate may or may not be exactly what you asked for.
Default is AHI_DEFAULT_FREQ. (AHI_DEFAULT_FREQ is the user's
prefered frequency.)

AHIA_Channels (UWORD) - Number of channel to use. The actual
number of channels used will be equal or grater than the
requested. If too many channels were requested, this function
will fail. This tag must be supplied.

AHIA_Sounds (UWORD) - Number of sounds to use. This tag must be
supplied.

AHIA_SoundFunc (struct Hook *) - A function to call each time
when a sound has been started. The function receives the
following parameters:
A0 - (struct Hook *)
A2 - (struct AHIAudioCtrl *)
A1 - (struct AHISoundMessage *)
The hook may be called from an interrupt, so normal interrupt
restrictions apply.

The called function should follow normal register conventions,
which means that d2-d7 and a2-a6 must be preserved.

Default is NULL.

AHIA_PlayerFunc (struct Hook *) - A function to be called at regular
intervals. By using this hook there is no need for music players
to use other timing, such as VBLANK or CIA timers. But the real
reason it's present is that it makes it possible to do non-
realtime mixing to disk.

Using this interrupt source is currently the only supported way
to ensure that no mixing occurs between calls to AHI_SetVol(),
AHI_SetFreq() or AHI_SetSound().

If the sound playback is done without mixing, 'realtime.library'
is used to provide timing. The function receives the following
parameters:
A0 - (struct Hook *)
A2 - (struct AHIAudioCtrl *)
A1 - Undefined.
Do not assume A1 contains any particular value!
The hook may be called from an interrupt, so normal interrupt
restrictions apply.

The called function should follow normal register conventions,
which means that d2-d7 and a2-a6 must be preserved.

Default is NULL.

AHIA_PlayerFreq (Fixed) - If non-zero, enables timing and specifies
how many times per second PlayerFunc will be called. This must
be specified if AHIA_PlayerFunc is! Do not use any extreme
frequencies. The result of MixFreq/PlayerFreq must fit an UWORD,
ie it must be less or equal to 65535. It is also suggested that
you keep the result over 80. For normal use this should not be a
problem. Note that the data type is Fixed, not integer. 50 Hz is
50<<16.

Default is a reasonable value. Don't depend on it.

AHIA_MinPlayerFreq (Fixed) - The minimum frequency (AHIA_PlayerFreq) you will use. You MUST supply this if you are using the device's interrupt feature!

AHIA_MaxPlayerFreq (Fixed) - The maximum frequency (AHIA_PlayerFreq) you will use. You MUST supply this if you are using the device's interrupt feature!

AHIA_RecordFunc (struct Hook *) - This function will be called regularly when sampling is turned on (see AHI_ControlAudioA()) with the following parameters:
A0 - (struct Hook *)
A2 - (struct AHIAudioCtrl *)
A1 - (struct AHIRecordMessage *)
The message (AHIRecordMessage) is filled as follows:
ahirm_Buffer - Pointer to the samples. The buffer is valid until next time the Hook is called.
ahirm_Length - Number of sample FRAMES in buffer. To get the size in bytes, multiply by 4 if ahiim_Type is AHIST_S16S.
ahirm_Type - Always AHIST_S16S at the moment, but you *must* check this, since it may change in the future!
The hook may be called from an interrupt, so normal interrupt restrictions apply. Signal a process if you wish to save the buffer to disk. The called function should follow normal register conventions, which means that d2-d7 and a2-a6 must be preserved.

NOTE: The function MUST return NULL (in d0). This was previously not documented. Now you know.

Default is NULL.

AHIA_UserData (APTR) - Can be used to initialize the ahiac_UserData field. Default is 0.

RESULT
A pointer to an AHIAudioCtrl structure or NULL if an error occured.

EXAMPLE

NOTES
SoundFunc will be called in the same manner as Paula interrupts occur; when the device has updated its internal variables and can accept new commands.

BUGS

SEE ALSO
AHI_FreeAudio(), AHI_ControlAudioA()

1.4 ahi.device/AHI_AllocAudioRequestA

```
ahi.device/AHI_AllocAudioRequestA

NAME
       AHI_AllocAudioRequestA -- allocate an audio mode requester.
       AHI_AllocAudioRequest -- varargs stub for AHI_AllocAudioRequestA()


SYNOPSIS
       requester = AHI_AllocAudioRequestA( tags );
       D0                                  A0

       struct AHIAudioModeRequester *AHI_AllocAudioRequestA(
           struct TagItem * );

       requester = AHI_AllocAudioRequest( tag1, ... );

       struct AHIAudioModeRequester *AHI_AllocAudioRequest( Tag, ... );


FUNCTION
       Allocates an audio mode requester data structure.


INPUTS
       tags - A pointer to an optional tag list specifying how to initialize
           the data structure returned by this function. See the
           documentation for AHI_AudioRequestA() for an explanation of how
           to use the currently defined tags.


RESULT
       requester - An initialized requester data structure, or NULL on
           failure.


EXAMPLE


NOTES
       The requester data structure is READ-ONLY and can only be modified
       by using tags!


BUGS


SEE ALSO
      AHI_AudioRequestA(), AHI_FreeAudioRequest()
```

1.5 ahi.device/AHI_AudioRequestA

ahi.device/AHI_AudioRequestA

NAME
AHI_AudioRequestA -- get an audio mode from user using an requester.
AHI_AudioRequest -- varargs stub for AHI_AudioRequestA()

SYNOPSIS
```
success = AHI_AudioRequestA( requester, tags );
D0                           A0         A1

BOOL AHI_AudioRequestA( struct AHIAudioModeRequester *,
    struct TagItem * );

result = AHI_AudioRequest( requester, tag1, ... );

BOOL AHI_AudioRequest( struct AHIAudioModeRequester *, Tag, ... );
```

FUNCTION
Prompts the user for an audio mode, based on the modifying tags. If the user cancels or the system aborts the request, FALSE is returned, otherwise the requester's data structure reflects the user input.

Note that tag values stay in effect for each use of the requester until they are cleared or modified by passing the same tag with a new value.

INPUTS
requester - Requester structure allocated with AHI_AllocAudioRequestA(). If this parameter is NULL, this function will always return FALSE with a dos.library/IoErr() result of ERROR_NO_FREE_STORE.
tags - Pointer to an optional tag list which may be used to control features of the requester.

TAGS
Tags used for the requester (they look remarkable similar to the screen mode requester in ASL, don't they? ;-))

AHIR_Window (struct Window *) - Parent window of requester. If no AHIR_Screen tag is specified, the window structure is used to determine on which screen to open the requesting window.

AHIR_PubScreenName (STRPTR) - Name of a public screen to open on. This overrides the screen used by AHIR_Window.

AHIR_Screen (struct Screen *) - Screen on which to open the requester. This overrides the screen used by AHIR_Window or by AHIR_PubScreenName.

AHIR_PrivateIDCMP (BOOL) - When set to TRUE, this tells AHI to allocate a new IDCMP port for the requesting window. If not

specified or set to FALSE, and if AHIR_Window is provided, the requesting window will share AHIR_Window's IDCMP port.

AHIR_IntuiMsgFunc (struct Hook *) - A function to call whenever an unknown Intuition message arrives at the message port being used by the requesting window. The function receives the following parameters:
A0 - (struct Hook *)
A1 - (struct IntuiMessage *)
A2 - (struct AHIAudioModeRequester *)

AHIR_SleepWindow (BOOL) - When set to TRUE, this tag will cause the window specified by AHIR_Window to be "put to sleep". That is, a busy pointer will be displayed in the parent window, and no gadget or menu activity will be allowed. This is done by opening an invisible Intuition Requester in the parent window.

AHIR_UserData (APTR) - A 32-bit value that is simply copied in the ahiam_UserData field of the requester structure.

AHIR_TextAttr (struct TextAttr *) - Font to be used for the requesting window's gadgets and menus. If this tag is not provided or its value is NULL, the default font of the screen on which the requesting window opens will be used. This font must already be in memory as AHI calls OpenFont() and not OpenDiskFont().

AHIR_Locale (struct Locale *) - Locale to use for the requesting window. This determines the language used for the requester's gadgets and menus. If this tag is not provided or its value is NULL, the system's current default locale will be used.

AHIR_TitleText (STRPTR) - Title to use for the requesting window. Default is no title.

AHIR_PositiveText (STRPTR) - Label of the positive gadget in the requester. English default is "OK".

AHIR_NegativeText (STRPTR) - Label of the negative gadget in the requester. English default is "Cancel".

AHIR_InitialLeftEdge (WORD) - Suggested left edge of requesting window.

AHIR_InitialTopEdge (WORD) - Suggested top edge of requesting window.

AHIR_InitialWidth (WORD) - Suggested width of requesting window.

AHIR_InitialHeight (WORD) - Suggested height of requesting window.

AHIR_InitialAudioID (ULONG) - Initial setting of the Mode list view gadget (ahiam_AudioID). Default is ~0 (AHI_INVALID_ID), which means that no mode will be selected.

AHIR_InitialMixFreq (ULONG) - Initial setting of the frequency slider. Default is the lowest frequency supported.

AHIR_InitialInfoOpened (BOOL) - Whether to open the property information window automatically. Default is FALSE.

AHIR_InitialInfoLeftEdge (WORD) - Initial left edge of information window.

AHIR_InitialInfoTopEdge (WORD) - Initial top edge of information window.

AHIR_DoMixFreq (BOOL) - Set this tag to TRUE to cause the requester to display the frequency slider gadget. Default is FALSE.

AHIR_DoDefaultMode (BOOL) - Set this tag to TRUE to let the user select the mode she has set in the preferences program. If she selects this mode, ahiam_AudioID will be AHI_DEFAULT_ID and ahiam_MixFreq will be AHI_DEFAULT_FREQ. Note that if you filter the mode list (see below), you must also check the mode (with AHI_BestAudioIDA()) before you use it since the user may change the meaning of AHI_DEFAULT_MODE anytime, without your knowledge. Default is FALSE. (V4)

AHIR_FilterFunc (struct Hook *) - A function to call for each mode encountered. If the function returns TRUE, the mode is included in the file list, otherwise it is rejected and not displayed. The function receives the following parameters:

A0 - (struct Hook *)
A1 - (ULONG) mode id
A2 - (struct AHIAudioModeRequester *)

AHIR_FilterTags (struct TagItem *) - A pointer to a tag list used to filter modes away, like AHIR_FilterFunc does. The tags are the same as AHI_BestAudioIDA() takes as arguments. See that function for an explanation of each tag.

RESULT

result - FALSE if the user cancelled the requester or if something prevented the requester from opening. If TRUE, values in the requester structure will be set.

If the return value is FALSE, you can look at the result from the dos.library/IoErr() function to determine whether the requester was cancelled or simply failed to open. If dos.library/IoErr() returns 0, then the requester was cancelled, any other value indicates a failure to open. Current possible failure codes are ERROR_NO_FREE_STORE which indicates there was not enough memory, and ERROR_NO_MORE_ENTRIES which indicates no modes were available (usually because the application filter hook filtered them all away).

EXAMPLE

NOTES

The requester data structure is READ-ONLY and can only be modified

```
       by using tags!

       The mixing/recording frequencies that are presented to the user
       may not be the only ones a driver supports, but just a selection.


BUGS


SEE ALSO
      AHI_AllocAudioRequestA(), AHI_FreeAudioRequest()
```

1.6 ahi.device/AHI_BestAudioIDA

```
ahi.device/AHI_BestAudioIDA

NAME
       AHI_BestAudioIDA -- calculate the best ModeID with given parameters
       AHI_BestAudioID -- varargs stub for AHI_BestAudioIDA()


SYNOPSIS
       ID = AHI_BestAudioIDA( tags );
       D0                     A1

       ULONG AHI_BestAudioIDA( struct TagItem * );

       ID = AHI_BestAudioID( tag1, ... );

       ULONG AHI_BestAudioID( Tag, ... );


FUNCTION
       Determines the best AudioID to fit the parameters set in the tag
       list.


INPUTS
       tags - A pointer to a tag list. Only the tags present matter.


TAGS
       Many combinations are probably stupid to ask for, like not supporting
       panning or recording.

       AHIDB_AudioID (ULONG) - The mode must use the same audio hardware
           as this mode does.

       AHIDB_Volume (BOOL) - If TRUE: mode must support volume changes.
           If FALSE: mode must not support volume changes.

       AHIDB_Stereo (BOOL) - If TRUE: mode must have stereo output.
           If FALSE: mode must not have stereo output (=mono).
```

AHIDB_Panning (BOOL) - If TRUE: mode must support volume panning. If FALSE: mode must not support volume panning.

AHIDB_HiFi (BOOL) - If TRUE: mode must have HiFi output. If FALSE: mode must not have HiFi output.

AHIDB_PingPong (BOOL) - If TRUE: mode must support playing samples backwards. If FALSE: mode must not support playing samples backwards.

AHIDB_Record (BOOL) - If TRUE: mode must support recording. If FALSE: mode must not support recording.

AHIDB_Realtime (BOOL) - If TRUE: mode must be realtime. If FALSE: take a wild guess.

AHIDB_FullDuplex (BOOL) - If TRUE: mode must be able to record and play at the same time.

AHIDB_Bits (UBYTE) - Mode must have greater or equal number of bits.

AHIDB_MaxChannels (UWORD) - Mode must have greater or equal number of channels.

AHIDB_MinMixFreq (ULONG) - Lowest mixing frequency supported must be less or equal.

AHIDB_MaxMixFreq (ULONG) - Highest mixing frequency must be greater or equal.

AHIB_Dizzy (struct TagItem *) - This tag points to a second tag list. After all other tags has been tested, the mode that matches these tags best is returned, i.e. the one that has most of the features you ask for, and least of the ones you don't want. Without this second tag list, this function hardly does what its name suggests. (V4)

RESULT

ID - The best AudioID to use or AHI_INVALID_ID if none of the modes in the audio database could meet the requirements.

EXAMPLE

NOTES

BUGS

Due to a bug in the code that compared the boolean tag values in version 4.158 and earlier, TRUE must be equal to 1. The bug is not present in later revisions.

SEE ALSO

```
      AHI_NextAudioID(), AHI_GetAudioAttrsA()
```

1.7 ahi.device/AHI_ControlAudioA

```
ahi.device/AHI_ControlAudioA

NAME
       AHI_ControlAudioA -- change audio attributes
       AHI_ControlAudio -- varargs stub for AHI_ControlAudioA()


SYNOPSIS
       error = AHI_ControlAudioA( audioctrl, tags );
       D0                         A2         A1

       ULONG AHI_ControlAudioA( struct AHIAudioCtrl *, struct TagItem * );

       error = AHI_ControlAudio( AudioCtrl, tag1, ...);

       ULONG AHI_ControlAudio( struct AHIAudioCtrl *, Tag, ... );


FUNCTION
       This function should be used to change attributes for a given
       AHIAudioCtrl structure. It is also used to start and stop playback,
       and to control special hardware found on some sound cards.


INPUTS
       audioctrl - A pointer to an AHIAudioCtrl structure.
       tags - A pointer to a tag list.


TAGS
       AHIC_Play (BOOL) - Starts (TRUE) and stops (FALSE) playback and
           PlayerFunc. NOTE: If the audio hardware cannot play at the same
           time as recording samples, the recording will be stopped.

       AHIC_Record (BOOL) - Starts (TRUE) and stops (FALSE) sampling and
           RecordFunc. NOTE: If the audio hardware cannot record at the same
           time as playing samples, the playback will be stopped.

       AHIC_MonitorVolume (Fixed) - Sets the input monitor volume, i.e. how
           much of the input signal is mixed with the output signal while
           recording. Use AHI_GetAudioAttrsA() to find the available range.

       AHIC_MonitorVolume_Query (Fixed *) - Get the current input monitor
           volume. ti_Data is a pointer to a Fixed variable, where the result
           will be stored.

       AHIC_MixFreq_Query (ULONG *) - Get the current mixing frequency.
           ti_Data is a pointer to an ULONG variable, where the result will
           be stored.
```

AHIC_InputGain (Fixed) - Set the input gain. Use AHI_GetAudioAttrsA() to find the available range. (V2)

AHIC_InputGain_Query (Fixed *) - Get current input gain. (V2)

AHIC_OutputVolume (Fixed) - Set the output volume. Use AHI_GetAudioAttrsA() to find the available range. (V2)

AHIC_OutputVolume_Query (Fixed *) - Get current output volume. (V2)

AHIC_Input (ULONG) - Select input source. See AHI_GetAudioAttrsA(). (V2)

AHIC_Input_Query (ULONG *) - Get current input source. (V2)

AHIC_Output (ULONG) - Select destination for output. See AHI_GetAudioAttrsA(). (V2)

AHIC_Output_Query (ULONG *) - Get destination for output. (V2)

The following tags are also recognized by AHI_ControlAudioA(). See AHI_AllocAudioA() for what they do. They may be used from interrupts.

AHIA_SoundFunc (struct Hook *)
AHIA_PlayerFunc (struct Hook *)
AHIA_PlayerFreq (Fixed)
AHIA_RecordFunc (struct Hook *)
AHIA_UserData (APTR)

Note that AHIA_PlayerFreq must never be outside the limits specified with AHIA_MinPlayerFreq and AHIA_MaxPlayerFreq! Also note that the timing feature is designed to be used for music. When you change the frequency, be reasonable. Using 50 Hz one moment and 5 the other is to ask for trouble.

RESULT

An error code, defined in <devices/ahi.h>.

EXAMPLE

NOTES

The AHIC_Play and AHIC_Record tags *must not* be used from interrupts.

BUGS

SEE ALSO

AHI_AllocAudioA(), AHI_GetAudioAttrsA(), <devices/ahi.h>

1.8 ahi.device/AHI_FreeAudio

```
ahi.device/AHI_FreeAudio

NAME
       AHI_FreeAudio -- deallocates the audio hardware


SYNOPSIS
       AHI_FreeAudio( audioctrl );
                      A2

       void AHI_FreeAudio( struct AHIAudioCtrl * );


FUNCTION
       Deallocates the AHIAudioCtrl structure and any other resources
       allocated by AHI_AllocAudioA(). After this call it must not be used
       by any other functions anymore. AHI_UnloadSound() is automatically
       called for every sound.


INPUTS
       audioctrl - A pointer to an AHIAudioCtrl structure obtained from
           AHI_AllocAudioA(). If NULL, this function does nothing.


EXAMPLE


NOTES


BUGS


SEE ALSO
       AHI_AllocAudioA(), AHI_UnloadSound()
```

1.9 ahi.device/AHI_FreeAudioRequest

```
ahi.device/AHI_FreeAudioRequest

NAME
       AHI_FreeAudioRequest -- frees requester resources


SYNOPSIS
       AHI_FreeAudioRequest( requester );
                             A0

       void AHI_FreeAudioRequest( struct AHIAudioModeRequester * );
```

```
FUNCTION
       Frees any resources allocated by AHI_AllocAudioRequestA(). Once a
       requester has been freed, it can no longer be used with other calls to
       AHI_AudioRequestA().


INPUTS
       requester - Requester obtained from AHI_AllocAudioRequestA(), or NULL
       in which case this function does nothing.


RESULT


EXAMPLE


NOTES


BUGS


SEE ALSO
      AHI_AllocAudioRequestA()
```

1.10 ahi.device/AHI_GetAudioAttrsA

```
ahi.device/AHI_GetAudioAttrsA

NAME
       AHI_GetAudioAttrsA -- examine an audio mode via a tag list
       AHI_GetAudioAttrs -- varargs stub for AHI_GetAudioAttrsA()


SYNOPSIS
       success = AHI_GetAudioAttrsA( ID, [audioctrl], tags );
       D0                            D0  A2           A1

       BOOL AHI_GetAudioAttrsA( ULONG, struct AHIAudioCtrl *,
                                struct TagItem * );

       success = AHI_GetAudioAttrs( ID, [audioctrl], attr1, &result1, ...);

       BOOL AHI_GetAudioAttrs( ULONG, struct AHIAudioCtrl *, Tag, ... );


FUNCTION
       Retrieve information about an audio mode specified by ID or audioctrl
       according to the tags in the tag list. For each entry in the tag
       list, ti_Tag identifies the attribute, and ti_Data is mostly a
       pointer to a LONG (4 bytes) variable where you wish the result to be
       stored.
```

INPUTS

ID - An audio mode identifier, AHI_DEFAULT_ID (V4) or AHI_INVALID_ID.
audioctrl - A pointer to an AHIAudioCtrl structure, only used if
ID equals AHI_INVALID_ID. Set to NULL if not used. If set to
NULL when used, this function returns immediately. Always set
ID to AHI_INVALID_ID and use audioctrl if you have allocated
a valid AHIAudioCtrl structure. Some of the tags return incorrect
values otherwise.
tags - A pointer to a tag list.

TAGS

AHIDB_Volume (ULONG *) - TRUE if this mode supports volume changes.

AHIDB_Stereo (ULONG *) - TRUE if output is in stereo. Unless
AHIDB_Panning (see below) is TRUE, all even channels are played
to the left and all odd to the right.

AHIDB_Panning (ULONG *) - TRUE if this mode supports stereo panning.

AHIDB_HiFi (ULONG *) - TRUE if no shortcuts, like pre-division, is
used by the mixing routines.

AHIDB_PingPong (ULONG *) - TRUE if this mode can play samples backwards.

AHIDB_Record (ULONG *) - TRUE if this mode can record samples.

AHIDB_FullDuplex (ULONG *) - TRUE if this mode can record and play at
the same time.

AHIDB_Realtime (ULONG *) - Modes which return TRUE for this fulfills
two criteria:
1) Calls to AHI_SetVol(), AHI_SetFreq() or AHI_SetSound() will be
performed within (about) 10 ms if called from a PlayFunc Hook.
2) The PlayFunc Hook will be called at the specified frequency.
If you don't use AHI's PlayFunc Hook, you must not use modes that
are not realtime. (Criterium 2 is not that obvious if you consider
a mode that renders the output to disk as a sample.)

AHIDB_Bits (ULONG *) - The number of output bits (8, 12, 14, 16 etc).

AHIDB_MaxChannels (ULONG *) - The maximum number of channels this mode
can handle.

AHIDB_MinMixFreq (ULONG *) - The minimum mixing frequency supported.

AHIDB_MaxMixFreq (ULONG *) - The maximum mixing frequency supported.

AHIDB_Frequencies (ULONG *) - The number of different sample rates
available.

AHIDB_FrequencyArg (ULONG) - Specifies which frequency
AHIDB_Frequency should return (see below). Range is 0 to
AHIDB_Frequencies-1 (including).
NOTE: ti_Data is NOT a pointer, but an ULONG.

AHIDB_Frequency (ULONG *) - Return the frequency associated with the
    index number specified with AHIDB_FrequencyArg (see above).

AHIDB_IndexArg (ULONG) - AHIDB_Index will return the index which
    gives the closest frequency to AHIDB_IndexArg
    NOTE: ti_Data is NOT a pointer, but an ULONG.

AHIDB_Index (ULONG *) - Return the index associated with the frequency
    specified with AHIDB_IndexArg (see above).

AHIDB_MaxPlaySamples (ULONG *) - Return the lowest number of sample
    frames that must be present in memory when AHIST_DYNAMICSAMPLE
    sounds are used. This number must then be scaled by Fs/Fm, where
    Fs is the frequency of the sound and Fm is the mixing frequency.

AHIDB_MaxRecordSamples (ULONG *) - Return the number of sample frames
    you will receive each time the RecordFunc is called.

AHIDB_BufferLen (ULONG) - Specifies how many characters will be
    copied when requesting text attributes. Default is 0, which
    means that AHIDB_Driver, AHIDB_Name, AHIDB_Author,
    AHIDB_Copyright, AHIDB_Version and AHIDB_Annotation,
    AHIDB_Input and AHIDB_Output will do nothing.

AHIDB_Driver (STRPTR) - Name of driver (excluding path and
    extension).
    NOTE: ti_Data is a pointer to an UBYTE array where the name
    will be stored. See AHIDB_BufferLen.

AHIDB_Name (STRPTR) - Human readable name of this mode.
    NOTE: ti_Data is a pointer to an UBYTE array where the name
    will be stored. See AHIDB_BufferLen.

AHIDB_Author (STRPTR) - Name of driver author.
    NOTE: ti_Data is a pointer to an UBYTE array where the name
    will be stored. See AHIDB_BufferLen.

AHIDB_Copyright (STRPTR) - Driver copyright notice.
    NOTE: ti_Data is a pointer to an UBYTE array where the name
    will be stored. See AHIDB_BufferLen

AHIDB_Version (STRPTR) - Driver version string.
    NOTE: ti_Data is a pointer to an UBYTE array where the name
    will be stored. See AHIDB_BufferLen.

AHIDB_Annotation (STRPTR) - Annotation by driver author.
    NOTE: ti_Data is a pointer to an UBYTE array where the name
    will be stored. See AHIDB_BufferLen.

AHIDB_MinMonitorVolume (Fixed *)
AHIDB_MaxMonitorVolume (Fixed *) - Lower/upper limit for input
    monitor volume, see AHI_ControlAudioA(). If both are 0.0,
    the sound hardware does not have an input monitor feature.
    If both are same, but not 0.0, the hardware always sends the
    recorded sound to the outputs (at the given volume). (V2)

AHIDB_MinInputGain (Fixed *)
AHIDB_MaxInputGain (Fixed *) - Lower/upper limit for input gain, see AHI_ControlAudioA(). If both are same, there is no input gain hardware. (V2)

AHIDB_MinOutputVolume (Fixed *)
AHIDB_MaxOutputVolume (Fixed *) - Lower/upper limit for output volume, see AHI_ControlAudioA(). If both are same, the sound card does not have volume control. (V2)

AHIDB_Inputs (ULONG *) - The number of inputs the sound card has. (V2)

AHIDB_InputArg (ULONG) - Specifies what AHIDB_Input should return (see below). Range is 0 to AHIDB_Inputs-1 (including). NOTE: ti_Data is NOT a pointer, but an ULONG. (V2)

AHIDB_Input (STRPTR) - Gives a human readable string describing the input associated with the index specified with AHIDB_InputArg (see above). See AHI_ControlAudioA() for how to select one. NOTE: ti_Data is a pointer to an UBYTE array where the name will be stored. See AHIDB_BufferLen. (V2)

AHIDB_Outputs (ULONG *) - The number of outputs the sound card has. (V2)

AHIDB_OutputArg (ULONG) - Specifies what AHIDB_Output should return (see below). Range is 0 to AHIDB_Outputs-1 (including) NOTE: ti_Data is NOT a pointer, but an ULONG. (V2)

AHIDB_Output (STRPTR) - Gives a human readable string describing the output associated with the index specified with AHIDB_OutputArg (see above). See AHI_ControlAudioA() for how to select one. NOTE: ti_Data is a pointer to an UBYTE array where the name will be stored. See AHIDB_BufferLen. (V2)

AHIDB_AudioID (ULONG *) - The ID for this mode. (V4)

If the requested information cannot be found, the variable will be not be touched.

RESULT

TRUE if everything went well.

EXAMPLE

NOTES

BUGS

In versions earlier than 3, the tags that filled a string buffer would not NULL-terminate the string on buffer overflows.

```
SEE ALSO
      AHI_NextAudioID(), AHI_BestAudioIDA()
```

1.11 ahi.device/AHI_LoadSound

```
ahi.device/AHI_LoadSound

NAME
       AHI_LoadSound -- prepare a sound for playback


SYNOPSIS
       error = AHI_LoadSound( sound, type, info, audioctrl );
       D0                     D0:16  D1    A0    A2

       ULONG AHI_LoadSound( UWORD, ULONG, APTR, struct AHIAudioCtrl * );


FUNCTION
       Defines an ID number for the sound and prepares it for playback.


INPUTS
       sound - The numeric ID to be used as a reference to this sound.
           The ID is a number greater or equal to 0 and less than what you
           specified with AHIA_Sounds when you called AHI_AllocAudioA().
       type - The type of the sound. Currently four types are supported:
           AHIST_SAMPLE - array of 8 or 16 bit samples. Note that the
               portion of memory where the sample is stored must NOT be
               altered until AHI_UnloadSound() has been called! This is
               because some audio drivers may wish to upload the samples
               to local RAM. It is OK to read, though.

           AHIST_DYNAMICSAMPLE - array of 8 or 16 bit samples, which can be
               updated dynamically. Typically used to play data that is
               loaded from disk or calculated realtime.
               Avoid using this sound type as much as possible; it will
               use much more CPU power than AHIST_SAMPLE on a DMA/DSP
               sound card.

           AHIST_INPUT - The input from your sampler (not fully functional
               yet).

       info - Depends on type:
           AHIST_SAMPLE - A pointer to a struct AHISampleInfo, filled with:
               ahisi_Type - Format of samples (four formats are supported).
                   AHIST_M8S: Mono, 8 bit signed (BYTEs).
                   AHIST_S8S: Stereo, 8 bit signed (2$\times$BYTEs) (V4).
                   AHIST_M16S: Mono, 16 bit signed (WORDs).
                   AHIST_S16S: Stereo, 16 bit signed (2$\times$WORDs) (V4).
               ahisi_Address - Address to the sample array.
               ahisi_Length - The size of the array, in samples.
               Don't even think of setting ahisi_Address to 0 and
               ahisi_Length to 0xffffffff as you can do with
```

```
                AHIST_DYNAMICSAMPLE! Very few DMA/DSP cards have 4 GB onboard
                RAM...

            AHIST_DYNAMICSAMPLE A pointer to a struct AHISampleInfo, filled
                as described above (AHIST_SAMPLE).
                If ahisi_Address is 0 and ahisi_Length is 0xffffffff
                AHI_SetSound() can take the real address of an 8 bit sample
                to be played as offset argument. Unfortunately, this does not
                work for 16 bit samples.

            AHIST_INPUT - Always set info to NULL.
                Note that AHI_SetFreq() may only be called with AHI_MIXFREQ
                for this sample type.

        audioctrl - A pointer to an AHIAudioCtrl structure.


RESULT
        An error code, defined in <devices/ahi.h>.


EXAMPLE


NOTES
        There is no need to place a sample array in Chip memory, but it
        MUST NOT be swapped out! Allocate your sample memory with the
        MEMF_PUBLIC flag set.


BUGS
        AHIST_INPUT does not fully work yet.


SEE ALSO
        AHI_UnloadSound(), AHI_SetEffect(), AHI_SetFreq(), AHI_SetSound(),
        AHI_SetVol(), <devices/ahi.h>
```

1.12 ahi.device/AHI_NextAudioID

```
ahi.device/AHI_NextAudioID

NAME
        AHI_NextAudioID -- iterate current audio mode identifiers


SYNOPSIS
        next_ID = AHI_NextAudioID( last_ID );
        D0                         D0

        ULONG AHI_NextAudioID( ULONG );


FUNCTION
```

```
       This function is used to iterate through all current AudioIDs in
       the audio database.


INPUTS
       last_ID - previous AudioID or AHI_INVALID_ID if beginning iteration.


RESULT
       next_ID - subsequent AudioID or AHI_INVALID_ID if no more IDs.


EXAMPLE


NOTES


BUGS


SEE ALSO
      AHI_GetAudioAttrsA(), AHI_BestAudioIDA()
```

1.13 ahi.device/AHI_PlayA

```
ahi.device/AHI_PlayA

NAME
       AHI_PlayA -- Start multiple sounds in one call (V4)
       AHI_Play -- varargs stub for AHI_PlayA()


SYNOPSIS
       AHI_PlayA( audioctrl, tags );
                  A2         A1

       void AHI_PlayA( struct AHIAudioCtrl *, struct TagItem * );

       AHI_Play( AudioCtrl, tag1, ...);

       void AHI_Play( struct AHIAudioCtrl *, Tag, ... );


FUNCTION
       This function performs the same actions as multiple calls to
       AHI_SetFreq(), AHI_SetSound() and AHI_SetVol(). The advantages
       of using only one call is that simple loops can be set without
       using a SoundFunc (see AHI_AllocAudioA(), tag AHIA_SoundFunc) and
       that sounds on different channels can be synchronized even when the
       sounds are not started from a PlayerFunc (see AHI_AllocAudioA(), tag
       AHIA_PlayerFunc). The disadvantage is that this call has more
       overhead than AHI_SetFreq(), AHI_SetSound() and AHI_SetVol(). It is
       therefore recommended that you only use this call if you are not
```

calling from a SoundFunc or PlayerFunc.

The supplied tag list works like a 'program'. This means that the order of tags matter.

INPUTS

audioctrl - A pointer to an AHIAudioCtrl structure.
tags - A pointer to a tag list.

TAGS

AHIP_BeginChannel (UWORD) - Before you start setting attributes for a sound to play, you have to use this tag to chose a channel to operate on. If AHIP_BeginChannel is omitted, the result is undefined.

AHIP_EndChannel (ULONG) - Signals the end of attributes for the current channel. If AHIP_EndChannel is omitted, the result is undefined. ti_Data MUST BE NULL!

AHIP_Freq (ULONG) - The playback frequency in Hertz or AHI_MIXFREQ.

AHIP_Vol (Fixed) - The desired volume. If omitted, but AHIP_Pan is present, AHIP_Vol defaults to 0.

AHIP_Pan (sposition) - The desired panning. If omitted, but AHIP_Vol is present, AHIP_Pan defaults to 0 (extreme left).

AHIP_Sound (UWORD) - Sound to be played, or AHI_NOSOUND.

AHIP_Offset (ULONG) - Specifies an offset (in samples) into the sound. If this tag is present, AHIP_Length MUST be present too!

AHIP_Length (LONG) - Specifies how many samples that should be player.

AHIP_LoopFreq (ULONG)
AHIP_LoopVol (Fixed)
AHIP_LoopPan (sposition)
AHIP_LoopSound (UWORD)
AHIP_LoopOffset (ULONG)
AHIP_LoopLength (LONG) - These tags can be used to set simple loop attributes. They default to their sisters. These tags must be after the other tags.

RESULT

EXAMPLE

NOTES

BUGS

```
SEE ALSO
       AHI_SetFreq(), AHI_SetSound(), AHI_SetVol()
```

1.14 ahi.device/AHI_SampleFrameSize

```
ahi.device/AHI_SampleFrameSize

NAME
       AHI_SampleFrameSize -- get the size of a sample frame (V4)


SYNOPSIS
       size = AHI_SampleFrameSize( sampletype );
       D0                          D0

       ULONG AHI_SampleFrameSize( ULONG );


FUNCTION
       Returns the size in bytes of a sample frame for a given sample type.


INPUTS
       sampletype - The sample type to examine. See <devices/ahi.h> for
           possible types.


RESULT


EXAMPLE


NOTES


BUGS


SEE ALSO
      <devices/ahi.h>
```

1.15 ahi.device/AHI_SetEffect

```
ahi.device/AHI_SetEffect

NAME
       AHI_SetEffect -- set effect
```

```
SYNOPSIS
       error = AHI_SetEffect( effect, audioctrl );
       d0                     A0      A2

       ULONG AHI_SetEffect( APTR, struct AHIAudioCtrl * );


FUNCTION
       Selects an effect to be used, described by a structure.


INPUTS
       effect - A pointer to an effect data structure, as defined in
           <devices/ahi.h>. The following effects are defined:
           AHIET_MASTERVOLUME - Changes the volume for all channels. Can
               also be used to boost volume over 100%.
           AHIET_OUTPUTBUFFER - Gives READ-ONLY access to the mixed output.
               Can be used to show nice scopes and VU-meters.
           AHIET_DSPMASK - Select which channels will be affected by the
               DSP effects. (V4)
           AHIET_DSPECHO - A DSP effects that adds (cross-)echo and delay.
               (V4)
           AHIET_CHANNELINFO - Get info about all channels. (V4)
       audioctrl - A pointer to an AHIAudioCtrl structure.


EFFECTS
       AHIET_MASTERVOLUME - Effect is a struct AHIEffMasterVolume, with
           ahiemv_Volume set to the desired volume. The range is 0 to
           (channels/hardware channel). Assume you have 4 channels in
           mono mode. The range is then 0.0 to 4.0. The range is the same
           if the mode is stereo with panning. However, assume you have 4
           channels with a stereo mode *without* panning. Then you have two
           channels to the left and two to the right => range is 0.0 - 2.0.
           Setting the volume outside the range will give an unpredictable
           result!

       AHIET_OUTPUTBUFFER - Effect is a struct AHIEffOutputBuffer, with
           ahieob_Func pointing to a hook that will be called with the
           following parameters:
               A0 - (struct Hook *)
               A2 - (struct AHIAudioCtrl *)
               A1 - (struct AHIEffOutputBuffer *)
           The information you are looking for then is in ahieob_Type,
           ahieob_Buffer and ahieob_Length. Always check ahieob_Type!
           ahieob_Length is neither in bytes nor samples, but sample frames.

       AHIET_DSPMASK - Effect is a struct AHIEffDSPMask, where ahiedm_Mask
           is an array with ahiedm_Channels elements. Each UBYTE in the
           array can either make the channel 'wet' (affected by the DSP
           effects), by using the AHIEDM_WET constant or 'dry' (not
           affected by the DSP effects) by using the AHIEDM_DRY constant.
           The default is all channels wet. If ahiedm_Channels does not
           equal the current number of channels allocated, the result of
           this call is undefined (crash warning!). (V4)
```

AHIET_DSPECHO - Effect is a struct AHIEffDSPEcho.
ahiede_Delay is the delay in samples (and thus depends on the mixing rate).

ahiede_Feedback is a Fixed value between 0 and 1.0, and defines how much of the delayed signal should be feed back to the delay stage. Setting this to 0 gives a delay effect, otherwise echo.

ahiede_Mix tells how much of the delayed signal should be mixed with the normal signal. Setting this to 0 disables delay/echo, and setting it to 1.0 outputs only the delay/echo signal.

ahiede_Cross only has effect of the current playback mode is stereo. It tells how the delayed signal should be panned to the other channel. 0 means no cross echo, 1.0 means full cross echo.

If the user has enabled "Fast Echo", AHI may take several short-cuts to increase the performance. This could include rounding the parameters to a power of two, or even to the extremes.

If you set ahiede_Mix to 0x10000 and ahiede_Cross to 0x0, much faster mixing routines will be used, and "Fast Echo" will improve that even more.

Otherwise, even with "Fast Echo" turned on, this effect will probably suck some major CPU cycles on most sound hardware. (V4)

AHIET_CHANNELINFO - Effect is a struct AHIEffChannelInfo, where
ahieci_Func is pointing to a hook that will be called with the following parameters:
A0 - (struct Hook *)
A2 - (struct AHIAudioCtrl *)
A1 - (struct AHIEffChannelInfo *)
ahieci_Channels must equal the current number of channels used.
ahieci_Offset is an array of ULONGs, which will be filled by AHI before the hook is called (the offset is specified in sample frames). The array must have at least ahieci_Channels elements.

This "effect" can be used to find out how far each channel has played. You must probably keep track of the other parameters yourself (like which sound is playing, it’s volume, balance and frequency etc) in order have meaningful usage of the information. (V4)

NOTE! To turn off an effect, call again with ahie_Effect OR:ed with AHIET_CANCEL. For example, it is NOT correct to disable the AHIET_MASTERVOLUME effect by setting ahiemv_Volume to 1.0!

It is important that you always turn off effects before you deallocate the audio hardware. Otherwise memory may be lost. It is safe to turn off an effect that has never been turned on in the first place.

Never count on that an effect will be available. For example,

```
       AHIET_OUTPUTBUFFER is impossible to implement with some sound
       cards.


RESULT
       An error code, defined in <devices/ahi.h>.


EXAMPLE


NOTES
       Unlike the other functions whose names begin with "AHI_Set", this
       function may NOT be called from an interrupt (or AHI Hook).

       Previous to V4, this call always returned AHIE_OK.


BUGS
       The idea of updating the source structure instead of allocating
       a new one that is passed the hook it pretty flawed. The reason is
       that AHI_SetEffect() originally could be called from interrupts,
       and memory allocation is not allowed from within interrupts.


SEE ALSO
       AHI_SetFreq(), AHI_SetSound(), AHI_SetVol(), AHI_LoadSound(),
       <devices/ahi.h>
```

1.16 ahi.device/AHI_SetFreq

```
ahi.device/AHI_SetFreq

NAME
       AHI_SetFreq -- set frequency for a channel


SYNOPSIS
       AHI_SetFreq( channel, freq, audioctrl, flags );
                    D0:16    D1    A2         D2

       void AHI_SetFreq( UWORD, ULONG, struct AHIAudioCtrl *, ULONG );


FUNCTION
       Sets the playback frequency for a channel.


INPUTS
       channel - The channel to set playback frequency for.
       freq - The playback frequency in Hertz. Can also be AHI_MIXFREQ,
           is the current mixing frequency (only usable with AHIST_INPUT
           sounds), or 0 to temporary stop the sound (it will restart at
           the same point when its frequency changed). Setting the frequency
```

```
            of an AHIST_INPUT sound is not supported, and the result is
            undefined.
        audioctrl - A pointer to an AHIAudioCtrl structure.
        flags - Only one flag is defined
            AHISF_IMM - Set this flag if this command should take effect
                immediately. If this bit is not set, the command will not
                take effect until the current sound is finished. MUST NOT
                be set if called from a SoundFunc. See the programming
                guidelines for more information about this flag.


RESULT


EXAMPLE


NOTES
        It is safe to call this function from an interrupt.


BUGS


SEE ALSO
        AHI_SetEffect(),  AHI_SetSound(), AHI_SetVol(), AHI_LoadSound()
```

1.17 ahi.device/AHI_SetSound

```
ahi.device/AHI_SetSound

NAME
        AHI_SetSound -- set what sound to play for a channel


SYNOPSIS
        AHI_SetSound( channel, sound, offset, length, audioctrl, flags );
                       D0:16   D1:16   D2      D3      A2         D4

        void AHI_SetSound( UWORD, UWORD, ULONG, LONG,
                           struct AHIAudioCtrl *, ULONG );


FUNCTION
        Sets a sound to be played on a channel.


INPUTS
        channel - The channel to set sound for.
        sound - Sound to be played, or AHI_NOSOUND to turn the channel off.
        offset - Only available if the sound type is AHIST_SAMPLE or
            AHIST_DYNAMICSAMPLE. Must be 0 otherwise.
            Specifies an offset (in samples) where the playback will begin.
            If you wish to play the whole sound, set offset to 0.
```

```
       length - Only available if the sound type is AHIST_SAMPLE or
           AHIST_DYNAMICSAMPLE. Must be 0 otherwise.
           Specifies how many samples that should be played. If you
           wish to play the whole sound forwards, set offset to 0 and length
           to either 0 or the length of the sample array. You may not set
           length to 0 if offset is not 0! To play a sound backwards, just
           set length to a negative number.
       audioctrl - A pointer to an AHIAudioCtrl structure.
       flags - Only one flag is defined
           AHISF_IMM - Set this flag if this command should take effect
               immediately. If this bit is not set, the command will not
               take effect until the current sound is finished. MUST NOT
               be set if called from a SoundFunc. See the programming
               guidelines for more information about this flag.


RESULT


EXAMPLE


NOTES
       It is safe to call this function from an interrupt.

       If offset or length is not zero, make sure you do not exceed the
       sample limits.


BUGS


SEE ALSO
       AHI_SetEffect(),  AHI_SetFreq(), AHI_SetVol(), AHI_LoadSound()
```

1.18 ahi.device/AHI_SetVol

```
ahi.device/AHI_SetVol

NAME
       AHI_SetVol -- set volume and stereo panning for a channel


SYNOPSIS
       AHI_SetVol( channel, volume, pan, audioctrl, flags );
                   D0:16    D1      D2   A2         D3

       void AHI_SetVol( UWORD, Fixed, sposition, struct AHIAudioCtrl *,
                        ULONG );


FUNCTION
       Changes the volume and stereo panning for a channel.
```

INPUTS

channel - The channel to set volume for.
volume - The desired volume. Fixed is a LONG fixed-point value with
16 bits to the left of the point and 16 to the right
(typedef LONG Fixed; from IFF-8SVX docs).
Maximum volume is 1.0 (0x10000L) and 0.0 (0x0L) will turn off
this channel. Note: The sound will continue to play, but you
wont hear it. To stop a sound completely, use AHI_SetSound().
Starting with V4 volume can also be negative, which tells AHI
to invert the samples before playing. Note that all drivers
may not be able to handle negative volume. In that case the
absolute volume will be used.
pan - The desired panning. sposition is the same as Fixed
(typedef Fixed sposition; from IFF-8SVX.PAN docs).
1.0 (0x10000L) means that the sound is panned all the way to
the right, 0.5 (0x8000L) means the sound is centered and 0.0
(0x0L) means that the sound is panned all the way to the left.
Try to set Pan to the 'correct' value even if you know it has no
effect. For example, if you know you use a mono mode, set pan to
0.5 even if it does not matter.
Starting with V4 pan can also be negative, which tells AHI to
use the surround speaker for this channel. Note that all drivers
may not be able to handle negative pan. In that case the absolute
pan will be used.
audioctrl - A pointer to an AHIAudioCtrl structure.
flags - Only one flag is defined
AHISF_IMM - Set this flag if this command should take effect
immediately. If this bit is not set, the command will not
take effect until the current sound is finished. MUST NOT
be set if called from a SoundFunc. See the programming
guidelines for more information about this flag.

RESULT

EXAMPLE

NOTES

It is safe to call this function from an interrupt.

Negative volume or negative pan may use more CPU time than positive.

Using both negative volume and negative pan will play the inverted
sound on the surround speaker.

BUGS

SEE ALSO

AHI_SetEffect(), AHI_SetFreq(), AHI_SetSound(), AHI_LoadSound()

1.19 ahi.device/AHI_UnloadSound

```
ahi.device/AHI_UnloadSound

NAME
       AHI_UnloadSound -- discard a sound


SYNOPSIS
       AHI_UnloadSound( sound, audioctrl );
                        D0:16  A2

       void AHI_UnloadSound( UWORD, struct AHIAudioCtrl * );


FUNCTION
       Tells 'ahi.device' that this sound will not be used anymore.


INPUTS
       sound - The ID of the sound to unload.
       audioctrl - A pointer to an AHIAudioCtrl structure.


RESULT


EXAMPLE


NOTES
       This call will not break a Forbid() state.


BUGS


SEE ALSO
       AHI_LoadSound()
```

1.20 ahi.device/CMD_FLUSH

```
ahi.device/CMD_FLUSH

NAME
       CMD_FLUSH -- Cancel all I/O requests (V4)


FUNCTION
       Aborts ALL current requests, both active and waiting, even
       other programs requests!
```

```
IO REQUEST INPUT
       io_Device       Preset by the call to OpenDevice().
       io_Unit         Preset by the call to OpenDevice().
       io_Command      CMD_FLUSH


IO REQUEST RESULT
       io_Error        0 for success, or an error code as defined in
                       <ahi/devices.h> and <exec/errors.h>.
       io_Actual       If io_Error is 0, number of requests actually
                       flushed.

       The other fields, except io_Device, io_Unit and io_Command, are
       trashed.


EXAMPLE


NOTES
       This command should only be used in very rare cases, like AHI
       system utilities. Never use this command in an application.


BUGS


SEE ALSO
       CMD_RESET, <ahi/devices.h>, <exec/errors.h>
```

1.21 ahi.device/CMD_READ

```
ahi.device/CMD_READ

NAME
       CMD_READ -- Read raw samples from audio input (V4)


FUNCTION
       Reads samples from the users prefered input to memory. The sample
       format and frequency will be converted on the fly.


IO REQUEST INPUT
       io_Device       Preset by the call to OpenDevice().
       io_Unit         Preset by the call to OpenDevice().
       io_Command      CMD_READ
       io_Data         Pointer to the buffer where the data should be put.
       io_Length       Number of bytes to read, must be a multiple of the
                       sample frame size (see ahir_Type).
       io_Offset       Set to 0 when you use for the first time or after
                       a delay.
       ahir_Type       The desired sample format, see <ahi/devices.h>.
       ahir_Frequency  The desired sample frequency in Hertz.
```

```
IO REQUEST RESULT
       io_Error        0 for success, or an error code as defined in
                       <ahi/devices.h> and <exec/errors.h>.
       io_Actual       If io_Error is 0, number of bytes actually
                       transferred.
       io_Offset       Updated to be used as input next time.

       The other fields, except io_Device, io_Unit and io_Command, are
       trashed.


EXAMPLE


NOTES


BUGS


SEE ALSO
       <ahi/devices.h>, <exec/errors.h>
```

1.22 ahi.device/CMD_RESET

```
ahi.device/CMD_RESET

NAME
       CMD_RESET -- Restore device to a known state (V4)


FUNCTION
       Aborts all current requests, even other programs requests
       (CMD_FLUSH), rereads the configuration file and resets the hardware
       to its initial state



IO REQUEST INPUT
       io_Device       Preset by the call to OpenDevice().
       io_Unit         Preset by the call to OpenDevice().
       io_Command      CMD_RESET


IO REQUEST RESULT
       io_Error        0 for success, or an error code as defined in
                       <ahi/devices.h> and <exec/errors.h>.

       The other fields, except io_Device, io_Unit and io_Command, are
       trashed.
```

```
EXAMPLE


NOTES
       This command should only be used in very rare cases, like AHI
       system utilities. Never use this command in an application.


BUGS


SEE ALSO
       CMD_FLUSH, <ahi/devices.h>, <exec/errors.h>
```

1.23 ahi.device/CMD_START

```
ahi.device/CMD_START

NAME
       CMD_START -- start device processing (like ^Q) (V4)


FUNCTION
       All CMD_WRITE's that has been sent to the device since CMD_STOP
       will be started at once, synchronized.


IO REQUEST INPUT
       io_Device       Preset by the call to OpenDevice().
       io_Unit         Preset by the call to OpenDevice().
       io_Command      CMD_START


IO REQUEST RESULT
       io_Error        0 for success, or an error code as defined in
                       <ahi/devices.h> and <exec/errors.h>.
       io_Actual       If io_Error is 0, number of requests actually
                       flushed.

       The other fields, except io_Device, io_Unit and io_Command, are
       trashed.


EXAMPLE


NOTES
       Unlike most (all?) other devices, CMD_STOP and CMD_START do nest in
       ahi.device.


BUGS
```

```
SEE ALSO
       CMD_STOP, <ahi/devices.h>, <exec/errors.h>
```

1.24 ahi.device/CMD_STOP

```
ahi.device/CMD_STOP

NAME
       CMD_STOP -- stop device processing (like ^S) (V4)


FUNCTION
       Stops all CMD_WRITE processing. All writes will be queued, and
       are not processed until CMD_START. This is useful for synchronizing
       two or more CMD_WRITE's.


IO REQUEST INPUT
       io_Device       Preset by the call to OpenDevice().
       io_Unit         Preset by the call to OpenDevice().
       io_Command      CMD_STOP


IO REQUEST RESULT
       io_Error        0 for success, or an error code as defined in
                       <ahi/devices.h> and <exec/errors.h>.
       io_Actual       If io_Error is 0, number of requests actually
                       flushed.

       The other fields, except io_Device, io_Unit and io_Command, are
       trashed.


EXAMPLE


NOTES
       This command affects ALL writes, even those sent by other
       applications. Make sure the code between CMD_STOP and CMD_START
       runs as fast as possible!

       Unlike most (all?) other devices, CMD_STOP and CMD_START do nest in
       ahi.device.


BUGS


SEE ALSO
       CMD_START, <ahi/devices.h>, <exec/errors.h>
```

1.25 ahi.device/CMD_WRITE

```
ahi.device/CMD_WRITE

NAME
       CMD_WRITE -- Write raw samples to audio output (V4)


FUNCTION
       Plays the samples to the users prefered audio output.


IO REQUEST INPUT
       io_Device       Preset by the call to OpenDevice().
       io_Unit         Preset by the call to OpenDevice().
       io_Command      CMD_WRITE
       io_Data         Pointer to the buffer of samples to be played.
       io_Length       Number of bytes to play, must be a multiple of the
                       sample frame size (see ahir_Type).
       io_Offset       Must be 0.
       ahir_Type       The desired sample format, see <ahi/devices.h>.
       ahir_Frequency  The desired sample frequency in Hertz.
       ahir_Volume     The desired volume. The range is 0 to 0x10000, where
                       0 means muted and 0x10000 (== 1.0) means full volume.
       ahir_Position   Defines the stereo balance. 0 is far left, 0x8000 is
                       center and 0x10000 is far right.
       ahir_Link       If non-zero, pointer to a previously sent AHIRequest
                       which this AHIRequest will be linked to. This
                       request will be delayed until the old one is
                       finished (used for double buffering). Must be set
                       to NULL if not used.


IO REQUEST RESULT
       io_Error        0 for success, or an error code as defined in
                       <ahi/devices.h> and <exec/errors.h>.
       io_Actual       If io_Error is 0, number of bytes actually
                       played.

       The other fields, except io_Device, io_Unit and io_Command, are
       trashed.


EXAMPLE


NOTES


BUGS
       32 bit samples are not allowed yet.


SEE ALSO
       <ahi/devices.h>, <exec/errors.h>
```

1.26 ahi.device/CloseDevice

```
ahi.device/CloseDevice

NAME
       CloseDevice -- Close the device


SYNOPSIS
       CloseDevice(ioRequest)
                   A1

       void CloseDevice(struct IORequest *);


FUNCTION
       This is an exec call that closes the device. Every OpenDevice()
       must be matched with a call to CloseDevice().

       The user must ensure that all outstanding IO Requests have been
       returned before closing the device.


INPUTS
       ioRequest - a pointer to the same struct AHIRequest that was used
           to open the device.


RESULT


EXAMPLE


NOTES


BUGS


SEE ALSO
      OpenDevice(), exec.library/CloseDevice()
```

1.27 ahi.device/NSCMD_DEVICEQUERY

```
ahi.device/NSCMD_DEVICEQUERY

NAME
       NSCMD_DEVICEQUERY -- Query the device for its capabilities (V4)


FUNCTION
       Fills an initialized NSDeviceQueryResult structure with
```

```
       information about the device.


IO REQUEST INPUT
       io_Device       Preset by the call to OpenDevice().
       io_Unit         Preset by the call to OpenDevice().
       io_Command      NSCMD_DEVICEQUERY
       io_Data         Pointer to the NSDeviceQueryResult structure,
                       initialized as follows:
                           DevQueryFormat - Set to 0
                           SizeAvailable  - Must be cleared.
                       It is probably good manners to clear all other
                       fields as well.
       io_Length       Size of the NSDeviceQueryResult structure.


IO REQUEST RESULT
       io_Error        0 for success, or an error code as defined in
                       <ahi/devices.h> and <exec/errors.h>.
       io_Actual       If io_Error is 0, the value in
                       NSDeviceQueryResult.SizeAvailable.

       The NSDeviceQueryResult structure now contains valid information.

       The other fields, except io_Device, io_Unit and io_Command, are
       trashed.


EXAMPLE


NOTES


BUGS


SEE ALSO
       <ahi/devices.h>, <exec/errors.h>
```

1.28 ahi.device/OpenDevice

```
ahi.device/OpenDevice

NAME
       OpenDevice -- Open the device


SYNOPSIS
       error = OpenDevice(AHINAME, unit, ioRequest, flags)
       D0                 A0       D0    A1         D1

       BYTE OpenDevice(STRPTR, ULONG, struct AHIRequest *, ULONG);
```

```
FUNCTION
       This is an exec call.  Exec will search for the ahi.device, and
       if found, will pass this call on to the device.


INPUTS
       AHINAME - pointer to the string "ahi.device".
       unit - Either AHI_DEFAULT_UNIT (0), AHI_NO_UNIT (255) or any other
           unit the user has requested, for example with a UNIT tooltype.
           AHI_NO_UNIT should be used when you're using the low-level
           API.
       ioRequest - a pointer to a struct AHIRequest, initialized by
           exec.library/CreateIORequest(). ahir_Version *must* be preset
           to the version you need!
       flags - There is only one flag defined, AHIDF_NOMODESCAN, which
           asks ahi.device not to build the audio mode database if not
           already initialized. It should not be used by applications
           without good reasons (AddAudioModes uses this flag).


RESULT
       error - Same as io_Error.
       io_Error - If the call succeeded, io_Error will be 0, else
           an error code as defined in <exec/errors.h> and
           <devices/ahi.h>.
       io_Device - A pointer to the device base, which can be used
           to call the functions the device provides.


EXAMPLE


NOTES


BUGS


SEE ALSO
      CloseDevice(), exec.library/OpenDevice(), <exec/errors.h>,
      <devices/ahi.h>.
```