

**ahi\_sub**

**COLLABORATORS**

	<i>TITLE :</i> ahi_sub		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 16, 2022	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>ahi_sub</b>	<b>1</b>
1.1	ahi_sub.guide . . . . .	1
1.2	[driver].audio/--background-- . . . . .	1
1.3	[driver].audio/AHISub_#? . . . . .	2
1.4	[driver].audio/AHISub_AllocAudio . . . . .	3
1.5	[driver].audio/AHISub_Disable . . . . .	5
1.6	[driver].audio/AHISub_Enable . . . . .	6
1.7	[driver].audio/AHISub_FreeAudio . . . . .	6
1.8	[driver].audio/AHISub_GetAttr . . . . .	7
1.9	[driver].audio/AHISub_HardwareControl . . . . .	9
1.10	[driver].audio/AHISub_Start . . . . .	10
1.11	[driver].audio/AHISub_Stop . . . . .	14
1.12	[driver].audio/AHISub_Update . . . . .	14

---

# Chapter 1

## ahi\_sub

### 1.1 ahi\_sub.guide

#### TABLE OF CONTENTS

[driver].audio/--background--  
[driver].audio/AHIsub\_#?  
[driver].audio/AHIsub\_AllocAudio  
[driver].audio/AHIsub\_Disable  
[driver].audio/AHIsub\_Enable  
[driver].audio/AHIsub\_FreeAudio  
[driver].audio/AHIsub\_GetAttr  
[driver].audio/AHIsub\_HardwareControl  
[driver].audio/AHIsub\_Start  
[driver].audio/AHIsub\_Stop  
[driver].audio/AHIsub\_Update

### 1.2 [driver].audio/--background--

[driver].audio/--background--

#### OVERVIEW

##### DRIVER VERSIONS

The lowest supported driver version is 2. If you use any feature

---

introduced in later versions of AHI, you should set the driver version to the same version as the features were introduced with. Example: You use `PreTimer()` and `PostTimer()`, and since these calls were added in V4 of `ahi.device`, your driver's version should be 4, too.

#### AUDIO ID NUMBERS

Just some notes about selecting ID numbers for different modes: It is up to the driver programmer to chose which modes should be available to the user. Take care when selecting.

The upper word is the hardware ID, and can only be allocated by Martin Blom <lcs@lysator.liu.se>. The lower word is free, but in order to allow enhancements, please only use bit 0 to 3 for modes! If your driver supports multiple sound cards, use bit 12-15 to select card (first one is 0). If your sound card has multiple AD/DA converters, you can use bit 8-11 to select them (the first should be 0).

Set the remaining bits to zero.

Use `AHI:Developer/Support/ScanAudioModes` to have a look at the modes currently available. Use `AHI:Developer/Support/sift` to make sure your mode descriptor file is a legal IFF file.

I do reserve the right to change the rules if I find them incorrect!

## 1.3 [driver].audio/AHIsb\_#?

[driver].audio/AHIsb\_#?

#### NAME

`AHIsb_SetEffect` -- Set effect.  
`AHIsb_SetFreq` -- Set frequency.  
`AHIsb_SetSound` -- Set sound.  
`AHIsb_SetVol` -- Set volume and stereo panning.  
`AHIsb_LoadSound` -- Prepare a sound for playback.  
`AHIsb_UnloadSound` -- Discard a sound.

#### SYNOPSIS

See functions in '`ahi.device`'.

#### IMPLEMENTATION

If  
`AHIsb_AllocAudio`  
 () did not return with bit `AHISB_MIXING` set,  
 all user calls to these function will be routed to the driver.

If  
`AHIsb_AllocAudio`  
 () did return with bit `AHISB_MIXING` set, the

calls will first be routed to the driver, and only handled by 'ahi.device' if the driver returned AHIS\_UNKNOWN. This way it is possible to add effects that the sound card handles on its own, like filter and echo effects.

For what each function does, see the autodocs for 'ahi.device'.

#### INPUTS

See functions in 'ahi.device'.

#### NOTES

See functions in 'ahi.device'.

#### SEE ALSO

ahi.device/AHI\_SetEffect(), ahi.device/AHI\_SetFreq(),  
 ahi.device/AHI\_SetSound(), ahi.device/AHI\_SetVol(),  
 ahi.device/AHI\_LoadSound(), ahi.device/AHI\_UnloadSound()

## 1.4 [driver].audio/AHISub\_AllocAudio

[driver].audio/AHISub\_AllocAudio

#### NAME

AHISub\_AllocAudio -- Allocates and initializes the audio hardware.

#### SYNOPSIS

```
result = AHISub_AllocAudio( tags, audioctrl);
D0                A1    A2
```

```
ULONG AHISub_AllocAudio( struct TagItem *, struct AHIAudioCtrlDrv * );
```

#### IMPLEMENTATION

Allocate and initialize the audio hardware. Decide if and how you wish to use the mixing routines provided by 'ahi.device', by looking in the AHIAudioCtrlDrv structure and parsing the tag list for tags you support.

- 1) Use mixing routines with timing:
  - You will need to be able to play any number of samples from about 80 up to 65535 with low overhead.
  - Update AudioCtrl->ahiac\_MixFreq to nearest value that your hardware supports.
  - Return AHISF\_MIXING|AHISF\_TIMING.
- 2) Use mixing routines without timing:
  - If the hardware can't play samples with any length, use this alternative and provide timing yourself. The buffer must take less than about 20 ms to play, preferable less than 10!

- Update AudioCtrl->ahiac\_MixFreq to nearest value that your hardware supports.
- Store the number of samples to mix each pass in AudioCtrl->ahiac\_BuffSamples.
- Return AHISF\_MIXING

Alternatively, you can use the first method and call the mixing hook several times in a row to fill up a buffer.

In that case,

AHISub\_GetAttr

(AHIDB\_MaxPlaySamples) should

return the size of the buffer plus AudioCtrl->ahiac\_MaxBuffSamples.

If the buffer is so large that it takes more than (approx.) 10 ms to play it for high sample frequencies,

AHISub\_GetAttr

(AHIDB\_Realtime)

should return FALSE.

### 3) Don't use mixing routines:

If your hardware can handle everything without using the CPU to mix the channels, you tell 'ahi.device' this by not setting either the AHISB\_MIXING or the AHISB\_TIMING bit.

If you can handle stereo output from the mixing routines, also set bit AHISB\_KNOWSTEREO.

If you can handle hifi (32 bit) output from the mixing routines, set bit AHISB\_KNOWHIFI.

If this driver can be used to record samples, set bit AHISB\_CANRECORD, too (regardless if you use the mixing routines in AHI or not).

If the sound card has hardware to do DSP effects, you can set the AHISB\_CANPOSTPROCESS bit. The output from the mixing routines will then be two separate buffers, one wet and one dry. You should then apply the Fx on the wet buffer, and post-mix the two buffers before you send the samples to the DAC. (V4)

## INPUTS

tags - pointer to a taglist.

audioctrl - pointer to an AHIAudioCtrlDrv structure.

## TAGS

The tags are from the audio database (AHIDB\_#? in <devices/ahi.h>), NOT the tag list the user called ahi.device/AHI\_AllocAudio() with.

## RESULT

Flags, defined in <libraries/ahi\_sub.h>.

## EXAMPLE

## NOTES

You don't have to clean up on failure,  
     AHISub\_FreeAudio

() will  
always be called.

BUGS

SEE ALSO

```
AHIsub_FreeAudio  
(),  
AHIsub_Start  
()
```

## 1.5 [driver].audio/AHIsub\_Disable

[driver].audio/AHIsub\_Disable

NAME

AHIsub\_Disable -- Temporary turn off audio interrupt/task

SYNOPSIS

```
AHIsub_Disable( audioctrl );  
                  A2
```

```
void AHIsub_Disable( struct AHIAudioCtrlDrv * );
```

IMPLEMENTATION

If you are lazy, then call `exec.library/Disable()`.  
If you are smart, only disable your own interrupt or task.

INPUTS

`audioctrl` - pointer to an `AHIAudioCtrlDrv` structure.

NOTES

This call should be guaranteed to preserve all registers.  
This call nests.

SEE ALSO

```
AHIsub_Enable  
(), exec.library/Disable()
```

---



## 1.6 [driver].audio/AHIsb\_Enable

[driver].audio/AHIsb\_Enable

### NAME

AHIsb\_Enable -- Turn on audio interrupt/task

### SYNOPSIS

```
AHIsb_Enable( audioctrl );  
            A2  
  
void AHIsb_Enable( struct AHIAudioCtrlDrv * );
```

### IMPLEMENTATION

If you are lazy, then call `exec.library/Enable()`.  
If you are smart, only enable your own interrupt or task.

### INPUTS

`audioctrl` - pointer to an `AHIAudioCtrlDrv` structure.

### NOTES

This call should be guaranteed to preserve all registers.  
This call nests.

### SEE ALSO

`AHIsb_Disable`  
`()`, `exec.library/Enable()`

## 1.7 [driver].audio/AHIsb\_FreeAudio

[driver].audio/AHIsb\_FreeAudio

### NAME

AHIsb\_FreeAudio -- Deallocates the audio hardware.

### SYNOPSIS

```
AHIsb_FreeAudio( audioctrl );  
                A2  
  
void AHIsb_FreeAudio( struct AHIAudioCtrlDrv * );
```

### IMPLEMENTATION

Deallocate the audio hardware and other resources allocated in

---

```

    AHISub_AllocAudio
    ().
    AHISub_Stop
    () will always be called by
    'ahi.device' before this call is made.

```

#### INPUTS

audioctrl - pointer to an AHIAudioCtrlDrv structure.

#### NOTES

It must be safe to call this routine even if  
 AHISub\_AllocAudio  
 ()  
 was never called, failed or called more than once.

#### SEE ALSO

```

    AHISub_AllocAudio
    ()

```

## 1.8 [driver].audio/AHISub\_GetAttr

[driver].audio/AHISub\_GetAttr

#### NAME

AHISub\_GetAttr -- Returns information about audio modes or driver

#### SYNOPSIS

```

    AHISub_GetAttr( attribute, argument, default, taglist, audioctrl );
    D0                D0                D1                D2                A1                A2

    LONG AHISub_GetAttr( ULONG, LONG, LONG, struct TagItem *,
                        struct AHIAudioCtrlDrv * );

```

#### IMPLEMENTATION

Return the attribute based on a tag list and an AHIAudioCtrlDrv structure, which are the same that will be passed to

```

    AHISub_AllocAudio
    () by 'ahi.device'. If the attribute is
    unknown to you, return the default.

```

#### INPUTS

attribute - Is really a Tag and can be one of the following:  
 AHIDB\_Bits - Return how many output bits the tag list will result in.  
 AHIDB\_MaxChannels - Return the resulting number of channels.

AHIDB\_Frequencies - Return how many mixing/sampling frequencies you support

AHIDB\_Frequency - Return the argument:th frequency  
 Example: You support 3 frequencies 32, 44.1 and 48 kHz.  
 If argument is 1, return 44100.

AHIDB\_Index - Return the index which gives the frequency closest to argument.  
 Example: You support 3 frequencies 32, 44.1 and 48 kHz.  
 If argument is 40000, return 1 (=> 44100).

AHIDB\_Author - Return pointer to name of driver author:  
 "Martin 'Leviticus' Blom"

AHIDB\_Copyright - Return pointer to copyright notice, including the '@' character: "@ 1996 Martin Blom" or "Public Domain"

AHIDB\_Version - Return pointer version string, normal Amiga format: "paula 1.5 (18.2.96)\r\n"

AHIDB\_Annotation - Return pointer to an annotation string, which can be several lines.

AHIDB\_Record - Are you a sampler, too? Return TRUE or FALSE.

AHIDB\_FullDuplex - Return TRUE or FALSE.

AHIDB\_Realtime - Return TRUE or FALSE.

AHIDB\_MaxPlaySamples - Normally, return the default. See

AHISub\_AllocAudio  
 (), section 2.

AHIDB\_MaxRecordSamples - Return the size of the buffer you fill when recording.

The following are associated with  
 AHISub\_HardwareControl  
 () and are  
 new for V2.

AHIDB\_MinMonitorVolume

AHIDB\_MaxMonitorVolume - Return the lower/upper limit for AHIC\_MonitorVolume. If unsupported but always 1.0, return 1.0 for both.

AHIDB\_MinInputGain

AHIDB\_MaxInputGain - Return the lower/upper limit for AHIC\_InputGain. If unsupported but always 1.0, return 1.0 for both.

AHIDB\_MinOutputVolume

AHIDB\_MaxOutputVolume - Return the lower/upper limit for AHIC\_OutputVolume.

AHIDB\_Inputs - Return how many inputs you have.

AHIDB\_Input - Return a short string describing the argument:th input. Number 0 should be the default one. Example strings can be "Line 1", "Mic", "Optical" or whatever.

AHIDB\_Outputs - Return how many outputs you have.

AHIDB\_Output - Return a short string describing the argument:th output. Number 0 should be the default one. Example strings can be "Line 1", "Headphone", "Optical" or whatever.

argument - extra info for some attributes.

default - What you should return for unknown attributes.

taglist - Pointer to a tag list that eventually will be fed to

AHISub\_AllocAudio  
 (), or NULL.

audioctrl - Pointer to an AHIAudioCtrlDrv structure that eventually

will be fed to  
 AHISub\_AllocAudio  
 (), or NULL.

#### NOTES

#### SEE ALSO

AHISub\_AllocAudio  
 (),  
 AHISub\_HardwareControl  
 (),  
 ahi.device/AHI\_GetAudioAttrsA()

## 1.9 [driver].audio/AHISub\_HardwareControl

[driver].audio/AHISub\_HardwareControl

#### NAME

AHISub\_HardwareControl -- Modify sound card settings

#### SYNOPSIS

```
AHISub_HardwareControl( attribute, argument, audioctrl );
D0                      D0          D1          A2

LONG AHISub_HardwareControl( ULONG, LONG, struct AHIAudioCtrlDrv * );
```

#### IMPLEMENTATION

Set or return the state of a particular hardware component. AHI uses

AHISub\_GetAttr  
 () to supply the user with limits and what tags are  
 available.

#### INPUTS

attribute - Is really a Tag and can be one of the following:  
 AHIC\_MonitorVolume - Set the input monitor volume to argument.  
 AHIC\_MonitorVolume\_Query - Return the current input monitor  
 volume (argument is ignored).  
  
 AHIC\_InputGain - Set the input gain to argument. (V2)  
 AHIC\_InputGain\_Query (V2)  
  
 AHIC\_OutputVolume - Set the output volume to argument. (V2)  
 AHIC\_OutputVolume\_Query (V2)  
  
 AHIC\_Input - Use the argument:th input source (default is 0). (V2)  
 AHIC\_Input\_Query (V2)

AHIC\_Output - Use the argument:th output destination (default is 0). (V2)  
 AHIC\_Output\_Query (V2)

argument - What value attribute should be set to.  
 audioctrl - Pointer to an AHIAudioCtrlDrv structure.

**RESULT**

Return the state of selected attribute. If you were asked to set something, return TRUE. If attribute is unknown to you or unsupported, return FALSE.

**NOTES**

This call must be safe from interrupts.

**SEE ALSO**

ahi.device/AHI\_ControlAudioA(),  
 AHISub\_GetAttr  
 ()

**1.10 [driver].audio/AHISub\_Start**

[driver].audio/AHISub\_Start

**NAME**

AHISub\_Start -- Starts playback or recording

**SYNOPSIS**

```
error = AHISub_Start( flags, audioctrl );
D0          D0          A2

ULONG AHISub_Start(ULONG, struct AHIAudioCtrlDrv * );
```

**IMPLEMENTATION**

What to do depends what you returned in  
 AHISub\_AllocAudio  
 ().

- \* First, assume bit AHISB\_PLAY in flags is set. This means that you should begin playback.

-

```
AHISub_AllocAudio  

() returned AHISF_MIXING|AHISF_TIMING:
```

- A) Allocate a mixing buffer of ahiac\_BuffSize bytes. The buffer must be long aligned!

- B) Create/start an interrupt or task that will do 1-6 over and over again until  
     AHISub\_Stop  
     () is called. Note that it is not a good idea to do the actual mixing and conversion in a real hardware interrupt. Signal a task or create a Software Interrupt to do the number crunching.
- 1) Call the user Hook ahiac\_PlayerFunc with the following parameters:  
     A0 - (struct Hook \*)  
     A2 - (struct AHIAudioCtrlDrv \*)  
     A1 - Set to NULL.
- 2) [Call the ahiac\_PreTimer function. If it returns TRUE (Z will be cleared so you don't have to test d0), skip step 3 and 4. This is used to avoid overloading the CPU. This step is optional. A2 is assumed to point to struct AHIAudioCtrlDrv. All registers except d0 are preserved. (V4)]
- 3) Call the mixing Hook (ahiac\_MixerFunc) with the following parameters:  
     A0 - (struct Hook \*)                     - The Hook itself  
     A2 - (struct AHIAudioCtrlDrv \*)  
     A1 - (WORD \*[])                         - The mixing buffer.
- Note that ahiac\_MixerFunc preserves ALL registers.  
 The user Hook ahiac\_SoundFunc will be called by the mixing routine when a sample have been processed, so you don't have to worry about that.  
 How the buffer will be filled is indicated by ahiac\_Flags.  
 It is always filled with signed 16-bit (32 bit if AHIAACB\_HIFI in ahiac\_Flags is set) words, even if playback is 8 bit. If AHIDBB\_STEREO is set (in ahiac\_Flags), data for left and right channel are interleaved:  
     1st sample left channel,  
     1st sample right channel,  
     2nd sample left channel,  
     .../  
     ahiac\_BuffSamples:th sample left channel,  
     ahiac\_BuffSamples:th sample right channel.
- If AHIDBB\_STEREO is cleared, the mono data is stored:  
     1st sample,  
     2nd sample,  
     .../  
     ahiac\_BuffSamples:th sample.
- Note that neither AHIAACB\_STEREO nor AHIAACB\_HIFI will be set if you didn't report that you understand these flags when AHI\_AllocAudio() was called.
- For AHI V2, the type of buffer is also available in ahiac\_BuffType. It is suggested that you use this value instead. ahiac\_BuffType can be one of AHIST\_M16S, AHIST\_S16S, AHIST\_M32S and AHIST\_S32S.
- 4) Convert the buffer if needed and feed it to the audio hardware. Note that you may have to clear CPU caches if you are using DMA to play the buffer, and the buffer is not allocated in non-cachable RAM.
-

- 5) [Call the ahiac\_PostTimer function. A2 is assumed to point to struct AHIAudioCtrlDrv. All registers are preserved. (V4)]
- 6) Wait until the whole buffer has been played, then repeat.

Use double buffering if possible!

You may DECREASE ahiac\_BuffSamples slightly, for example to force an even number of samples to be mixed. By doing this you will make ahiac\_PlayerFunc to be called at wrong frequency so be careful! Even if ahiac\_BuffSamples is defined ULONG, it will never be greater than 65535.

ahiac\_BuffSize is the largest size of the mixing buffer that will be needed until

```
AHIsub_Stop
() is called.
```

ahiac\_MaxBuffSamples is the maximum number of samples that will be mixed (until

```
AHIsub_Stop
() is called). You can use this value if you
```

need to allocate DMA buffers.

ahiac\_MinBuffSamples is the minimum number of samples that will be mixed. Most drivers will ignore it.

If

```
AHIsub_AllocAudio
() returned with the AHISB_CANPOSTPROCESS bit set,
```

ahiac\_BuffSize is large enough to hold two buffers. The mixing buffer will be filled with the wet buffer first, immediately followed by the dry buffer. I.e., ahiac\_BuffSamples sample frames wet data, then ahiac\_BuffSamples sample frames dry data. The DSP fx should only be applied to the wet buffer, and the two buffers should then be added together. (V4)

- If

```
AHIsub_AllocAudio
() returned AHISF_MIXING, do as described above,
```

except calling ahiac\_PlayerFunc. ahiac\_PlayerFunc should be called ahiac\_PlayerFreq times per second, clocked by timers on your sound card or by using 'realtime.library'. No other Amiga resources may be used for timing (like direct CIA timers).

ahiac\_MinBuffSamples and ahiac\_MaxBuffSamples are undefined if

```
AHIsub_AllocAudio
() returned AHISF_MIXING (AHISB_TIMING bit not set).
```

- If

```
AHIsub_AllocAudio
() returned with neither the AHISB_MIXING nor
```

the AHISB\_TIMING bit set, then just start playback. Don't forget to call ahiac\_PlayerFunc ahiac\_PlayerFreq times per second. Only your own timing hardware or 'realtime.library' may be used. Note that ahiac\_MixerFunc, ahiac\_BuffSamples, ahiac\_MinBuffSamples, ahiac\_MaxBuffSamples and ahiac\_BuffSize are undefined. ahiac\_MixFreq

is the frequency the user wants to use for recording, if you support that.

- \* Second, assume bit AHISB\_RECORD in flags is set. This means that you should start to sample. Create an interrupt or task that does the following:

Allocate a buffer (you chose size, but try to keep it reasonable small to avoid delays - it is suggested that RecordFunc is called at least 4 times/second for the lower sampling rate, and more often for higher rates), and fill it with the sampled data. The buffer must be long aligned, and its size must be evenly divisible by four. The format should always be AHIST\_S16S (even with 8 bit mono samplers), which means:

```

1st sample left channel,
1st sample right channel (same as prev. if mono),
2nd sample left channel,
... etc.

```

Each sample is a signed word (WORD). The sample rate should be equal to the mixing rate.

Call the ahiac\_SamplerFunc Hook with the following parameters:

```

A0 - (struct Hook *)           - The Hook itself
A2 - (struct AHIAudioCtrlDrv *)
A1 - (struct AHIREcordMessage *)

```

The message should be filled as follows:

```

ahirm_Type - Set to AHIST_S16S.
ahirm_Buffer - A pointer to the filled buffer.
ahirm_Samples - How many sample frames stored.

```

You must not destroy the buffer until next time the Hook is called.

Repeat until

```

    AHISub_Stop
    () is called.

```

- \* Note that both bits may be set when this function is called.

#### INPUTS

flags - See <libraries/ahi\_sub.h>.  
 audioctrl - pointer to an AHIAudioCtrlDrv structure.

#### RESULT

Returns AHIE\_OK if successful, else an error code as defined in <devices/ahi.h>.

```

    AHISub_Stop
    () will always be called, even
if this call failed.

```

#### NOTES

The driver must be able to handle multiple calls to this routine without preceding calls to

```

    AHISub_Stop
    ().

```



SEE ALSO

```
AHIsb_Update
(),
AHIsb_Stop
()
```

## 1.11 [driver].audio/AHIsb\_Stop

[driver].audio/AHIsb\_Stop

NAME

AHIsb\_Stop -- Stops playback.

SYNOPSIS

```
AHIsb_Stop( flags, audioctrl );
           D0      A2

void AHIsb_Stop( ULONG, struct AHIAudioCtrlDrv * );
```

IMPLEMENTATION

Stop playback and/or recording, remove all resources allocated by

```
AHIsb_Start
().
```

INPUTS

flags - See <libraries/ahi\_sub.h>.  
audioctrl - pointer to an AHIAudioCtrlDrv structure.

NOTES

It must be safe to call this routine even if  
AHIsb\_Start  
() was never  
called, failed or called more than once.

SEE ALSO

```
AHIsb_Start
()
```

## 1.12 [driver].audio/AHIsb\_Update

---

[driver].audio/AHIsub\_Update

**NAME**

AHIsub\_Update -- Update some variables

**SYNOPSIS**

```
AHIsub_Update( flags, audioctrl );  
                D0      A2
```

```
void AHIsub_Update(ULONG, struct AHIAudioCtrlDrv * );
```

**IMPLEMENTATION**

All you have to do is to update some variables:

Mixing & timing: ahiac\_PlayerFunc, ahiac\_MixerFunc, ahiac\_SamplerFunc, ahiac\_BuffSamples (and perhaps ahiac\_PlayerFreq if you use it).

Mixing only: ahiac\_PlayerFunc, ahiac\_MixerFunc, ahiac\_SamplerFunc and ahiac\_PlayerFreq.

Nothing: ahiac\_PlayerFunc, ahiac\_SamplerFunc and ahiac\_PlayerFreq.

**INPUTS**

flags - Currently no flags defined.

audioctrl - pointer to an AHIAudioCtrlDrv structure.

**RESULT****NOTES**

This call must be safe from interrupts.

**SEE ALSO**

AHIsub\_Start  
( )

---