Technical Document

AUDIO API (RIV101)

ECCN EAR99

Document Number:	88_02_03_00218
Version:	6.5
Status:	Approved
Approval Authority:	
Creation Date:	2004-Mar-01
Last changed:	2007-May-14 by A0876504
File Name:	88_02_03_00218_RIV101_audio_api.doc

Important Notice

Texas Instruments Incorporated and/or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and/or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and/or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and/or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and/or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
2001-Jun-11	F. Mazard	Christian Livadiotti	1.1	Approved	1
2001-Jun-20	F. Mazard	Christian Livadiotti	1.2	Approved	2
2001-Jul-13	F. Mazard	Christian Livadiotti	1.3	Approved	3
2001-Jul -20	S.Gerthoux	Francois Mazard	1.4	Approved	4
2001-Aug-15	S.Gerthoux	Francois Mazard	1.5	Approved	5
2001-Nov-19	F. Mazard	Christian Livadiotti	1.6	Approved	6
2001-Dec-14	F.Mazard	Christian Livadiotti	1.7	Approved	7
2002-Jan-31	F.Mazard	Christian Livadiotti	1.8	Approved	8

2002-Mar-29	S.Gerthoux	Francois Mazard	1.9	Approved	9
2002-Apr-15	S.Gerthoux	Francois Mazard	2.0	Approved	10
2003-Mar-17	F. Turgis	Stephanie Gerthoux	2.1	Approved	11
2003-Apr-14	F. Turgis	Gerard Cauvy	2.2	Approved	12
2003-May-05	F. Turgis	S. Gerthoux	2.3	Approved	13
2003-Aug-05	F. Turgis	S. Gerthoux	2.4	Approved	14
2003-Oct-20	F. Turgis	J. Longchamp	2.5	Approved	15
2003-Oct-28	F. Turgis	S. Gerthoux	2.6	Approved	16
2003-Nov-14	J. Longchamp	S. Gerthoux	2.7	Approved	17
2004-Jan-01	F.Olivero	S. Gerthoux S. Guiriec	2.8	Approved	18
2004-Mar-01	S.Levieil (Gerthoux)		2.9		19
2004-Mar-02	F.Olivero		3.0		20
2004-Aug-02	S.Levieil		4.0		21
2004-Sep-06	S.Levieil		5.0		22
2004-Sep-08	S.Levieil		5.1		23
2004-Oct-18	S.Levieil		5.2		24
2004-Dec-14	S.Levieil		5.3		25
2005-Jan-05	S.Levieil		5.4		26
2005-Mar-18	S.Levieil		6.0		27
2005-may-24	S.Levieil		6.1		28
2005-Oct-01	Ranga Ramanujam S		6.2		29
2005-Oct-21	Vidhya Krishnamoorthy		6.3		30
2006-Mar-27	Srinath Ananthaswamy	Suyog Moogi	6.4		31
2007-Apr-30	Ranga Ramanujam S		6.5		32

Notes:

1. Initial version
2. Add two modes for the melody E1: a game mode and a normal mode. Add a parameter in the melody E1 stop function. Add some restriction of the use of melody E1. Change the melody parameter structure.
3. Migration from the Bluetooth database to the Riviera database:
   change T_RIV_RETURN into T_RV_RETURN.
   change T_RIV_HDR into T_RV_HDR.
4. Update of the Voice memorization
   Added the recorded_size parameter in the AUDIO_VM_RECORD_STATUS_MSG.
   Added parameters definitions (memo_size)
   Change the file name of the melody "char *memo_name" into
   "char memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];"
5. Add a restriction of use for the voice memorization recording task :
   All directories included in the pathname must be declared before.
6. Align the specification with the Riviera 1.6, Add in the restriction of use section that the incompatibilities between the audio features are managed by the audio entity.
7. Add a chapter about the Modem-Audio incompatibilities.
   Update the Speech Recognition API with more information and align the parameters of speech recognition API functions with the software.
8. Change the minimum duration of a keybeep and a tone from 1 ms to 20 ms.
   Update the audio mode configuration features.
9. Add the Audio tasks compatibilities
   Add the Melody E2 API and the Audio melody E2 tools
   Remove "Audio Information" of the API
10. Add Melody E2 incompatibilities
11. Integration of AMR play/record feature
12. Integration of TTY feature
13. Integration of New AEC
14. Added new VM AMR APIs functions + small correction in audio_amr_play_from_ram/ffs_start
15. Added SYREN audio mode: HEADSET DIFFERENTIAL input, HANDFREE 8OHM output, high pass DL filter bypass, STEREO AUDIO output modes, STEREO AUDIO volume + a few corrections.
16. Update compatibility matrix for DSP codes 34, 35, 36

17. Added new interface when using Perseus2 with a non-TI audio ABB + Audio on/off
18. added TCS 3.1 enhanced audio feature (Limiter, IIR filter, ANR) in audio mode configuration
19. Update de E2 Melody: 16 tones in parallel are supported.
20. Added echo suppressor feature in audio mode configuration
21. Added MP3 feature + small correction in section 9.2.1.4.2 (output gain parameter) + small correction in section 9.2.1.4.2 (Number of IIR blocks)
22. Added new parameter in MP3 structure and added a new parameter in audio_mp3_stop function
23. Change the ES parameters
24. Update the ANR values
25. Add some comments in the AEC, ANR, IIR, ES and Limiter parameters
26. IIR Minimum number of blocks changed
27. Added AAC feature
28. Change the characteristics function of the signal wanted at the output of the limiter (section 9.2.1.4.1)
29. Added TRITON related audio mode details
30. Update chapter - Audio Mode Configuration for TCS3.2
31. Removed older DSP compatibility matrices, Removed SR chapter, Removed Melo E2, Changed references, Section numbers
32. File path maximum size for all media changed to 80 characters and added ROM39 compatibility table in Section 3.1. Added MP3 Progress Bar, Forward and Rewind implementation and AAC Progress Bar implementation. Added descriptions on RFS support

Table of Contents

List of Figures and Tables

List of References

[1] Herc815 E2 melody generator Ver2.1

[2] S892 AEC interface and tuning

[3] L1D_AS120 IIR output filter overview

[4] L1D_AS127-1 IIR - Design tool user’s guide

[5] L1D_AS070_1 Limiter overview

[6] L1D_AS111-1 ANR overview

[7] L1D_AS260 Echo suppressor overview

[8] ITU-T P.340 Transmission characteristics of hands-free telephones

1 Introduction

This document provides an interface specification of the AUDIO SW entity. The purpose of the AUDIO SW entity is to provide an abstraction layer to SW developers in order to access the audio services available on the platform.

Voice Memo, Melody generation, Key Beep generation, Tones generation, audio mode configuration and speaker volume are part of the services provided by the AUDIO SW entity.

2 Overview

2.1 Generality

The AUDIO services provided by the AUDIO SW entity can be accessed by any SW entity running on the mobile. Several SW entities can use the audio services at the same time. Note that for commodity reasons, the SW entity using the AUDIO services will be called MMI in the rest of the document.

All the services provided by the audio entity can be accessed via direct function call. These functions are listed in this document. The AUDIO SW entity use the return mechanism defined in the Riviera Environment to provide information back to the MMI.

2.2 Return Mechanism

All the functions return an immediate value, providing information on the success or the failure of the function call. In some (most of the) cases, extra processing time is needed to perform the action requested when calling the function. In this case, the function is exit and later on, one or several EVENTs are sent back by the AUDIO SW entity. Note that for commodity, all the events are always mentioned in upper case.

The AUDIO SW entity use the EVENT format and the return path method defined in Riviera Environment. Basically, in order to send information back, the AUDIO SW entity sends EVENTs to the MMI. An event is a buffer, with a header, common to any EVENT, and a custom field related to the EVENT. The header is a C structure, containing the *opcode* field. This field contains the unique opcode of the EVENT and is the only way to know which kind of EVENT has been received. Based on this value, the MMI can re-cast the buffer and access to custom information related to the EVENT.

MMI have two ways to get access to the EVENTS:
Call back functions or message posted in its mailbox.

A call back function is a function name, provided by MMI as a parameter and which will be called by the AUDIO SW when a EVENT is available. When a callback function is defined, it is always the callback function mechanism that is used to return EVENTs to the MMI.
But for more efficient implementation, it also possible to directly post a message into the MMI mailbox. In this case, the task id and mailbox id of the MMI must be provided to the AUDIO SW entity. That implies that the MMI is a Riviera SW entity.

For every audio service, the MMI can define which return mechanism should be used. For that purpose, it must provide a *return_path*. The generic *return_path* type is a C structure, defined as:

```
/* unique ADDRess IDentifier of a SWE */
typedef UINT16 T_RVF_ADDR_ID;

typedef struct
{
  T_RVF_ADDR_ID addr_id;
  void         (*callback_func)(void *);
```

```
} T_RV_RETURN;
```

2.3 RFS Support

Audio module gives RFS API support which can be controlled compile time using AS_RFS_API flag. When AS_RFS_API flag is 1, Audio module supports RFS APIs instead of FFS APIs and this has an impact on the signature of the Audio APIs exposed. All the filenames which were character arrays in FFS based implementation are T_WCHAR arrays in RFS based implementation. For example T_AUDIO_MELODY_E1_PARAMETER with AS_RFS_API flag 0 is,

- **T_AUDIO_MELODY_E1_PARAMETER**

Specifies the characteristic of the melody to start.

```
typedef struct {
  char   melody_name[AUDIO_MELODY_PATH_NAME_MAX_SIZE];
  // File name of the melody
  BOOL   loopback;    // the melody is played indefinitely
  BOOL   melody_mode; // mode of the melody
}T_AUDIO_MELODY_E1_PARAMETER;
```

and when AS_RFS_API flag 1 is,

- **T_AUDIO_MELODY_E1_PARAMETER**

Specifies the characteristic of the melody to start.

```
typedef struct {
  T_WCHAR   melody_name[AUDIO_MELODY_PATH_NAME_MAX_SIZE];
  // File name of the melody
  BOOL   loopback;    // the melody is played indefinitely
  BOOL   melody_mode; // mode of the melody
}T_AUDIO_MELODY_E1_PARAMETER;
```

Where, T_WCHAR is UINT16.

This change is applicable to all the APIs that has filename as parameter. This will have impact on the task stack usage and the partition memory usage, as the filename size is doubled (with T_WCHAR).

3 Audio-Modem incompatibilities

Due to the share of the CPU load and of the memory in DSP, some audio task can't run with certain modem state. For instance, the audio MCU software doesn't manage this constraint. Therefore any user of the audio services described in this document must follow the audio-modem incompatibilities described in the table below.
In case of incompatibilities, the user of the audio must stop as soon as possible the audio before to enter to a modem state incompatible with the current audio task.

3.1 DSP code 39

Feature/Modem	ROM3900												
	GSM		Dedicated non AMR					Dedicated AMR		GPRS		GSM->GPRS	GP>G
	Idle	SMS	FACCH	Ringer	Speech	TCH/Data	IDS	Ringer	Speech	Idle	Transfer		
Keybeep	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tones	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Melody_E1	✓	✓	✓	✓				✓		✓	✓	✓	✓
Voice Memo	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
AMR MMS(*)	✓	✓	✓	✓				✓		✓	✓	✓	✓

Audio Mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
FIR (UL)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AEC/ES (UL) (*)					✓				✓				
ANR (UL) (*)					✓				✓				
AGC (UL/DL)					✓				✓				
IIR/FIR (DL)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DRC (DL)					✓				✓				
AAC(*)	✓	✓		✓				✓		✓	✓	✓	✓
MP3(*)	✓	✓		✓				✓		✓	✓	✓	✓
MP3(*)+SAS_NO_IIR	✓	✓		✓				✓		✓	✓	✓	✓

4 Audio task compatibilities

The sheets below show the compatibility of all audio tasks with another audio task.
These compatibilities are managed by the Riviera Audio incompatibilities manager.

4.1 DSP code 39

Feature/Feature	ROM CODE 39									
	Keybeep	Tones	Melody E1	Voice Memo	AMR MMS	FIR/IIR	AEC/ES/ ANR/AGC/DRC	Audio Mode	AAC	MP3
Keybeep		✓	✓	✓	✓	✓	✓	✓	✓	✓
Tones	✓			✓	✓	✓	✓	✓	✓	✓
Melody E1	✓		(*)			✓	✓	✓		
Voice Memo	✓	✓				✓	✓	✓		
AMR MMS	✓	✓				✓		✓		
FIR/IIR	✓	✓	✓	✓	✓		✓	✓	✓	✓
AEC/ES/ ANR/AGC/DRC	✓	✓	✓	✓		✓		✓		
Audio Mode	✓	✓	✓	✓	✓	✓	✓		✓	✓
AAC(**)	✓	✓				✓		✓		
MP3(**)	✓	✓				✓		✓		

(*): Second Melody E1 should be started during (and not at beginning of) first Melody E1, it corresponds to “game mode”
(**): MP3 & AAC support depends upon HW configuration and Business agreement.

5 Keybeep Generation

This chapter describes how to generate one key beep, using the AUDIO SW entity service.

5.1 audio_keybeep_start

```
T_AUDIO_RET audio_keybeep_start (T_AUDIO_KEYBEEP_PARAMETER parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to initiate a key beep generation and DTMF generation. The key beep is the generation of two simultaneous sine waves.

Parameters

- **T_AUDIO_KEYBEEP_PARAMETER**

Specifies the characteristic of the keybeep to start.

```
typedef struct {
  UINT16       frequency_beep[2]; // Frequency of the 2 beeps
  INT8         amplitude_beep[2]; // Amplitude of the 2 beeps
  UINT16       duration;
}T_AUDIO_KEYBEEP_PARAMETER;
```

Below the detail of each parameters:

frequency_beep[2] Specifies the frequency of the beeps 1 and 2 in 1 Hz unit. Note the range is [0…2000] Hz. If the frequency value is equal to NO_BEEP, the beep isn’t generated.
amplitude_beep[2] Specifies the amplitude of the beeps 1 and 2 in 1 dB unit. Note the range is [-48…0] dB.
duration Specifies the duration of the key beep in ms. Note this duration can’t be equal to 20ms.

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	An error is occurred during the execution of this function

Event Return

- **AUDIO_KEYBEEP_STATUS_MSG**

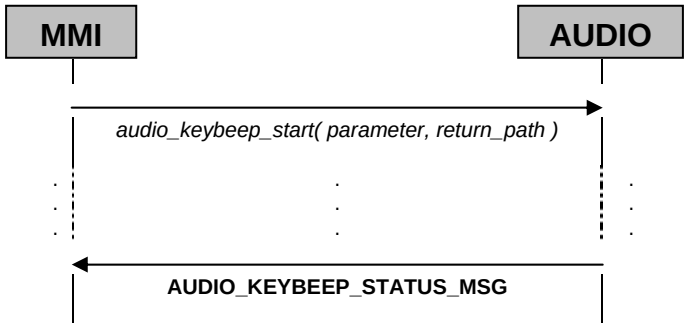
This event is the status send at the end of the keybeep generation or if an error occurred.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
}T_AUDIO_KEYBEEP_STATUS;
```

The possible values of *status* are:

value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	The audio features was not successfully executed

Process flow



5.2 audio_keybeep_stop

```
T_AUDIO_RET audio_keybeep_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the key beep generation.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_keybeep_start*.

Event Return

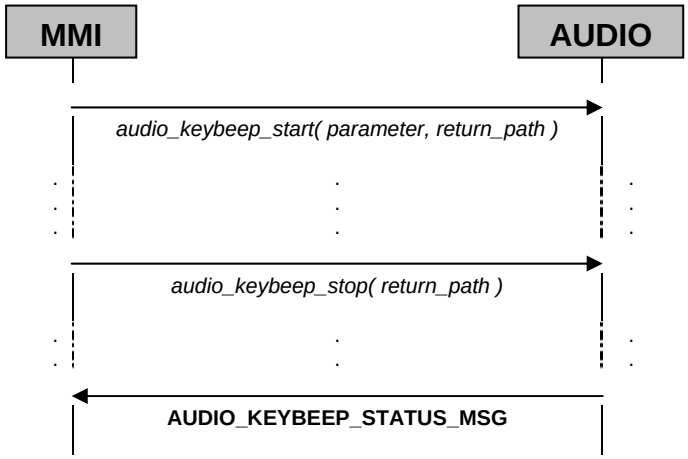
- **AUDIO_KEYBEEP_STATUS_MSG**

C.f. API function *audio_keybeep_start*.

Current restriction of use

Refer to section 4 on Audio Task Incompatibilities

Process flow



6 Tones Generation

This chapter describes how to generate the tones, using the AUDIO SW entity service.

6.1 audio_tones_start

```
T_AUDIO_RET audio_tones_start (T_AUDIO_TONES_PARAMETER *p_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to initiate the tones generation. The tones are the generation of up to three scheduled sine waves.

Parameters

- **T_AUDIO_TONES_PARAMETER**

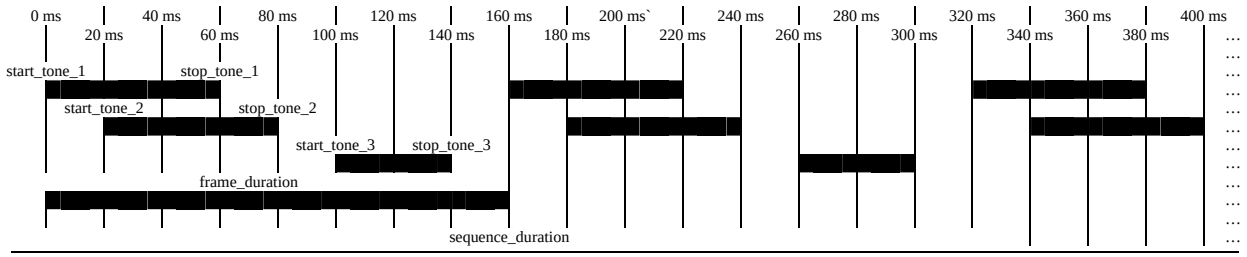
Specifies the characteristic of the keybeep to start.

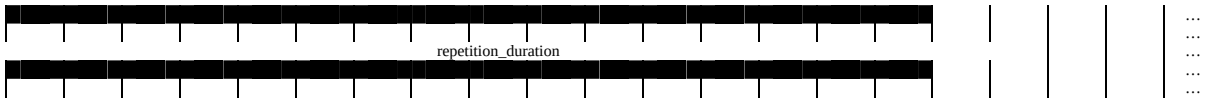
```
typedef struct {
  T_AUDIO_TONE_DESC   tones[3];          // Description of the  3 tones
  UINT16              frame_duration;    // duration of the tones frame
  UINT16              sequence_duration; // duration of the sequence
  UINT16              period_duration;   // duration of the period
  UINT16
}T_AUDIO_TONES_PARAMETER;

typedef struct {
  UINT16   start_tone;    // start time of the tone
  UINT16   stop_tone;     // stop time of the tone
  UINT16   frequency_tone;// frequency of the tone
  INT8     amplitude_tone;// amplitude of the tone
}T_AUDIO_TONE_DESC;
```

Below the detail of each parameters:

start_tone Specifies when the tone 1, 2, 3 must be start in ms unit.
stop_tone Specifies when the tone 1, 2, 3 must be stop in ms unit. **Note the stop_tone > start_tone and stop_tone-start_tone > 20ms.**
frequency_tone Specifies the frequency of the tone 1, 2, 3 in 1 Hz unit. Note the range is [0…2000] Hz. If the frequency value is equal to NO_TONE, the beep isn't generated.
amplitude_tone Specifies the amplitude of the tone 1, 2, 3 in 1 dB unit. Note the range is [-48…0] dB.
frame_duration Specifies the duration of the tones frame (c.f. figure below) in ms unit**. Note this duration can't be equal to 20ms.**
sequence_duration Specifies the duration of the sequence (c.f. figure below) in ms unit**. Note the sequence_duration >= frame_duration.**
period_duration Specifies the duration of the repetition (c.f. figure below) in ms unit**. Note the repetition_duration >= sequence_duration.**
repetition Specifies the number of repetition the tones defined with the parameters above must be played (c.f. figure below). **If the repetition = TONE_INFINITE, the tones is played indefinitely.**

To understand each parameter, please see the figure and example below:





The parameters corresponding to the figure above are:

```
parameter->tone[0].start_tone     = 0;
parameter->tone[0].stop_tone      = 60;
parameter->tone[0].frequency_tone = 520// Hz
parameter->tone[0].amplitude_tone = -24// dB
parameter->tone[1].start_tone     = 20;
parameter->tone[1].stop_tone      = 80;
parameter->tone[1].frequency_tone = 775// Hz
parameter->tone[1].amplitude_tone = -15// dB
parameter->tone[2].start_tone     = 100;
parameter->tone[2].stop_tone      = 140;
parameter->tone[2].frequency_tone = 643 // Hz
parameter->tone[2].amplitude_tone = -6  // dB
parameter->frame_duration         = 160;
parameter->sequence_duration      = 320;
parameter->period_duration        = 320;
parameter->repetition             = TONE_INFINITE;  // infinite tones
```

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_TONES_STATUS_MSG**

This event indicates that the tones task is stopped or an error occured.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_TONES_STATUS;
```

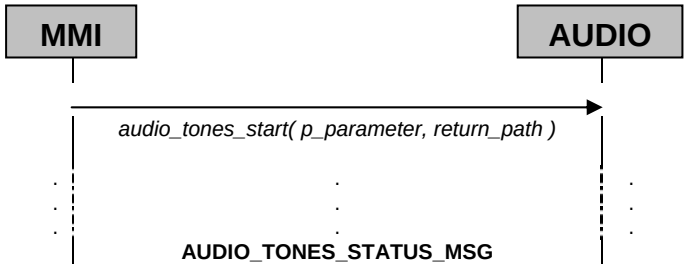
The possible values of *status* are:

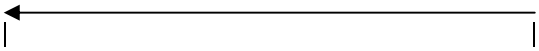
value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	The audio features was not successfully executed

Current restriction of use

Refer to section 4 on Audio Task Incompatibilities

Process flow





6.2 audio_tones_stop

```
T_AUDIO_RET audio_tones_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the tones generation**.**

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_keybeep_start*.

Event Return

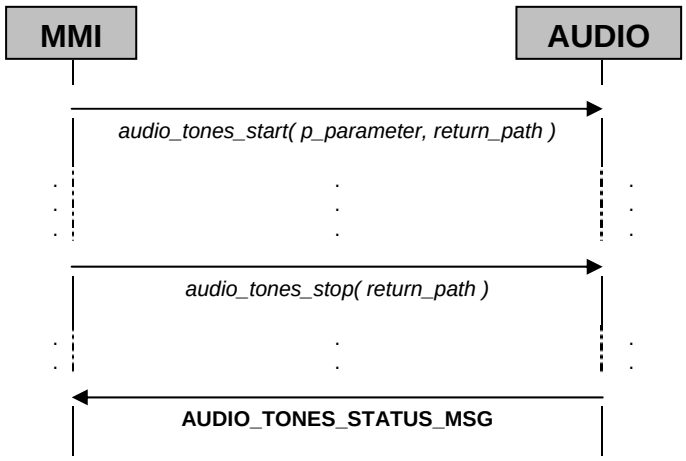
- **AUDIO_TONES_STATUS_MSG**

C.f. API function *audio_tones_start*.

Current restriction of use

Refer to section 4 on Audio Task Incompatibilities

Process flow



7 Melody E1 generation

This chapter describes how to generate a melody E1 format, using the AUDIO SW entity service.

7.1 audio_melody_E1_start

```
T_AUDIO_RET audio_melody_E1_start (
```

```
T_AUDIO_MELODY_E1_PARAMETER *p_parameter, T_RV_RETURN return_path)
```

Description

This function is called to initiate the melody E1 generation. **Note: two melodies can be run in parallel.**

Parameters

- **T_AUDIO_MELODY_E1_PARAMETER**

Specifies the characteristic of the melody to start.

```
typedef struct {
  char   melody_name[AUDIO_MELODY_PATH_NAME_MAX_SIZE];
  // File name of the melody
  BOOL   loopback;    // the melody is played indefinitely
  BOOL   melody_mode; // mode of the melody
}T_AUDIO_MELODY_E1_PARAMETER;
```

Below the detail of each parameters:

melody_name Specifies the file name of the melody. Note that this file name is used by the audio entity to request the melody data to the File Flash System. Moreover, the file name must contain the entire path to access to the melody file. Note the maximum size of the path plus the name is 80 characters.
loopback If *loop_back* = AUDIO_MELODY_LOOPBACK the melody is played indefinitely else one time else if *loop_back* = AUDIO_MELODY_NO_LOOPBACK the melody is played only one time.
melody_mode If *melody_mode* = AUDIO_MELODY_GAME_MODE two melody can be played in parallel in order to use the melody for the game (Note the 8 notes resource is shared between this two melody). If *melody_mode* = AUDIO_MELODY_NORMAL_MODE only one melody is played. So all the 8 notes resource is for this melody.

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_MELODY_E1_STATUS_MSG**

This event indicates that the melody task is stopped correctly or an error occurred during the execution.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_MELODY_E1_STATUS;
```

The possible values of *status* are:

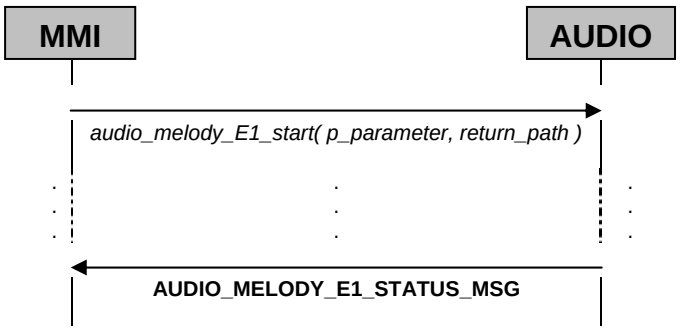
value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed
-2	AUDIO_MODE_ERROR	A melody is running in normal mode. Therefor, no more melody can not be run.

Current restriction of use

Refer to section 4 on Audio Task Incompatibilities

- In normal mode, only one melody can be run.
- Two melodies with the same name can't be run together (i.e. in game mode).

Process flow



7.2 audio_melody_E1_stop

```
T_AUDIO_RET audio_melody_E1_stop (
                              T_AUDIO_MELODY_E1_STOP_PARAMETER *p_parameter,
                              T_RV_RETURN return_path)
```

Description

This function is called in order to stop the melody generation.

Parameters

- **T_AUDIO_MELODY_E1_STOP_PARAMETER**

Specifies the characteristic of the melody to stop.

```
typedef struct {
  char          melody_name[AUDIO_MELODY_PATH_NAME_MAX_SIZE];
  // File name of the melody
}T_AUDIO_MELODY_E1_PARAMETER;
```

Below the detail of each parameters:

melody_name
Specifies the file name of the melody to stop. Note that this file name must be the name of the melody previously started. Moreover, the file name must contain the entire path to access to the melody file. Note the maximum size of the path plus the name is 80 characters.

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_keybeep_start*.

Event Return

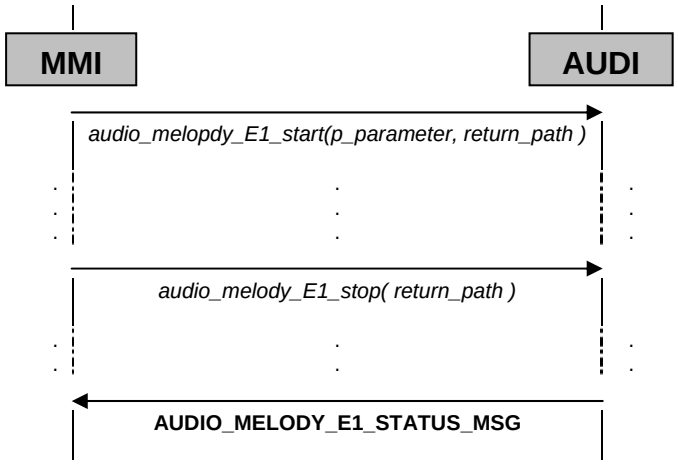
- **AUDIO_MELODY_E1_STATUS_MSG**

C.f. API function *audio_melody_E1_start*.

Current restriction of use

C.f. API function *audio_melody_E1_start*.

Process flow



8 Voice memorization

This chapter describes how to use the voice memorization, using the AUDIO SW entity service.

8.1 audio_vm_record_start

```
T_AUDIO_RET audio_vm_record_start (
            T_AUDIO_VM_RECORD_PARAMETER *p_record_parameter,
            T_AUDIO_TONES_PARAMETER *p_tones_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to initiate the voice memorization recording phase. Note tones are generated only if the conversation is recording during a call.

Parameters

- **T_AUDIO_VM_RECORD_PARAMETER**

Specifies the parameters using during the voice memorization phase.

```
typedef struct {
  char            memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];
  UINT32          memo_duration;       // maximum duration of the voice memo
  BOOL            compression_mode;    // activate the compression
  UINT16          microphone_gain;     // recording gain applies to microphone
  UINT16          network_gain;        // gain applies to the network voice
}T_AUDIO_VM_RECORD_PARAMETER;
```

Below the detail of each parameters:

memo_name Specifies the file name of the voice memo. Note that this file name is used by the audio entity to request the memo data to the File Flash System. Moreover, the file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 80 characters.
memo_duration Specifies the duration of the voice memo in second unit when the compression of the voice recorded is deactivated. In case of COMPRESSION_MODE, this duration indicates the minimum duration of the voice memo.
microphone_gain Specifies the gain multiplied to the voice sample from the microphone. The format is Q8.8, for example: if microphone_gain = 0x0100, the gain is 1 and if microphone_gain = 0x0080, the gain is 0,5.
network_gain Specifies the gain multiplied to the voice sample from the network (if the mobile is in dedicated mode). The format is Q8.8, for example: if network_gain = 0x0100, the gain is 1 and if network_gain = 0x0080, the gain is 0,5.
compression_mode Activate (COMPRESSION_MODE) or deactivate (NO_COMPRESSION_MODE) the compression of the voice recorded. It means that the silence between two voice activity are compressed.

- **T_AUDIO_TONES_PARAMETER**

See the API function: "*audio_tones_start*". Note that these tones are generated only if the conversation is recording during a call.

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_VM_RECORD_STATUS_MSG**

This event indicates that the melody task is stopped or an error is occurred.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
  UINT16     recorded_duration;
}T_AUDIO_VM_RECORD_STATUS;
```

Below the detail of the parameter:

recorded_duration
Specifies the size in seconds' unit of the recorded data.

The possible values of *status* are:

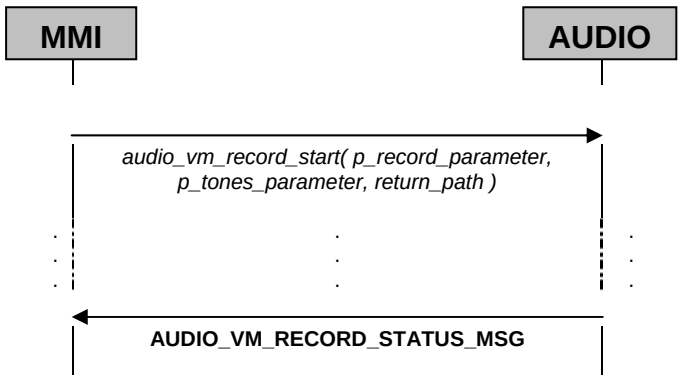
value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed

Current restriction of use

Refer to section 4 on Audio Task Incompatibilities

- All directories included in the pathname must be declared before

Process flow



8.2 audio_vm_record_stop

```
T_AUDIO_RET audio_vm_record_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the current voice memorization record.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

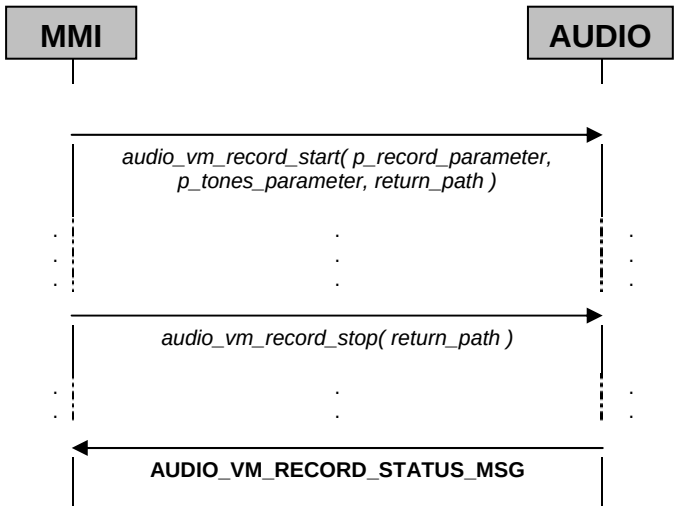
- **AUDIO_VM_RECORD_STATUS_MSG**

C.f. API function *audio_vm_record_start*.

Current restriction of use

C.f. API function *audio_vm_record_start*.

Process flow



8.3 audio_vm_play_start

```
T_AUDIO_RET audio_vm_play_start (T_AUDIO_VM_PLAY_PARAMETER *p_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to initiate the voice memorization playing phase.

Parameters

- **T_AUDIO_VM_PLAY_PARAMETER**

Specifies the parameters using during the voice memorization phase.

```
typedef struct {
  char           memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];
}T_AUDIO_VM_PLAY_PARAMETER;
```

Below the detail of each parameters:

memo_name
Specifies the file name of the voice memo. Note that this file name is used by the audio entity to request the memo data to the File Flash System. Moreover, the file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 80 characters.

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_VM_PLAY_STATUS_MSG**

This event indicates that the voice memorization playing task is stopped or an error is occurred.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_VM_PLAY_STATUS;
```

The possible values of *status* are:

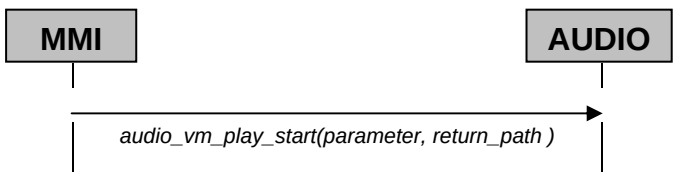
value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed

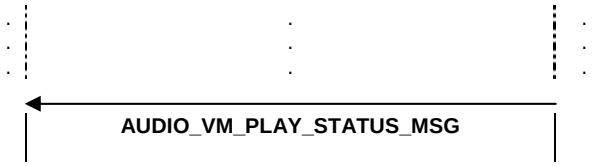
Current restriction of use

Refer to section 4 on Audio Task Incompatibilities

- All directories included in the pathname must be declared before

Process flow




AUDIO_VM_PLAY_STATUS_MSG

8.4 audio_vm_play_stop

```
T_AUDIO_RET audio_vm_play_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the current voice memorization play.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

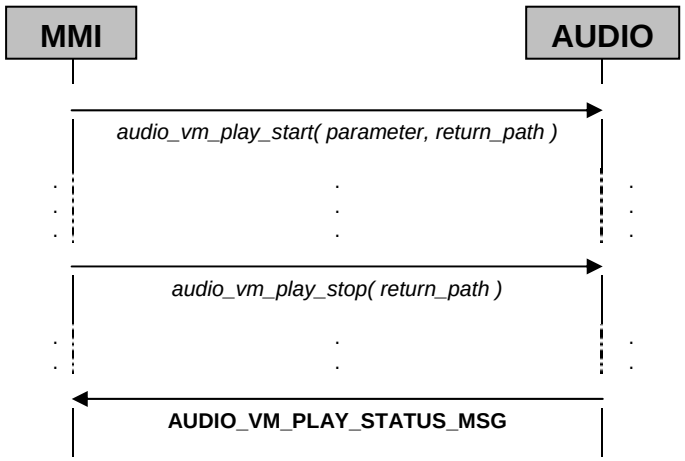
- **AUDIO_VM_PLAY_STATUS_MSG**

C.f. API function *audio_vm_play_start*.

Current restriction of use

C.f. API function *audio_vm_play_start*.

Process flow



9 Audio mode configuration

9.1 Introduction

This document sums up all the API functions useful to handle all possible the audio paths embedded in a mobile. These API functions can be grouped in several family of use:

- **The MMI family:**
  These API functions are dedicated to facilitate the creation, the calibration, and the change of the audio mode.

Note: An audio mode is a fixed setting of all audio features embedded in the mobile. For example, during an incoming call, the mobile rings so this task uses a particular setting of all audio features therefore it corresponds to a particular audio mode. In this case, the audio mode is called RING_MODE.

- **The full access family:**
  These API functions are dedicated to permit a direct tuning of all audio module involved in the audio paths. For example, with these functions, you can directly tune the PGA gain of the microphone connected to the Analog Base Band.

9.2 MMI family

This chapter describes all the API functions belong to the MMI family.
Before to define these API functions, the first step is to define several vocabulary and concept used in this chapter.

- Audio mode:
  An audio mode is a fixed setting of all audio features embedded in the mobile. For example, during an incoming call, the mobile rings so this task uses a particular setting of all audio features therefore it corresponds to a particular audio mode. In this case, the audio mode is called RING_MODE.
  The list of all standard audio modes is listed below, but there's a possibility to extend this list.

Moreover, the audio setting of an audio mode can be grouped in several family of audio setting:

- Audio path setting:
  This group of setting is used to define the audio path used.
  The different audio paths are:
  - **GSM normal path**: voice samples are exchanged between GSM network and GSM Analog Base Band.
  - **Bluetooth Cordless path**: voice samples are exchanged between the GSM Analog Base Band and the Bluetooth module.
  - **Bluetooth Headset path**: voice samples are exchanged between GSM network and Bluetooth module.
  - **DAI encoder path**: path to test the speech encoder and its DTX functions.
  - **DAI decoder path**: path to test the speech decoder and its DTX functions.
  - **DAI acoustic path**: path to test the acoustic devices and the audio A/D and D/A devices.
- Microphone voice path setting:
  This group of setting configures the audio voice path of the microphone.
- Speaker voice path setting:
  This group of setting configures the audio voice path of the speaker.
- Microphone speaker loop setting:
  This group of setting configures the audio module involved in the microphone and speaker voice loop.
- Speaker audio stereo path:
  This group of setting configures the audio stereo path of the speaker.

9.2.1 Audio mode file structure

The audio mode is described by the structure below *T_AUDIO_MODE*. So for each audio mode (i.e. game, audio off, ringer, handheld…), a *T_AUDIO_MODE* variable is saved in a flash file in the folder */aud/* with the extension .cfg. The file name is specified by the customer (c.f. audio_mode_save/load function)

9.2.1.1 T_AUDIO_MODE

Specifies the structure of each audio mode:

- With analog base band OMEGA and IOTA:

```
typedef struct
{
  /* group of setting to define the audio path used */
  T_AUDIO_VOICE_PATH_SETTING              audio_path_setting;
  /* group of setting to configure the audio voice path of the microphone */
  T_AUDIO_MICROPHONE_SETTING              audio_microphone_setting;
  /* group of setting to configure the audio voice path of the speaker */
  T_AUDIO_SPEAKER_SETTING                 audio_speaker_setting;
  /* group of setting to configure the audio mode involved */
  /* in the microphone and speaker loop */
  T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING audio_microphone_speaker_loop_setting;
}
T_AUDIO_MODE;
```

- With analog base band SYREN

```
typedef struct
{
  /* group of setting to define the audio path used */
  T_AUDIO_VOICE_PATH_SETTING              audio_path_setting;
  /* group of setting to configure the audio voice path of the microphone */
  T_AUDIO_MICROPHONE_SETTING              audio_microphone_setting;
  /* group of setting to configure the audio voice path of the speaker */
  T_AUDIO_SPEAKER_SETTING                 audio_speaker_setting;
  /* group of setting to configure the audio stereo path of the speaker */
  T_AUDIO_STEREO_SPEAKER_SETTING          audio_stereo_speaker_setting;
  /* group of setting to configure the audio mode involved */
  /* in the microphone and speaker loop */
  T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING audio_microphone_speaker_loop_setting;
  /* group of settings to configure audio features common to */
  /* microphone and speaker */
  T_AUDIO_MICROPHONE_SPEAKER_SETTING audio_microphone_speaker_setting;
}
T_AUDIO_MODE;
```

- With analog base band TRITON

```
typedef struct
{
  /* group of setting to define the audio path used */
  T_AUDIO_VOICE_PATH_SETTING              audio_path_setting;
  /* group of setting to configure the audio voice path of the microphone */
  T_AUDIO_MICROPHONE_SETTING              audio_microphone_setting;
  /* group of setting to configure the audio voice path of the speaker */
  T_AUDIO_SPEAKER_SETTING                 audio_speaker_setting;
  /* group of setting to configure the audio stereo path of the speaker */
  T_AUDIO_STEREO_SPEAKER_SETTING          audio_stereo_speaker_setting;
  /* group of setting to configure the audio mode involved */
  /* in the microphone and speaker loop */
  T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING audio_microphone_speaker_loop_setting;
  /* group of settings to configure audio features common to */
  /* microphone and speaker */
  T_AUDIO_MICROPHONE_SPEAKER_SETTING audio_microphone_speaker_setting;
}
T_AUDIO_MODE;
```

- With non-TI audio analog base band used with P2 samples

```
typedef struct
{
  /* group of setting to define the audio path used */
  T_AUDIO_VOICE_PATH_SETTING              audio_path_setting;
  /* group of setting to configure the audio voice path of the microphone */
  T_AUDIO_MICROPHONE_SETTING              audio_microphone_setting;
  /* group of setting to configure the audio voice path of the speaker */
  T_AUDIO_SPEAKER_SETTING                 audio_speaker_setting;
  /* group of setting to configure the audio mode involved */
  /* in the microphone and speaker loop */
  T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING audio_microphone_speaker_loop_setting;
  /* group of settings to configure audio features common to */
  /* microphone and speaker */
  T_AUDIO_MICROPHONE_SPEAKER_SETTING audio_microphone_speaker_setting;
}
T_AUDIO_MODE;
```

9.2.1.2 T_AUDIO_VOICE_PATH_SETTING

This parameter specifies the audio path mode.

```
/* audio path used */
typedef UINT8 T_AUDIO_VOICE_PATH_SETTING;
```

The different values are:

Value	Path
AUDIO_GSM_VOICE_PATH	GSM normal
AUDIO_BLUETOOTH_CORDLESS_VOICE_PATH	Bluetooth cordless
AUDIO_BLUETOOTH_HEADSET_PATH	Bluetooth headset
AUDIO_DAI_ENCODER	DAI encoder
AUDIO_DAI_DECODER	DAI decoder
AUDIO_DAI_ACOUSTIC	DAI acoustic

9.2.1.3 T_AUDIO_MICROPHONE_SETTING

Specifies the setting of the microphone voice path.

- For P2 samples with non TI audio ABB :

```
typedef struct
{
  /* gain of the microphone */
  INT16    micro_gain;
  /* coefficients of the speaker FIR */
  T_AUDIO_FIR_COEF    fir;
   /* ANR configuration */
   T_AUDIO_ANR_CFG     anr;
   /* ES configuration */
   T_AUDIO_ES_CFG      es;
}
T_AUDIO_MICROPHONE_SETTING;
```

IMPORTANT NOTE:

ANR and echo suppressor (ES) features are only present in TCS 3.x software except TCS 3.0. Other software versions do not include T_AUDIO_ANR_CFG and T_AUDIO_ES_CFG structures.

For T_AUDIO_ANR_CFG details: see 9.2.1.3.3.

For T_AUDIO_ES_CFG details: see 9.2.1.3.4

micro_gain Gain of the microphone. The range is from 0x0001 (-72 dB) to 0x7FFF (18 dB) in signed Q12 format, ex: 0dB → (2^12)*10^(0/20) = 0x1000 -6 dB → (2^12)*10^(-6/20) = 0x804 0x4000 → 20*log(16384/(2^12)) = 12 dB
Fir List of the 31 coefficients of the FIR of the microphone. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. Note: the FIR is available only in DAI and GSM path mode.
Anr *See 9.2.1.3.3 for details.*

- For analog base band TRITON:

```
typedef struct
{
  /* mode of the microphone */
  INT8                    mode;
  /* Setting of the current mode */
  T_AUDIO_MICROPHONE_MODE  setting;
}
T_AUDIO_MICROPHONE_SETTING;
```

Where the microphone modes are :

```
typedef union
 {
   /* handheld mode parameters */
   T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB      handset_25_6db;
  /* handheld mode parameters */
   T_AUDIO_MICROPHONE_MODE_HEADSET_4_9_DB      headset_4_9db;
  /* handheld mode parameters */
   T_AUDIO_MICROPHONE_MODE_HEADSET_25_6DB      headset_25_6db;
  /* handheld mode parameters */
   T_AUDIO_MICROPHONE_MODE_HEADSET_18DB       headset_18db;
  /* Aux mode parameters */
   T_AUDIO_MICROPHONE_MODE_AUX_4_9DB          aux_4_9db;
  /* Aux mode parameters */
   T_AUDIO_MICROPHONE_MODE_AUX_28_2DB         aux_28_2db;
   /* handfree mode parameters */
   T_AUDIO_MICROPHONE_MODE_FM_MONO_4_9db       fm_mono_4_9db;
   /* headset mode parameters */
   T_AUDIO_MICROPHONE_MODE_CARKIT_4_9db        carkit_4_9db;
   /* FM */
   T_AUDIO_MICROPHONE_MODE_FM_4_9db            fm_4_9db;
 }
 T_AUDIO_MICROPHONE_MODE;

typedef struct
  {
    /* gain of the microphone */
    INT8    gain;
    /* microphone output bias */
    INT8    output_bias;
    /* coefficients of the microphone FIR */
    T_AUDIO_FIR_COEF    fir;
    /* ANR configuration */
    T_AUDIO_ANR_CFG     anr;
    /* AGC UL configuration */
    T_AUDIO_AGC_UL_CFG     agc ;
  }
  T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB;

typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_HEADSET_4_9_DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_HEADSET_25_6DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_HEADSET_18DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_AUX_4_9DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_AUX_28_2DB;
typedef T_AUDIO_MICROPHONE_MODE_HANDSET_25_6DB T_AUDIO_MICROPHONE_MODE_CARKIT_4_9db;
typedef struct
  {
    /* gain of the microphone */
    INT8    gain;
```

```
        /* microphone output bias */
        INT8    output_bias;
        /* microphone output bias */
        INT8    extra_gain;
      }
      T_AUDIO_MICROPHONE_MODE_FM_MONO_4_9db;

typedef T_AUDIO_MICROPHONE_MODE_FM_MONO_4_9db T_AUDIO_MICROPHONE_MODE_FM_4_9db;
```

IMPORTANT NOTE:

ANR feature is only present in TCS 3.x software except TCS 3.0. Other software versions do not include T_AUDIO_ANR_CFG structure.

The AGC module is included in TCS 3.2 software.

For T_AUDIO_ANR_CFG details: see 9.2.1.3.1

For T_AUDIO_AGC_UL_CFG details: see 9.2.1.3.2

Mode Specifies the mode of the microphone: **AUDIO_MICROPHONE_MODE_HANDSET_25_6DB** or **AUDIO_MICROPHONE_MODE_HEADSET_4_9_DB** or **AUDIO_MICROPHONE_MODE_HEADSET_25_6DB** or **AUDIO_MICROPHONE_MODE_HEADSET_18DB or AUDIO_MICROPHONE_MODE_AUX_4_9DB or AUDIO_MICROPHONE_MODE_AUX_28_2DB or AUDIO_MICROPHONE_MODE_CARKIT_4_9DB or AUDIO_MICROPHONE_MODE_FM_MONO_4_9db or AUDIO_MICROPHONE_MODE_FM_4_9db**	
All the modes are available in GSM, bluetooth cordless voice and all DAI path mode.	
	gain gain of the microphone in 1dB unit. The range is from –12 dB to 12 dB. Note if the gain is equal to AUDIO_MICROPHONE_MUTE, the microphone is muted.
	output_bias Specifies the output voltage of the microphone. This value could be 2.0V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_0V) or 2.5 V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_5V).
	extra_gain **Specifies the FM gain. The range is -2dB to 14dB in 2dB steps which corresponds to 0 to 8. Applicable only for `T_AUDIO_MICROPHONE_MODE_FM_MONO` and `T_AUDIO_MICROPHONE_MODE_FM`.**
	fir_coef List of the 31 coefficients of the FIR of the microphone. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.**
	anr *See 9.2.1.3.1 for details*
	agc *See 9.2.1.3.2 for details*

9.2.1.3.1 T_AUDIO_ANR_CFG

ANR (Ambient Noise Reduction) module allows reducing the noise present in the speech uplink path.

ANR is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the microphone settings.
ANR module only works in DAI acoustic and GSM and Bluetooth headset path modes.
The following ANR settings are to be used for ANR 2.13 used in the TCS 3.2 acoustic chain.

ANR settings are inside following structure of the microphone settings:

```
typedef struct
```

```
{
  T_ANR_CONTROL           anr_ul_control;
   T_AUDIO_ANR_PARAMS     parameters ;
}
T_AUDIO_ANR_CFG;

typedef enum
{
  ANR_STOP  = 0,
  ANR_START = 1,
  ANR_UPDATE = 2
}
T_ANR_CONTROL;

typedef struct
{
  INT16    control;
  INT16    ns_level;
  INT16    tone_ene_th;
  INT16    tone_cnt_th ;
}
T_AUDIO_ANR_PARAMS;
```

control Setting of ANR module. The supported values are: 0x0000 : module is bypassed (ANR disabled) 0x0001 : ANR and tone detector are enabled (default value) 0x0003 : ANR is enabled and tone detector is disabled
In case of Read Access, the following parameters are valid only if control is non-zero
ns_level Attenuation applied in the spectral subtraction. Attenuation is equal to ns_level x 6dB **Recommended value is 0x0002 (12 dB).** Other applicable values: 0x0000 : noise attenuation depends on incoming signal SNR 0x0001 : 6 dB noise attenuation 0x0002 : 12 dB noise attenuation (default value)
tone_ene_th SNR threshold for tone detection. 0x0007 : is default value set for an SNR of 21 dB. This is the recommended value. Lowering threshold would cause over-detection of tone and, in turn, would cause sputtered noise in speech. Increasing threshold would cause under-detection and might modify audio test cases FTA test signals.
tone_cnt_th Maximum number of tones to be detected. 0x0001: is the default value. Changing value to 0x0002 allows detecting dual-tone such as DTMF tone, but it could cause over detection (annoying sputtered noise in speech). Therefore, the value of 0x0001 is recommended. Even if DTMF tone is not detected, ANR 2.13 attenuates DTMF tone level only with very small amount and it should not be a problem

9.2.1.3.2 T_AUDIO_AGC_UL_CFG

The primary goal of AGC is to adjust the input speech signal to a targeted level at the output. AGC is used in both the uplink and downlink in the TCS3.2 acoustic chain.
AGC is only available from TCS 3.2. AGC module only works in DAI acoustic and GSM and Bluetooth headset path modes.
The following AGC settings are to be used for AGC 1.x.

AGC settings are inside following structure of the microphone settings:

```
typedef struct
```

```
{
  T_AGC_CONTROL           agc_ul_control;
   T_AUDIO_AGC_PARAMS     parameters ;
}
T_AUDIO_AGC_UL_CFG;

typedef enum
{
  ANR_STOP  = 0,
  ANR_START = 1,
  ANR_UPDATE = 2
}
T_ANR_CONTROL;

typedef struct
{
    UINT16  control;
    UINT16  frame_size;
    INT16   targeted_level;
    INT16   signal_up;
    INT16   signal_down;
    INT16   max_scale;
    INT16   gain_smooth_alpha;
    INT16   gain_smooth_alpha_fast;
    INT16   gain_smooth_beta;
    INT16   gain_smooth_beta_fast;
    INT16   gain_intp_flag;
}
T_AUDIO_AGC_PARAMS;
```

• name	• Description	• Type/Format	• Range	• Comment
Control	enable/disable	UINT16/Q0	[0x0000,	disabled
			0x0001,	enabled 8kHz
			0x0002]	enabled 16kHz [1]
In case of Read Access, the following parameters are valid only if control is non-zero				
frame_size	aquistion/restoration frame size	UINT16/Q0	[0x0050,	80 samples, 1X10ms frame processing (FP at 8kHz
			0x00A0,	160 samples, 2X10ms FP (8kHz), 1X10ms FP (16kHz)
			0x0140]	320 samples, 2X10ms FP (16kHz)
targeted_level [2, 3]	targeted level from dBm0 spec.	INT16/Q15	[0x0502…	minimal: -22dBm0 power
			0x09FE…	nominal: -16dBm0 power
			0x13F0]	maximal: -10dBm0 power
signal_up[4]	gain up from dB spec.	INT16/Q10	0x7E7E	nominal: +15dB power
signal_down[5]	gain down from dB spec.	INT16/Q15	0x7FB4	nominal: Ref. -0.01dB

[1] Though AGC 1.x has the capability to operate at 8 kHz and 16 kHz sampling frequencies, the TCS 3.2 voice acoustic chain modules currently operate only at 8 kHz. So in the TCS 3.2 acoustic chain, AGC 1.x can be set to operate at ONLY at 8 kHz. Hence parameter control shall not use value 0x0002 for TCS 3.2

[2] The AGC targeted level can be updated on the fly from frame to frame without re-initialization. For example, the targeted level can be changed from -16dBm0 to -10dBm0 from frame m to frame m + 1.

[3] The targeted level of the signal at AGC output could be computed from the specification of targeted level in dBm0 as follows:

$$\text{targeted\_level} = round\left\{2^{15}.10^{\frac{L_{\text{Target}}-6.15}{20}}\right\},$$

where the targeted level is L . For example, L $= -16dBm0$ leads to targeted_level = 2558 i.e. targeted_level = 0x09FE.

[4] The AGC gain increasing rate signal_up could be customized from specification of rate in dB as follows:

$$\text{signal\_up} = round\left\{\frac{2^{15}}{2^{5}}.10^{\frac{\Gamma_{up}}{10}}\right\},$$

where the increasing rate is Γ_{up} . For example, $\Gamma_{up} = 15dB$ leads to signal_up = 32382 i.e. signal_up = 0x7E7E.

[5] The AGC gain decreasing rate signal_down could be customized from specification of rate in dB as follow:

$$\text{signal\_down} = round\left\{2^{15}.10^{\frac{\Gamma_{down}}{10}}\right\},$$

where the increasing rate is Γ_{down} . For example, $\Gamma_{down} = -0.01dB$ leads to signal_down = 32692 i.e. signal_down = 0x7FB4.

max_scale[6]	maximum gain from dB spec.	INT16/Q12	0x59F9	nominal Ref. +15dB
gain_smooth_alpha	gain smoothing factor	INT16/Q15	0x7EB8	nominal Ref. 0.99
gain_smooth_alpha_fast	gain fast smoothing factor	INT16/Q15	0x7333	nominal Ref. 0.9
gain_smooth_beta	gain smoothing factor	INT16/Q15	0x7F5C	nominal Ref. 0.095
gain_smooth_beta_fast	gain fast smoothing factor	INT16/Q15	0x7333	nominal Ref. 0.9
gain_intp_flag[7]	gain interpolation enable/disable	INT16/Q0	0x0000	nominal, interpolation disabled
			0x0001	interpolation enabled

[6] The AGC maximal amplification max_scale could be customized from specification of amplification in dB as follow:

$$\text{max\_scale} = round\left\{\frac{2^{15}}{2^{3}}.10^{\frac{g_{\max}}{20}}\right\},$$

where the maximum amplification is $g_{\max}$. For example, $g_{\max} = 15dB$ leads to max_scale = 23033 i.e. 0x59F9

[7] The AGC gain is interpolated from frame to frame on 32 samples (8000Hz) or 64 samples (16000Hz). The interpolation can be enable/disable using gain_intp_flag. Nominal value is gain_intp_flag = 0.

For other configurations :

```
typedef struct
{
  /* mode of the microphone */
  INT8                      mode;
  /* Setting of the current mode */
  T_AUDIO_MICROPHONE_MODE  setting;
}
T_AUDIO_MICROPHONE_SETTING;
```

Where the microphone modes are :

```
typedef union
{
  /* handheld mode parameters */
  T_AUDIO_MICROPHONE_MODE_HANDHELD handheld;
  /* handfree mode parameters */
  T_AUDIO_MICROPHONE_MODE_HANDFREE handfree;
  /* headset mode parameters */
  T_AUDIO_MICROPHONE_MODE_HEADSET  headset;
  /* headset differential mode parameters */
  T_AUDIO_MICROPHONE_MODE_HEADSET_DIFF  headset_diff;
}
T_AUDIO_MICROPHONE_MODE;



typedef struct
{
  UINT16 coefficient[31];
}
T_AUDIO_FIR_COEF;

typedef struct
{
  /* gain of the microphone */
  INT8    gain;
  /* microphone output bias */
  INT8    output_bias;
  /* coefficients of the microphone FIR */
  T_AUDIO_FIR_COEF    fir;
  /* ANR configuration*/
  T_AUDIO_ANR_CFG     anr;
  /* ES configuration */
  T_AUDIO_ES_CFG      es;

}
T_AUDIO_MICROPHONE_MODE_HANDHELD;

typedef struct {
  /* gain of the microphone */
  INT8    gain;
  /* extra gain of the microphone */
  INT8    extra_gain;
  /* microphone output bias */
  INT8    output_bias;
  /* coefficients of the microphone FIR */
  T_AUDIO_FIR_COEF    fir;
  /* ANR configuration */
  T_AUDIO_ANR_CFG     anr;
  /* ES configuration */
  T_AUDIO_ES_CFG      es;
}
T_AUDIO_MICROPHONE_MODE_HANDFREE;

typedef struct {
  /* gain of the microphone */
  INT8    gain;
  /* microphone output bias */
  INT8    output_bias;
  /* coefficients of the microphone FIR */
  T_AUDIO_FIR_COEF    fir;
  /* ANR configuration */
```

```
    T_AUDIO_ANR_CFG     anr;
    /* ES configuration */
    T_AUDIO_ES_CFG      es;
  }
  T_AUDIO_MICROPHONE_MODE_HEADSET;

typedef struct {
    /* gain of the microphone */
    INT8    gain;
    /* microphone output bias */
    INT8    output_bias;
    /* coefficients of the microphone FIR */
    T_AUDIO_FIR_COEF    fir;
    /* ANR configuration */
    T_AUDIO_ANR_CFG     anr;
    /* ES configuration */
    T_AUDIO_ES_CFG      es;
  }
  T_AUDIO_MICROPHONE_MODE_HEADSET_DIFF;
```

IMPORTANT NOTE:

ANR and Echo Suppressor (ES) features are only present in TCS 3.x software except TCS 3.0. Other software versions do not include T_AUDIO_ANR_CFG and T_AUDIO_ES_CFG structures.

For T_AUDIO_ANR_CFG details: see 9.2.1.3.3.

For T_AUDIO_ES_CFG details: see 9.2.1.3.4.

The different values of each microphone setting parameters are :

Mode Specifies the mode of the microphone: AUDIO_MICROPHONE_HANDHELD or AUDIO_MICROPHONE_HANDFREE or AUDIO_MICROPHONE_HEADSET or AUDIO_MICROPHONE_HEADSET_DIFFERENTIAL..	
AUDIO_MICROPHONE_HANDHELD mode: this mode is available in GSM, bluetooth cordless voice and all DAI path mode.	
	Gain gain of the microphone in 1dB unit. The range is from –12 dB to 12 dB. Note if the gain is equal to AUDIO_MICROPHONE_MUTE, the microphone is muted.
	Output_bias Specifies the output voltage of the microphone. This value could be 2.0V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_0V) or 2.5 V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_5V).
	fir_coef List of the 31 coefficients of the FIR of the microphone. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.**
	Anr *See 9.2.1.3.3 for details.*
	Es *See 9.2.1.3.4 for details*
AUDIO_MICROPHONE_HANDFREE mode: this mode is available in GSM, bluetooth cordless voice and all DAI path mode.	
	Gain gain of the microphone in 1dB unit. The range is from –12 dB to 12 dB. Note if the gain is equal to AUDIO_MICROPHONE_MUTE, the microphone is muted.
	extra_gain Specifies an additional gain of the microphone handfree path: 4.6 dB (AUDIO_MICROPHONE_AUX_GAIN_4_6dB) or 28.2 dB (AUDIO_MICROPHONE_AUX_GAIN_28_2dB).

	output_bias Specifies the output voltage of the microphone. This value could be 2.0V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_0V) or 2.5 V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_5V).
	fir_coef List of the 31 coefficients of the FIR of the microphone. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.**
	anr *See 9.2.1.3.3 for details.*
	Es *See 9.2.1.3.4 for details*
AUDIO_MICROPHONE_HEADSET mode: this mode is available in GSM, bluetooth cordless voice and DAI path mode. It is only available with the analog base band IOTA and SYREN.	
	Gain gain of the microphone in 1dB unit. The range is from –12 dB to 12 dB. Note if the gain is equal to AUDIO_MICROPHONE_MUTE, the microphone is muted.
	output_bias Specifies the output voltage of the microphone. This value could be 2.0V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_0V) or 2.5 V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_5V).
	fir_coef List of the 31 coefficient of the FIR of the microphone. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.**
	anr *See 9.2.1.3.3 for details.*
	Es *See 9.2.1.3.4 for details*
AUDIO_MICROPHONE_HEADSET_DIFFERENTIAL mode: this mode is available in GSM, bluetooth cordless voice and DAI path mode. It is only available with the analog base band SYREN.	
	Gain gain of the microphone in 1dB unit. The range is from –12 dB to 12 dB. Note if the gain is equal to AUDIO_MICROPHONE_MUTE, the microphone is muted.
	output_bias Specifies the output voltage of the microphone. This value could be 2.0V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_0V) or 2.5 V (AUDIO_MICROPHONE_OUTPUT_BIAS_2_5V).
	fir_coef List of the 31 coefficient of the FIR of the microphone. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. Note: the FIR is available only in DAI and GSM path mode.
	anr *See 9.2.1.3.3 for details.*
	Es *See 9.2.1.3.4 for details*

9.2.1.3.3 T_AUDIO_ANR_CFG

ANR (Ambient Noise Reduction) module allows reducing the noise present in the speech uplink path.

ANR is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the microphone settings.
ANR module only works in DAI acoustic and GSM path mode.

ANR settings are inside following structure of the microphone settings:

```
typedef struct
{
```

```
  BOOLEAN  anr_enable;
  INT16    min_gain;
  INT8     div_factor_shift;
  UINT8    ns_level;
}
T_AUDIO_ANR_CFG;
```

ANR excepted noise attenuation (dB) = Temp. Att. (dB) + Spec. Att. (dB).

anr_enable
Enable/disable the ANR (Ambiant Noise Reduction) module:
0- disable
1- enable

In case of Read Access, the following parameters are valid only if anr_enable = 1.

min_gain
Temp. Att. (dB): temporal attenuation applied on signal detected as noise, considering speech isn't attenuated. Format is Q15.
Temp. Att. (dB) = 20*log(d_anr_min_gain/32768);
Ex: d_anr_min_gain = 0x4000 -> Temp. Att. = -6dB
Recommended value is 0x3313 (-8 dB)

div_factor_shift
Used to control variations of temporal attenuation. Time periods where signal is considered as noise are attenuated. In order to avoid erroneous speech attenuation, this value permits to adjust the freezing of the gain after the speech detection.
Recommended value is -2.

anr_ns_level
Spec. Att. (dB) : spectral subtraction in 6dB steps.
Ex: d_anr_ns_level | Spec. Att. (dB)

Ex:	-1	0 dB (*)
	0	-x dB (**)
	1	-6 dB
	2	-12 dB

(*) Customers shouldn't use ANR2.0 without spectral subtraction ;
(**) ANR 2.0 performs the maximum of spectral subtraction depending on the incoming signal.
Recommended value is 1 (-6 dB).

9.2.1.3.4 T_AUDIO_ES_CFG

The echo suppressor (ES) role is to control the residual echo in a speakerphone application, where the AEC is unable to cancel the entire echo in the uplink due to non-ideal acoustical environment such as a non-linear loudspeaker response for example. Please refer to [7] for an overview of the module.

ES is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the microphone settings.
ES module only works in DAI acoustic and GSM path mode.

ES settings are inside following structure of the microphone settings:

```
typedef struct
{
  BOOLEAN es_enable;
  UINT8   es_behavior;
  UINT8   es_mode;
  INT16   es_gain_dl;
  INT16   es_gain_ul_1;
  INT16   es_gain_ul_2;
  INT16   tcl_fe_ls_thr;
  INT16   tcl_dt_ls_thr;
  INT16   tcl_fe_ns_thr;
  INT16   tcl_dt_ns_thr;
  INT16   tcl_ne_thr;
```

```
  INT16   ref_ls_pwr;
  UINT16  switching_time;
  UINT16  switching_time_dt;
  UINT16  hang_time;
  INT16   gain_lin_dl_vect[4];
  INT16   gain_lin_ul_vect[4];
}
T_AUDIO_ES_CFG;
```

es_enable
Enable/disable the echo suppressor module:
0- disable
1- enable

In case of Read Access, the following parameters are valid only if es_enable = 1.

es_behavior
Permit to setup pre-defined ES behavior as described in [8]:
0- Behavior 1
1- Behavior 1a
2- Behavior 2a
3- Behavior 2b
4- Behavior 2c
5- Behavior 2c_idle
6- Custom: all parameters setup by the user

In 'custom' mode, following parameters must be set. Custom mode isn't recommended.
In other modes, following parameters aren't used.

es_mode
Bitmap defining the ES mode:

bit		
0	ES UL	0- Disable ES on UL path 1- Enable ES on UL path
1	ES DL	0- Disable ES on DL path 1- Enable ES on DL path
2	CNG	0- Disable CNG* algorithm 1- Enable CNG* algorithm
3	NSF	0- Disable NSF** algorithm 1- Enable NSF** algorithm
4	ALS UL	0- Disable ALS*** on UL path 1- Enable ALS*** on UL path
5	ALS DL	0- Disable ALS*** on DL path 1- Enable ALS*** on DL path

* CNG = Comfort Noise Generation
** NSF=Noise Floor
*** ALS = Attenuation Level Smoothing

Notes:
- **Disabling ES UL has no sense**
- **CNG and NSF mustn't be enabled together**

es_gain_dl
es_gain_dl is the receive loss compensation.

es_gain_ul_1
es_gain_ul_1 is the coupling loss compensation.

es_gain_ul_2 *es_gain_ul_2* is the near-end propagation loss compensation.
tcl_fe_ls_thr *d_tcl_fe_ls_thr* is the TCL reference threshold in far-end mode for loud signals. This value is in Q15 format.
tcl_dt_ls_thr *d_tcl_fd_ls_thr* is the TCL reference threshold in double-talk mode for loud signals. This value is in Q15 format
tcl_fe_ns_thr *d_tcl_fe_ns_thr* is the TCL reference threshold in far-end mode for nominal signals. This value is in Q15 format
tcl_dt_ns_thr *d_tcl_fd_ns_thr* is the TCL reference threshold in double-talk mode for nominal signals. This value is in Q15 format
tcl_ne_thr *d_tcl_ne_thr* is the TCL reference threshold in near-end mode. This value is in Q15 format
ref_ls_pwr *d_ref_ls_pwr* is the TCL reference threshold in near-end mode. This value is in Q15 format
switching_time *d_ switching_time_dt* is the switching time value in milliseconds.
switching_time_dt *d_ switching_time_dt* is the double-talk switching time value in milliseconds.
hang_time *d_hang_time* is the hangover time for switching
gain_lin_dl_vect Table containing downlink linear attenuation levels per state: gain_lin_dl_vect[0] - idle state gain_lin_dl_vect[1] - double talk gain_lin_dl_vect[2] - far-end gain_lin_dl_vect[3] - near-end Format is Q15.
gain_lin_ul_vect Table containing uplink linear attenuation levels per state: gain_lin_ul_vect[0] - idle state gain_lin_ul_vect[1] - double talk gain_lin_ul_vect[2] - far-end gain_lin_ul_vect[3] - near-end Format is Q15.

9.2.1.4 T_AUDIO_SPEAKER_SETTING

Specifies the characteristic of the speaker voice path.

- For P2 samples with non TI audio ABB :

```
typedef struct
{
  /* gain of the speaker */
  INT16    speaker_gain;
  /* coefficients of the speaker FIR */
  T_AUDIO_FIR_COEF    fir;
   /* Limiter parameters */
   T_AUDIO_LIMITER_CFG limiter;
   /* IIR filter parameters */
   T_AUDIO_IIR_CFG     iir;
}
T_AUDIO_SPEAKER_SETTING;
```

IMPORTANT NOTE:

IIR and LIMITER features are only present in TCS 3.x software except TCS 3.0. Other software versions do not include T_AUDIO_LIMITER_CFG and T_AUDIO_IIR_CFG structures.

As the IIR filter replaces the FIR filter, T_AUDIO_FIR_COEF isn't present in software versions supporting the IIR filter.

For parameters details:

T_AUDIO_LIMITER_CFG: see 9.2.1.4.4

T_AUDIO_IIR_CFG: see 9.2.1.4.5

speaker_gain Gain of the speaker. The range is from 0x0001 (-72 dB) to 0x7FFF (18 dB) in signed Q12 format, ex: 0dB → (2^12)*10^(0/20) = 0x1000 -6 dB → (2^12)*10^(-6/20) = 0x804 0x4000 → 20*log(16384/(2^12)) = 12 dB
Fir List of the 31 coefficients of the FIR of the microphone. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.** **The FIR filter is replaced by IIR filter in TCS3.x software, except TCS 3.0.**

- For analog base band TRITON:

```
 typedef struct
{
  /* mode of the speaker */
  INT8                      mode;
  /* Setting of the current mode */
  T_AUDIO_SPEAKER_MODE      setting;
}
T_AUDIO_SPEAKER_SETTING;
```

where the speaker modes are:

```
typedef union
{
  /* handheld mode parameters */
  T_AUDIO_SPEAKER_MODE_HANDHELD handheld;
  /* handfree mode parameters */
  T_AUDIO_SPEAKER_MODE_HANDFREE handfree;
  /* headset mode parameters */
  T_AUDIO_SPEAKER_MODE_HEADSET  headset;
  T_AUDIO_SPEAKER_MODE_AUX     aux;
  T_AUDIO_SPEAKER_MODE_CARKIT carkit;
}
T_AUDIO_SPEAKER_MODE;

typedef struct
{
   /* gain of the speaker */
  INT8    gain;
   /* use the audio filter */
  INT8  audio_filter;
   /* use the audio highpass filter */
    INT8  audio_highpass_filter;
   /* extra gain of the speaker */
  INT8   extra_gain;
    /* AGC parameter */
    T_AUDIO_AGC_DL_CFG   agc;
    /* DRC parameter */
    T_AUDIO_DRC_CFG   drc;
    /* IIR filter parameters */
    T_AUDIO_IIR_CFG       iir;
}
T_AUDIO_SPEAKER_MODE_HANDHELD;

typedef struct
{
  /* gain of the speaker */
  INT8    gain;
  /* use the audio filter */
  INT8  audio_filter;
   /* use the audio highpass filter */
    INT8  audio_highpass_filter;
  /* extra gain of the speaker */
  INT8   extra_gain;
    /* AGC parameter */
    T_AUDIO_AGC_DL_CFG   agc;
    /* DRC parameter */
    T_AUDIO_DRC_CFG   drc;
    /* IIR filter parameters */
    T_AUDIO_IIR_CFG    iir;

}
T_AUDIO_SPEAKER_MODE_HANDFREE;

typedef struct
{
  /* gain of the speaker */
  INT8    gain;
  /* use the audio filter */
  INT8  audio_filter;
   /* use the audio highpass filter */
    INT8  audio_highpass_filter;
    /* AGC parameter */
```

```
          T_AUDIO_AGC_DL_CFG   agc;
          /* DRC parameter */
          T_AUDIO_DRC_CFG   drc;
          /* IIR filter parameters */
          T_AUDIO_IIR_CFG   iir;
      }
      T_AUDIO_SPEAKER_MODE_HEADSET;

typedef struct
      {
        /* gain of the speaker */
        INT8    gain;
        /* use the audio filter */
        INT8  audio_filter;
         /* use the audio highpass filter */
          INT8  audio_highpass_filter;
          /* AGC parameter */
          T_AUDIO_AGC_DL_CFG   agc;
          /* DRC parameter */
          T_AUDIO_DRC_CFG   drc;
          /* IIR filter parameters */
          T_AUDIO_IIR_CFG   iir;

      }
      T_AUDIO_SPEAKER_MODE_AUX;

typedef struct
      {
        /* gain of the speaker */
        INT8    gain;
        /* use the audio filter */
        INT8  audio_filter;
         /* use the audio highpass filter */
          T_AUDIO_FIR_COEF    fir;
          /* AGC parameter */
          T_AUDIO_AGC_DL_CFG   agc;
          /* DRC parameter */
          T_AUDIO_DRC_CFG   drc;
          /* IIR filter parameters */
          T_AUDIO_IIR_CFG   iir;
      }
      T_AUDIO_SPEAKER_MODE_CARKIT;
```

IMPORTANT NOTE:

IIR feature is only present in TCS 3.x software except TCS 3.0. Other software versions do not include T_AUDIO_IIR_CFG structure.

As the IIR filter replaces the FIR filter, T_AUDIO_FIR_COEF isn't present in software versions supporting the IIR filter.

In TCS 3.2 the LIMITER is replaced by the AGC and DRC modules.

For parameters details:

T_AUDIO_AGC_DL_CFG: see 9.2.1.4.1

T_AUDIO_DRC_CFG: see 9.2.1.4.2

T_AUDIO_IIR_CFG: see 9.2.1.4.3

Mode
Specifies the mode of the microphone: **AUDIO_SPEAKER_MODE_HANDHELD**, AUDIO_SPEAKER_MODE_HANDFREE or AUDIO_SPEAKER_MODE_HEADSET or AUDIO_SPEAKER_MODE_AUX or AUDIO_SPEAKER_MODE_CARKIT.

All these modes are available in GSM, bluetooth cordless voice and all DAI path mode.

gain
Specifies the gain in 1 dB unit of the speaker. The range is from –6 dB to 6 dB.

audio_filter
Add an audio filter in the speaker path in order to enhance the audio quality. The filter is added if *audio_filter* = AUDIO_SPEAKER_FILTER_ON else the filter is bypassed if *audio_filter* = AUDIO_SPEAKER_FILTER_OFF. The frequency response of this hardware filter is the following:

Frequency Response	Gain relative to reference gain at 1kHz	Min	Typ	Max	Units
<= 100 Hz				-20	DB
100 Hz to 200 Hz				-10	dB
300 Hz to 400 Hz		-2	0	+1	dB
400 Hz to 3300 Hz		-1	0	+1	dB
3300 Hz to 3400 Hz		-2	0	+1	dB
4000 Hz to 4600 Hz				-17	dB
4600 Hz to 6000 Hz				-40	dB
>= 6000 Hz				-45	dB

WARNING: IF THE FILTER IS BYPASSED, THE GAIN IS EQUAL TO 0 AND THE VOLUME TOO (c.f. speaker volume API function).

audio_highpass_filter
Add or bypass the high-pass part of the audio filter: AUDIO_SPEAKER_HIGHPASS_FILTER_ON to add it, AUDIO_SPEAKER_HIGHPASS_FILTER_OFF to bypass it.

extra_gain
Extra gain for AUDIO_SPEAKER_HANDHELD and AUDIO_SPEAKER_HANDFREE modes.

For AUDIO_SPEAKER_HANDHELD the extra gain values are AUDIO_SPEAKER_SPK_GAIN_8_5DB, AUDIO_SPEAKER_SPK_GAIN_2_5DB, AUDIO_SPEAKER_SPK_GAIN_MINUS_3_5DB and AUDIO_SPEAKER_SPK_GAIN_MINUS_22_5DB.

For AUDIO_SPEAKER_HANDFREE the extra gain values are AUDIO_EAR_GAIN_MINUS_11DB and AUDIO_EAR_GAIN_1DB

agc
See 9.2.1.4.1 for details.

Drc
See 9.2.1.4.2 for details.

Iir
See 9.2.1.4.3 for details.

9.2.1.4.1 T_AUDIO_AGC_DL_CFG

The primary goal of AGC is to adjust the input speech signal to a targeted level at the output. AGC is used in both the uplink and downlink in the TCS3.2 acoustic chain.
AGC is only available from TCS 3.2. AGC module only works in DAI acoustic and GSM and Bluetooth headset path modes.
The following AGC settings are to be used for AGC 1.x.

AGC settings are inside following structure of the microphone settings:

```
typedef struct
{
  T_AGC_CONTROL          agc_dl_control;
   T_AUDIO_AGC_PARAMS    parameters ;
```

```
}
T_AUDIO_AGC_DL_CFG;

typedef enum
{
  ANR_STOP  = 0,
  ANR_START = 1,
  ANR_UPDATE = 2
}
T_ANR_CONTROL;

typedef struct
{
    UINT16  control;
    UINT16  frame_size;
    INT16   targeted_level;
    INT16   signal_up;
    INT16   signal_down;
    INT16   max_scale;
    INT16   gain_smooth_alpha;
    INT16   gain_smooth_alpha_fast;
    INT16   gain_smooth_beta;
    INT16   gain_smooth_beta_fast;
    INT16   gain_intp_flag;
}
T_AUDIO_AGC_PARAMS;
```

For details of each parameter, please refer to 9.2.1.3.2.

9.2.1.4.2 T_AUDIO_DRC_CFG

The primary goal of DRC is to increase the perceptual loudness at the loudspeaker. And hence the DRC is implemented on the RX path in the Digital Base Band (DBB). A dynamic range compression algorithm typically amplifies the quiet parts and attenuates loud parts of the input signal so that the final dynamic range of the output signal fits into a fixed range.

DRC module only works in DAI acoustic and GSM and Bluetooth headset path modes. The following DRC settings are to be used for DRC 1.x in the TCS 3.2 acoustic chain.

DRC settings are inside following structure of the speaker settings:

```
typedef struct
{
  T_DRC_CONTROL           drc_dl_control;
   T_AUDIO_DRC_PARAMS     parameters ;
}
T_AUDIO_DRC_CFG;

typedef enum
{
  DRC_STOP  = 0,
  DRC_START = 1,
  DRC_UPDATE = 2
}
T_DRC_CONTROL;

typedef struct
{
 UINT16 speech_mode_samp_f;
 INT16  num_subbands;
 INT16  frame_size;
 UINT16 expansion_knee_fb_bs;
 UINT16 expansion_knee_md_hg;
 UINT16 compression_knee_fb_bs;
 UINT16 compression_knee_md_hg;
 UINT16 expansion_ratio_fb_bs;
 UINT16 expansion_ratio_md_hg;
 UINT16 compression_ratio_fb_bs;
 UINT16 compression_ratio_md_hg;
 UINT16 max_amplification_fb_bs;
 UINT16 max_amplification_md_hg;
 UINT16 energy_limiting_th_fb_bs;
 UINT16 energy_limiting_th_md_hg;
 INT16  limiter_threshold_fb;
 INT16  limiter_threshold_bs;
 INT16  limiter_threshold_md;
 INT16  limiter_threshold_hg;
 UINT16 limiter_release_fb_bs;
 UINT16 limiter_release_md_hg;
 UINT16 limiter_hangover_spect_preserve;
 UINT16 gain_track_fb_bs;
 UINT16 gain_track_md_hg;
 INT16  low_pass_filter[17];
 INT16  mid_band_filter[17];
}
T_AUDIO_DRC_PARAMS;
```

Parameter Name	Type	Format		Range	Comments
speech_mode_samp_f	UINT16	MSB	8b/Q0	[0x00,	DRC bypassed
				0x01]	DRC enabled in speech mode[8]
		LSB	8b/Q0	[0x01,	8kHz sampling frequency

[8] The parameters and values listed are applicable to DRC 1.0 only. Hence only speech mode is available. Subsequent versions of DRC will support both speech and music and then the MSB of parameter drc_speech_mode_samp_f may take value 0x02 to indicate music.

				0x02]	16kHz sampling frequency[9]
In case of Read Access, the following parameters are valid only if speech_mode_samp_f is non-zero					
num_subbands	INT16		16b/Q0	[0x0001,	full-band processing
				0x0003]	three sub-bands processing
frame_size	INT16		16b/Q0	[0x0050,	80 samples, 1X10ms frame processing (FP) at 8kHz
				0x00A0,	160 samples, 2X10ms FP (8kHz), 1X10ms FP (16kHz)
				0x0140]	320 samples, 2X10ms FP (16kHz)
expansion_knee_fb_bs	UINT16	MSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
		LSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
expansion_knee_md_hg	UINT16	MSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
		LSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
compression_knee_fb_bs	UINT16	MSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
		LSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
compres-sion_knee_md_hg	UINT16	MSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
		LSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
expansion_ratio_fb_bs	UINT16	MSB	8b/Q2	[0x05…,	minimal: 1.25
				0x7F]	maximal: 31.75 0.25 steps
		LSB	8b/Q2	[0x05…,	minimal: 1.25
				0x7F]	maximal: 31.75 0.25 steps
expansion_ratio_md_hg	UINT16	MSB	8b/Q2	[0x05…,	minimal: 1.25
				0x7F]	maximal: 31.75 0.25 steps
		LSB	8b/Q2	[0x05…,	minimal: 1.25
				0x7F]	maximal: 31.75 0.25 steps
compression_ratio_fb_bs	UINT16	MSB	8b/Q2	[0x05…,	minimal: 1.25
				0x7F]	maximal: 31.75 0.25 steps
		LSB	8b/Q2	[0x05…,	minimal: 1.25
				0x7F]	maximal: 31.75 0.25 steps
compres-sion_ratio_md_hg	UINT16	MSB	8b/Q2	[0x05…,	minimal: 1.25
				0x7F]	maximal: 31.75 0.25 steps
		LSB	8b/Q2	[0x05…,	minimal: 1.25
				0x7F]	maximal: 31.75 0.25 steps
max_amplification_fb_bs	UINT16	MSB	8b/Q0	[0x00…,	minimal: 0dB
				0x12]	maximal: 18dB 1dB steps
		LSB	8b/Q0	[0x00…,	minimal: 0dB
				0x12]	maximal: 18dB 1dB steps
max_amplification_md_hg	UINT16	MSB	8b/Q0	[0x00…,	minimal: 0dB
				0x12]	maximal: 18dB 1dB steps
		LSB	8b/Q0	[0x00…,	minimal: 0dB
				0x12]	maximal: 18dB 1dB steps
energy_limiting_th_fb_bs	UINT16	MSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps

[9] Though DRC 1.0 has the capability to operate on apeech at 8 kHz and 16 kHz sampling frequencies, the modules of the TCS 3.2 voice acoustic chain currently operate at 8 kHz. Using DRC 1.0 in this chain limits its capability to 8 kHz. Hence the LSB of this parameter shall not use value 0x02 for TCS 3.2.

		LSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
en-ergy_limiting_th_md_hg	UINT16	MSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
		LSB	8b/Q0	[0x00…,	minimal: 0dB
				0x5A]	maximal: 90dB 1dB steps
limiter_threshold_fb	INT16		16b/Q0	[0x0001 …,	minimal: 1
				0x7FFF]	maximal: 32767 1 steps
limiter_threshold_bs	INT16		16b/Q0	[0x0001 …,	minimal: 1
				0x7FFF]	maximal: 32767 1 steps
limiter_threshold_md	INT16		16b/Q0	[0x0001 …,	minimal: 1
				0x7FFF]	maximal: 32767 1 steps
limiter_threshold_hg	INT16		16b/Q0	[0x0001 …,	minimal: 1
				0x7FFF]	maximal: 32767 1 steps
limiter_release_fb_bs	UINT16	MSB	index	[0x01,	500dB/sec (for high-crest), 200dB/sec (for low-crest)
				0x02,	200dB/sec (for high-crest), 50dB/sec (for low-crest)
				0x03,	100dB/sec (for high-crest), 15B/sec (for low-crest)
				0x04,	50dB/sec (for high-crest), 10dB/sec (for low-crest)
				0x05]	30dB/sec (for high-crest), 2dB/sec (for low-crest)
		LSB	index	[0x01…	Same as above
				0x05]	
limiter_release_md_hg	UINT16	MSB	index	[0x01…	Same as above
				0x05]	
		LSB	index	[0x01…	Same as above
				0x05]	
lim-iter_hangover_spect_preserve	UINT16	MSB	8b/Q0	[0x00…	minimal: 0 frame
				0x7F]	maximal: 255 frames
		LSB	8b/Q8	[0x00..	minimal: 0.0
				0x7F]	maximal: 1.0
gain_track_fb_bs	UINT16	MSB	index	[0x01,	instantaneous gain tracking
				0x02,	fast gain tracking
				0x03,	medium gain tracking
				0x04,	gain tracking for speech
				0x05,	slow gain tracking
				0x06]	very slow gain tracking
		LSB	index	[0x01…	Same as above
				0x06]	
gain_track_md_hg	UINT16	MSB	index	[0x01…	Same as above
				0x06]	
		LSB	index	[0x01…	Same as above
				0x06]	
low_pass_filter[17]	INT16 [17]		index	coeffi-cients	low pass filter
mid_band_filter[17]	INT16 [17]		index	coeffi-cients	band pass filter

9.2.1.4.3 T_AUDIO_IIR_CFG

The primary goal of the IIR 4.x is to equalize the Receiving Frequency Response (RFR) and the Sending Frequency Response (SFR) of the User Equipment (UE) to fit in 3GPP Full-Type Agreement (FTA) tests cases. IIR is used the downlink in the TCS3.2 acoustic chain and is used for the equalization of the RFR only.

IIR module only works in DAI acoustic and GSM and Bluetooth headset path modes.
The following IIR settings are to be used for IIR 4.x used in the TCS 3.2 acoustic chain.

IIR settings are inside following structure of the speaker settings:

```
typedef struct
{
    T_IIR_CONTROL            iir_dl_control;
    T_AUDIO_IIR_PARAMS       parameters ;
}
T_AUDIO_IIR_CFG;

typedef enum
{
    IIR_STOP  = 0,
    IIR_START = 1,
    IIR_UPDATE = 2
}
T_IIR_CONTROL;

typedef struct
   {
     UINT16 control;
     UINT16 frame_size;
     UINT16 fir_swap;
     T_AUDIO_IIR_FIR_PARAMS fir_filter;
     T_AUDIO_IIR_SOS_PARAMS sos_filter;
     INT16 gain;
   }
   T_AUDIO_IIR_PARAMS;

typedef struct
   {
      UINT16 fir_enable;
      UINT16 fir_length;
      INT16  fir_shift;
      INT16  fir_taps[40];
   }
   T_AUDIO_IIR_FIR_PARAMS;

typedef struct
   {

     UINT16 sos_enable;
     UINT16 sos_number;
     T_AUDIO_IIR_SINGLE_SOS_PARAMS sos_filter[6];
   }
   T_AUDIO_IIR_SOS_PARAMS;

   typedef struct
   {
      INT16 sos_fact;
      INT16 sos_fact_form;
      INT16 sos_den[2];
      INT16 sos_num[3];
      INT16 sos_num_form;
   }
   T_AUDIO_IIR_SINGLE_SOS_PARAMS;
```

T_AUDIO_IIR_CFG				
Variable Name	**Type**	**Format**	**Range**	**Comments**
control	UINT16	Q0	0x0000	Disable
			0x0001	Enable 8kHz
			0x0002	Enable 16kHz[10]
In case of Read Access, the following parameters are valid only if control is non-zero				
frame_size	UINT16	Q0	0x0050	80 samples (10ms@ 8kHz)
			0x00A0	160 samples (2x10ms @ 8kHz, 10ms @16kHz)
			0x0140	320 samples (2x10ms @16kHz)
fir_swap	UINT16	Q0	0x0001	FIR acting before IIR (SOS)
			0x0002	IIR SOS acting before FIR
fir_filter	T_AUDIO_IIR_FIR_PARAMS	structure	see below	see below
sos_filter	T_AUDIO_IIR_SOS_PARAMS	structure	see below	see below
gain	INT16	Q13	0x0000	Digital gain OFF (bypass)
			[0x0001, 0x7FFF]	Digital gain ON [-78dB,…+12dB].

T_AUDIO_IIR_FIR_PARAMS				
fir_enable	UINT16	Q0	0x0000	FIR Disable
			0x0001	FIR Enable
fir_length	UINT16	Q0	[0x0001, …0x0028]	Min = 1 tap
				Max = 40 taps, $N = 40$.
fir_shift	INT16	Q0	[0x0000, 0x7FFF]	$Q(\boldsymbol{H}_N)$.
fir_taps[40]	INT16	$Q(\boldsymbol{H}_N)$	[0x8000, 0x7FFF]	$\{h_i\}, 0 \le i \le N-1.$

T_AUDIO_IIR_SOS_PARAMS				
sos_enable	UINT16	Q0	0x0000	IIR (SOS) disable
			0x0001	IIR (SOS) enable
sos_number	UINT16	Q0	[0x0001, …0x0006]	Min = 1 SOS (biquad)
				Max = 6 SOS (biquads), $K = 6$.
sos_params[6]	T_AUDIO_IIR_SINGLE_SOS_PARAMS	structure		

T_AUDIO_IIR_SINGLE_SOS_PARAMS				
sos_fact	INT16	$Q(\boldsymbol{G}_k^{'})$	[0x8000, 0x7FFF]	$\{G_k^{'}\}, 1 \le k \le K.$
sos_fact_form	INT16	Q0	[0x0000, 0x7FFF]	$Q(\boldsymbol{G}_k^{'})$.
sos_den[2]	INT16	Q14	[0x8000, 0x7FFF]	SOS denominators (b11,...b26)

[10] Though IIR 4.x has the capability to operate at 8 kHz and 16 kHz sampling frequencies, using IIR 4.x in the TCS3.2 acoustic chain limits it to 8 kHz. So in theis chain, IIR 4.x can be set to operate at ONLY at 8 kHz. Hence parameter iir4x _control shall not use value 0x0002 for TCS 3.2

sos_num[3]	INT16	$Q(\boldsymbol{a}_k)$	[0x8000, 0x7FFF]	SOS numerators (a01,...a21) -> (a06,...a26).
sos_num_form	INT16	Q0	0x0000, 0x7FFF]	$Q(\boldsymbol{a}_k)$.

- For other configurations :

```
typedef struct
{
  /* mode of the speaker */
  INT8                      mode;
  /* Setting of the current mode */
  T_AUDIO_SPEAKER_MODE      setting;
}
T_AUDIO_SPEAKER_SETTING;
```

where the speaker modes are:

```
typedef union
{
  /* handheld mode parameters */
  T_AUDIO_SPEAKER_MODE_HANDHELD handheld;
  /* handfree mode parameters */
  T_AUDIO_SPEAKER_MODE_HANDFREE handfree;
  /* headset mode parameters */
  T_AUDIO_SPEAKER_MODE_HEADSET  headset;
  /* buzzer mode parameters */
  T_AUDIO_SPEAKER_MODE_BUZZER   buzzer;
  /* handheld and handfree mode parameters */
  T_AUDIO_SPEAKER_MODE_HANDHELD_HANDFREE handheld_handfree;
  /* handheld and handfree mode parameters */
  T_AUDIO_SPEAKER_MODE_HANDHELD_8OHM     handheld_8ohm;
  /* handheld and handfree mode parameters */
  T_AUDIO_SPEAKER_MODE_HANDFREE_8OHM     handfree_8ohm;
}
T_AUDIO_SPEAKER_MODE;

typedef struct
{
  /* gain of the speaker */
  INT8  gain;
  /* use the audio filter */
  INT8  audio_filter;
#if (ANALOG == 3)
  /* use the audio highpass filter */
  INT8  audio_highpass_filter;
#endif
  /* coefficients of the speaker FIR */
  T_AUDIO_FIR_COEF    fir;
  /* Limiter parameters */
  T_AUDIO_LIMITER_CFG limiter;
  /* IIR filter parameters */
  T_AUDIO_IIR_CFG     iir;      }
T_AUDIO_SPEAKER_MODE_HANDHELD;

typedef struct
{
  /* gain of the speaker */
  INT8   gain;
  /* use the audio filter */
  INT8   audio_filter;
#if (ANALOG == 3)
  /* use the audio highpass filter */
  INT8  audio_highpass_filter;
#endif
  /* coefficients of the speaker FIR */
  T_AUDIO_FIR_COEF    fir;
  /* Limiter parameters */
  T_AUDIO_LIMITER_CFG limiter;
  /* IIR filter parameters */
  T_AUDIO_IIR_CFG     iir;
}
T_AUDIO_SPEAKER_MODE_HANDFREE;

typedef struct
{
  /* gain of the speaker */
  INT8   gain;
  /* use the audio filter */
```

```
  INT8   audio_filter;
#if (ANALOG == 3)
  /* use the audio highpass filter */
  INT8  audio_highpass_filter;
#endif
  /* coefficients of the speaker FIR */
  T_AUDIO_FIR_COEF    fir;
  /* Limiter parameters */
  T_AUDIO_LIMITER_CFG limiter;
  /* IIR filter parameters */
  T_AUDIO_IIR_CFG     iir;
}
T_AUDIO_SPEAKER_MODE_HEADSET;

typedef struct
{
  /* activate the buzzer */
  INT8  activate;
}
T_AUDIO_SPEAKER_MODE_BUZZER;

typedef struct
{
  /* gain of the speaker */
  INT8   gain;
  /* use the audio filter */
  INT8   audio_filter;
#if (ANALOG == 3)
  /* use the audio highpass filter */
  INT8  audio_highpass_filter;
#endif
  /* coefficients of the speaker FIR */
  T_AUDIO_FIR_COEF    fir;
  /* Limiter parameters */
  T_AUDIO_LIMITER_CFG limiter;
  /* IIR filter parameters */
  T_AUDIO_IIR_CFG     iir;
}
T_AUDIO_SPEAKER_MODE_HANDHELD_HANDFREE;

typedef struct
{
  /* gain of the speaker */
  INT8   gain;
  /* extra gain of the speaker */
  INT8   extra_gain;
  /* use the audio filter */
  INT8   audio_filter;
  /* use the audio highpass filter */
  INT8  audio_highpass_filter;
  /* coefficients of the speaker FIR */
  T_AUDIO_FIR_COEF    fir;
  /* Limiter parameters */
  T_AUDIO_LIMITER_CFG limiter;
  /* IIR filter parameters */
  T_AUDIO_IIR_CFG     iir;
}
T_AUDIO_SPEAKER_MODE_HANDHELD_8OHM;

typedef T_AUDIO_SPEAKER_MODE_HANDHELD_8OHM T_AUDIO_SPEAKER_MODE_HANDFREE_8OHM;
```

IMPORTANT NOTE:

IIR and LIMITER features are only present in TCS 3.x software except TCS 3.0. Other software versions do not include T_AUDIO_LIMITER_CFG and T_AUDIO_IIR_CFG structures.

As the IIR filter replaces the FIR filter, T_AUDIO_FIR_COEF isn't present in software versions supporting the IIR filter.

For parameters details:

T_AUDIO_LIMITER_CFG: see 9.2.1.4.4

T_AUDIO_IIR_CFG: see 9.2.1.4.5

Mode
Specifies the mode of the microphone: AUDIO_SPEAKER_HANDHELD, AUDIO_SPEAKER_HANDFREE or AUDIO_SPEAKER_HEADSET or AUDIO_SPEAKER_BUZZER or AUDIO_SPEAKER_HANDHELD_HANDFREE or AUDIO_SPEAKER_HANDHELD_8OHM/AUDIO_SPEAKER_HANDFREE_8OHM.

AUDIO_SPEAKER_HANDHELD mode: this mode is available in GSM, bluetooth cordless voice and all DAI path mode.

gain
Specifies the gain in 1 dB unit of the speaker. The range is from –6 dB to 6 dB.

audio_filter
Add an audio filter in the speaker path in order to enhance the audio quality. The filter is added if *audio_filter* = AUDIO_SPEAKER_FILTER_ON else the filter is bypassed if *audio_filter* = AUDIO_SPEAKER_FILTER_OFF. The frequency response of this hardware filter is the following:

Frequency Response	Gain relative to reference gain at 1kHz	Min	Typ	Max	Units
<= 100 Hz				-20	DB
100 Hz to 200 Hz				-10	dB
300 Hz to 400 Hz		-2	0	+1	dB
400 Hz to 3300 Hz		-1	0	+1	dB
3300 Hz to 3400 Hz		-2	0	+1	dB
4000 Hz to 4600 Hz				-17	dB
4600 Hz to 6000 Hz				-40	dB
>= 6000 Hz				-45	dB

WARNING: IF THE FILTER IS BYPASSED, THE GAIN IS EQUAL TO 0 AND THE VOLUME TOO (c.f. speaker volume API function).

audio_highpass_filter (only available with analog base band SYREN)
Add or bypass the high-pass part of the audio filter: AUDIO_SPEAKER_HIGHPASS_FILTER_ON to add it, AUDIO_SPEAKER_HIGHPASS_FILTER_OFF to bypass it.

fir_coef
List of the coefficient of the FIR of the speaker. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000.
Note: the FIR is available only in DAI and GSM path mode.
The FIR filter is replaced by IIR filter in TCS3.x software, except TCS 3.0.

limiter
See 9.2.1.4.4 for details.

Iir
See 9.2.1.4.5 for details.

SPEAKER_HANDFREE mode: this mode is only available in GSM, bluetooth cordless voice and all DAI path mode.

Gain
Specifies the gain in 1 dB unit of the speaker. The range is from –6 dB to 6 dB.

audio_filter
Add an audio filter in the speaker path in order to enhance the audio quality. The filter is added if *audio_filter* = AUDIO_SPEAKER_FILTER_ON else the filter is bypassed if *audio_filter* = AUDIO_SPEAKER_FILTER_OFF. The frequency response of this hardware filter is the following:

Frequency Response	Gain relative to reference gain at 1kHz	Min	Typ	Max	Units
<= 100 Hz				-20	DB
100 Hz to 200 Hz				-10	DB
300 Hz to 400 Hz		-2	0	+1	DB
400 Hz to 3300 Hz		-1	0	+1	DB
3300 Hz to 3400 Hz		-2	0	+1	DB

	4000 Hz to 4600 Hz				-17	DB
	4600 Hz to 6000 Hz				-40	DB
	>= 6000 Hz				-45	DB
	WARNING: IF THE FILTER IS BYPASSED, THE GAIN IS EQUAL TO 0 AND THE VOLUME TOO (c.f. speaker volume API function).					
	audio_highpass_filter (only available with analog base band SYREN) Add or bypass the high-pass part of the audio filter: AUDIO_SPEAKER_HIGHPASS_FILTER_ON to add it, AUDIO_SPEAKER_HIGHPASS_FILTER_OFF to bypass it.					
	fir_coef List of the coefficient of the FIR of the speaker. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.** **The FIR filter is replaced by IIR filter in TCS3.x software, except TCS 3.0.**					
	limiter *See 9.2.1.4.4for details.*					
	Iir *See 9.2.1.4.5 for details.*					
SPEAKER_HEADSET mode: this mode is only available in GSM, bluetooth cordless voice and all DAI path mode. It is only available with the analog base band IOTA and SYREN.						
	Gain Specifies the gain in 1 dB unit of the speaker. The range is from –6 dB to 6 dB.					
	audio_filter Add an audio filter in the speaker path in order to enhance the audio quality. The filter is added if *audio_filter* = AUDIO_SPEAKER_FILTER_ON else the filter is bypassed if *audio_filter* = AUDIO_SPEAKER_FILTER_OFF. The frequency response of this hardware filter is the following:					
	Frequency Response	**Gain relative to reference gain at 1kHz**	**Min**	**Typ**	**Max**	**Units**
	<= 100 Hz				-20	DB
	100 Hz to 200 Hz				-10	DB
	300 Hz to 400 Hz		-2	0	+1	DB
	400 Hz to 3300 Hz		-1	0	+1	DB
	3300 Hz to 3400 Hz		-2	0	+1	DB
	4000 Hz to 4600 Hz				-17	DB
	4600 Hz to 6000 Hz				-40	DB
	>= 6000 Hz				-45	DB
	WARNING: IF THE FILTER IS BYPASSED, THE GAIN IS EQUAL TO 0 AND THE VOLUME TOO (c.f. speaker volume API function).					
	audio_highpass_filter (only available with analog base band SYREN) Add or bypass the high-pass part of the audio filter: AUDIO_SPEAKER_HIGHPASS_FILTER_ON to add it, AUDIO_SPEAKER_HIGHPASS_FILTER_OFF to bypass it.					
	fir_coef List of the coefficient of the FIR of the speaker. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000. **Note: the FIR is available only in DAI and GSM path mode.** **The FIR filter is replaced by IIR filter in TCS3.x software, except TCS 3.0.**					
	limiter *See 9.2.1.4.4 for details.*					
	Iir *See 9.2.1.4.5 details.*					
SPEAKER_BUZZER mode: this mode is only available in GSM, bluetooth cordless voice and all DAI path mode. It is only available with the analog base band NAUSICA-OMEGA.						
	activate Specifies if the buzzer is activated (AUDIO_SPEAKER_BUZZER_ON) or not (AUDIO_SPEAKER_BUZZER_OFF).					
AUDIO_SPEAKER_HANDHELD_HANDFREE mode: this mode is available in GSM, bluetooth cordless voice and all DAI path mode.						
	gain Specifies the gain in 1 dB unit of the speaker. The range is from –6 dB to 6 dB.					

audio_filter
Add an audio filter in the speaker path in order to enhance the audio quality. The filter is added if *audio_filter* = AUDIO_SPEAKER_FILTER_ON else the filter is bypassed if *audio_filter* = AUDIO_SPEAKER_FILTER_OFF. The frequency response of this hardware filter is the following:

Frequency Response	Gain relative to reference gain at 1kHz	Min	Typ	Max	Units
<= 100 Hz				-20	dB
100 Hz to 200 Hz				-10	dB
300 Hz to 400 Hz		-2	0	+1	dB
400 Hz to 3300 Hz		-1	0	+1	dB
3300 Hz to 3400 Hz		-2	0	+1	dB
4000 Hz to 4600 Hz				-17	dB
4600 Hz to 6000 Hz				-40	dB
>= 6000 Hz				-45	dB

WARNING: IF THE FILTER IS BYPASSED, THE GAIN IS EQUAL TO 0 AND THE VOLUME TOO (c.f. speaker volume API function).

audio_highpass_filter (only available with analog base band SYREN)
Add or bypass the high-pass part of the audio filter: AUDIO_SPEAKER_HIGHPASS_FILTER_ON to add it, AUDIO_SPEAKER_HIGHPASS_FILTER_OFF to bypass it.

fir_coef
List of the coefficient of the FIR of the speaker. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000.
Note: the FIR is available only in DAI and GSM path mode.
The FIR filter is replaced by IIR filter in TCS3.x software except TCS 3.0.

limiter
See 9.2.1.4.4 for details.

Iir
See 9.2.1.4.5 for details.

AUDIO_SPEAKER_HANDHELD_8OHM, AUDIO_SPEAKER_HANDFREE_8OHM mode: this mode is available in GSM, Bluetooth cordless voice and all DAI path mode. It is only available with the analog base band SYREN.

gain
Specifies the gain in 1 dB unit of the speaker. The range is from –6 dB to 6 dB.

extra_gain
Specifies an additional gain of the speaker 8ohm path: 2.5 dB (AUDIO_SPEAKER_SPK_GAIN_2_5dB) or 8.5 dB (AUDIO_SPEAKER_SPK_GAIN_8_5dB).

audio_filter
Add an audio filter in the speaker path in order to enhance the audio quality. The filter is added if *audio_filter* = AUDIO_SPEAKER_FILTER_ON else the filter is bypassed if *audio_filter* = AUDIO_SPEAKER_FILTER_OFF. The frequency response of this hardware filter is the following:

Frequency Response	**Gain relative to reference** gain at 1kHz	Min	Typ	Max	Units
<= 100 Hz				**-20**	**dB**
100 Hz to 200 Hz				**-10**	**dB**
300 Hz to 400 Hz		**-2**	**0**	**+1**	**dB**
400 Hz to 3300 Hz		**-1**	**0**	**+1**	**dB**
3300 Hz to 3400 Hz		**-2**	**0**	**+1**	**dB**
4000 Hz to 4600 Hz				**-17**	**dB**
4600 Hz to 6000 Hz				**-40**	**dB**
>= 6000 Hz				**-45**	**dB**

WARNING: IF THE FILTER IS BYPASSED, THE GAIN IS EQUAL TO 0 AND THE VOLUME TOO (c.f. speaker volume API function).

audio_highpass_filter (only available with analog base band SYREN)
Add or bypass the high-pass part of the audio filter: AUDIO_SPEAKER_HIGHPASS_FILTER_ON to add it, AUDIO_SPEAKER_HIGHPASS_FILTER_OFF to bypass it.

fir_coef
List of the coefficient of the FIR of the speaker. The format of each coefficient is F2.14. For example: 0,5 = 0x2000, 1 = 0x4000 and –1=0xc000.
Note: the FIR is available only in DAI and GSM path mode.
The FIR filter is replaced by IIR filter in TCS3.x software except TCS 3.0.

	limiter *See 9.2.1.4.4 for details.*
	Iir *See 9.2.1.4.5 for details.*

9.2.1.4.4 T_AUDIO_LIMITER_CFG

Limiter aim is to avoid using the non-linear regions of the speaker response in order to avoid audio saturation/distortion. Please refer to [5] for an overview of the module,

Limiter is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the speaker settings.
Limiter module only works in DAI acoustic and GSM path mode.

Limiter settings are inside following structure of the speaker settings:

```
typedef struct
{
  BOOLEAN  limiter_enable;
  UINT16   block_size;
  UINT16   slope_update_period;
  UINT16   nb_fir_coefs;
  INT16    filter_coefs[16];
  UINT16   thr_low_0;
  INT16    thr_low_slope;
  UINT16   thr_high_0;
  INT16    thr_high_slope;
  INT16    gain_fall;
  INT16    gain_rise;
}
T_AUDIO_LIMITER_CFG;
```

limiter_enable
Enable/disable the limiter
0- disable
1- enable

In case of Read Access, the following parameters are valid only if limiter_enable = 1.

block_size
Number of samples in an input block.
Currently, mandatory value is 160.

slope_update_period
Number of samples between each update of the limiter slope. It must be a divider of block_size.
Recommended value is 160.

nb_fir_coefs
Number of coefficients in the filter. It must be an odd number. Maximum number is 31.
Recommended value is 31.

filter_coefs
Array containing the filter coefficients. This array must contains (nb_fir_coefs-1)/2+1 coefficients. The filter being a symmetric one, other coefficients do not need to be saved in the array.

thr_low_0 / thr_low_slope
thr_high_0 / thr_high_slope

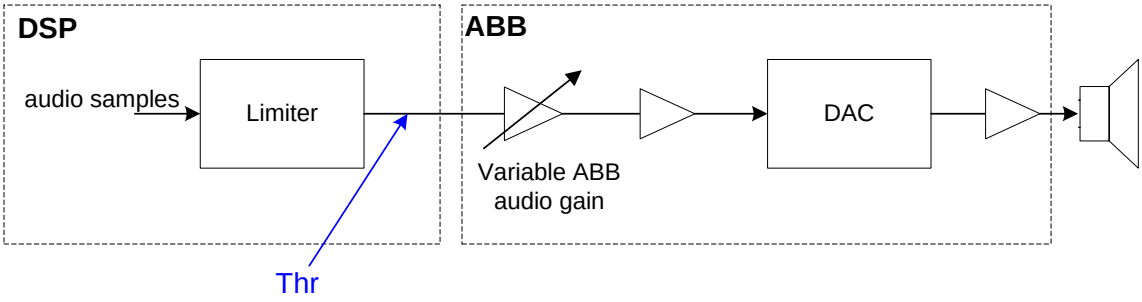
thr_X_0 (range 0..32767)
percentage of the maximum level of signal wanted at the output of the limiter with respect to the maximum possible level set to 1. It has to be multiplied by 32767 to be expressed with only 1 bit significant for integer part and 15 for decimal part.

thr_X_slope (range -30..+6)
Slope threshold above which signal has to be decreased. It is expressed in dB.
Minimum and maximum values depend on possible values for voiceband downlink control register.

These values permit to define Thr(low) and Thr(high) characteristic function to volume setup in the ABB.
Thr(low) and Thr(high) define the maximum level of the signal wanted at the output of the **limiter**:

- Thr(low) for the low frequency part of the signal
- Thr(high) for the high frequency part of the signal
-

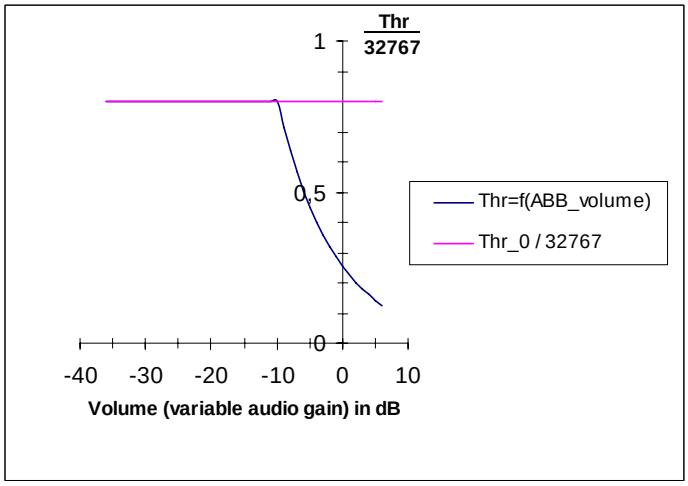
: Following scheme shows a model of the audio patch after the limiter:



Thr(low) or Thr(high) are processed using the following characteristic:

- For $Volume_{(dB)} <= thr\_low\_slope_{(dB)}$,
  $Thr(low) = thr\_low\_0$
- For $Volume_{(dB)} > thr\_low\_slope_{(dB)}$,
  $Thr(low) = thr\_low\_0 \times thr\_low\_slope_{(lin)} / Volume_{(lin)}$
  or
  $Thr(low) = thr\_low\_0 \times 10^{(thr\_low\_slope(dB) - Volume(dB))/20}$

- For $Volume_{(dB)} <= thr\_high\_slope_{(dB)}$,
  $Thr(high) = thr\_high\_0$
- For $Volume_{(dB)} > thr\_high\_slope_{(dB)}$,
  $Thr(high) = thr\_high\_0 \times thr\_high\_slope_{(lin)} / Volume_{(lin)}$
  or
  $Thr(high) = thr\_high\_0 \times 10^{(thr\_high\_slope(dB) - Volume(dB))/20}$



NB:

The name of the thr_low_slope on the L1 code is thr_ABB_vol_low.
The name of the thr_high_slope on the L1 code is thr_ABB_vol_high.

gain_fall
Decrease the slope when saturation has been detected on the previous block of samples. Format is Q15.
Recommended value is 26214
Ex: gain_fall = 26214 => slope(n+1) = 0.8 * slope(n)

gain_rise
Increase the slope when no saturation has been detected on the previous block of samples. The top limit of the slope is 1. Format is Q15 and 1 is added to get the coefficient of the multiplication.
Recommended value is 655
Ex: gain_rise = 655 => slope(n+1) = max(1, 1.02 * slope(n))

Information about Limiter parameters setting can be found in [6].

9.2.1.4.5 T_AUDIO_IIR_CFG

IIR filter replaces the FIR filter by having best performances using fewer coefficients. The aim of the IIR filter is to compensate the speaker frequency response in order to fit in ETSI requirements. Please refer to [3] for an overview of the module,

IIR filter is only available in TCS 3.x software except TCS 3.0. Other software versions do not include this structure in the speaker settings. When IIR is supported, the FIR filter isn't used so the FIR coefficients aren't included in the speaker settings.
IIR module only works in DAI acoustic and GSM path mode.

IIR settings are inside following structure of the speaker settings:

```
typedef struct
{
  BOOLEAN  iir_enable;
  UINT8    nb_iir_blocks;
  INT16    iir_coefs[80];
  UINT8    nb_fir_coefs;
  INT16    fir_coefs[32];
  INT8     input_scaling;
  INT8     fir_scaling;
  INT8     input_gain_scaling;
  INT8     output_gain_scaling;
  UINT16   output_gain;
  INT16    feedback;
}
T_AUDIO_IIR_CFG;
```

iir_enable
Enable/disable the IIR filter
0- disable
1- enable

In case of Read Access, the following parameters are valid only if iir_enable = 1.

nb_iir_blocks
Number of blocks for the given implementation of the systolic lattice IIR filter.
Value can be:
0 Recursive filtering part is disabled
1 Forbidden
[4:6] Number of IIR blocks

iir_coefs
Array containing the coefficients of the IIR lattice filter. There are 8 coefficients per block. The coefficients are generated by the MATLAB script sections.m. See [4] for more information.

nb_fir_coefs
Number of coefficients for the FIR (degree+1 of the FIR polynomial). Thus number must be greater or equal to 2 or the FIR filtering will be removed. It must be **lower or equal to 32**.

fir_coefs
Array containing the coefficients of the FIR filter. First coefficient (index 0) is the last coefficient corresponding to the term of higher degree in the FIR polynomial.

input_scaling Used to scale the input at entry of the IIR to avoid overflows inside the IIR. Note that the output of the filter (after the FIR) is scaled in the opposite way to compensate for this initial scaling. The scaling is in $[-16,15]$ range.
fir_scaling Used to scale the output of the IIR before using the FIR to avoid any scaling in the FIR. The output of the filter is scaled in the opposite way to compensate for this temporary scaling. The scaling is in $[-16,15]$ range.
input_gain_scaling The scaling factor applied at input of the filter for the global gain. Useful if the gain to implement is lower than 1 and if there are some overflows in the filter. The scaling is in $[-16,15]$ range.
output_gain_scaling Scaling factor at the output of the filter for the gain. Useful if the global gain to implement is higher than 1. The scaling is in $[-16,15]$ range.
output_gain A gain between [0,2[to tune the value of the global gain applied by the module. Format is unsigned Q15.
feedback Used to tune the rounding noise of the IIR implementation and to remove the bias. This value is filter dependent and should be tuned for a given set of IIR coefficients. Format is Q15.

Information about IIR parameters setting can be found in [4].

9.2.1.5 T_AUDIO_STEREO_SPEAKER_SETTING

Specifies the characteristic of the speaker audio stereo path.

- For analog base band TRITON:

Specifies the characteristic of the speaker audio stereo path.

```
typedef struct
{
  /* mode of the speaker */
  INT8                          mode;
  /* Setting of the current mode */
  T_AUDIO_STEREO_SPEAKER_MODE setting;
}
T_AUDIO_STEREO_SPEAKER_SETTING;

typedef union
{
  /* headphone mode parameters */
  T_AUDIO_STEREO_SPEAKER_MODE_HEADPHONE headphone;
  /* handheld mode parameters */
  T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD  handheld;
  /* handfree mode parameters */
  T_AUDIO_STEREO_SPEAKER_MODE_HANDFREE  handfree;
  T_AUDIO_STEREO_SPEAKER_MODE_AUX      aux;
  T_AUDIO_STEREO_SPEAKER_MODE_CARKIT   carkit;
}
T_AUDIO_STEREO_SPEAKER_MODE;

typedef struct
{
  /* stereo/mono configuration of the speaker */
  INT8   stereo_mono;
  /* sampling rate frequency */
  INT8   sampling_frequency;
}
T_AUDIO_STEREO_SPEAKER_MODE_HEADPHONE;

typedef struct
{
  /* sampling rate frequency */
  INT8   sampling_frequency;
}
T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD;

typedef struct
{
  /* sampling rate frequency */
  INT8   sampling_frequency;
}
T_AUDIO_STEREO_SPEAKER_MODE_HANDFREE;

typedef struct
{
  /* sampling rate frequency */
  INT8   sampling_frequency;
}
T_AUDIO_STEREO_SPEAKER_MODE_AUX;

typedef struct
{
  /* stereo/mono configuration of the speaker */
  INT8   stereo_mono;
  /* sampling rate frequency */
  INT8   sampling_frequency;
}
T_AUDIO_STEREO_SPEAKER_MODE_CARKIT;
```

Mode
Specifies the mode of the microphone: **AUDIO_STEREO_SPEAKER_MODE_HEADPHONE**, **AUDIO_STEREO_SPEAKER_MODE_HANDHELD**, **AUDIO_STEREO_SPEAKER_MODE_HANDFREE**, **AUDIO_STEREO_SPEAKER_MODE_AUX**,

AUDIO_STEREO_SPEAKER_MODE_CARKIT	
AUDIO_STEREO_SPEAKER_MODE_HEADPHONE ***mode: this mode is only available with the analog base band SYREN.***	
	stereo_mono: Specifies the possible stereo-mono conversion: - AUDIO_STEREO (no conversion) - AUDIO_MONO_LEFT (convert to mono and transmit on left channel) - AUDIO_MONO_RIGHT (convert to mono and transmit on right channel) - AUDIO_MONO_LEFT | AUDIO_MONO_RIGHT (convert to mono and transmit on both channels)
	sampling_frequency: Specifies the audio stereo sampling rate frequency - AUDIO_STEREO_SAMPLING_FREQUENCY_48KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_44_1KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_32KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_24KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_22_05KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_16KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_12KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_11_025KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_8KHZ **WARNING:** sampling frequency can not be changed after PLL power on.
AUDIO_STEREO_SPEAKER_MODE_HANDHELD ***mode:***	
	sampling_frequency: See above
	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.
AUDIO_STEREO_SPEAKER_MODE_HANDFREE ***mode:***	
	sampling_frequency: See above
	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.
AUDIO_STEREO_SPEAKER_MODE_AUX ***mode:***	
	sampling_frequency: See above
	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.
AUDIO_STEREO_SPEAKER_MODE_CARKIT ***mode:***	
	stereo_mono: See above
	sampling_frequency: See above

- For other configurations

```
typedef struct
{
  /* mode of the speaker */
  INT8                          mode;
  /* Setting of the current mode */
  T_AUDIO_STEREO_SPEAKER_MODE setting;
}
T_AUDIO_STEREO_SPEAKER_SETTING;
```

where the speaker modes are:

```
typedef struct
{
  /* stereo/mono configuration of the speaker */
  INT8   stereo_mono;
  /* sampling rate frequency */
  INT8   sampling_frequency;
}
T_AUDIO_STEREO_SPEAKER_MODE_HEADPHONE;

typedef struct
{
  /* sampling rate frequency */
  INT8   sampling_frequency;
}
```

```
      T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD;

      typedef struct
      {
        /* sampling rate frequency */
        INT8   sampling_frequency;
      }
      T_AUDIO_STEREO_SPEAKER_MODE_HANDFREE;

      typedef struct
      {
        /* sampling rate frequency */
        INT8   sampling_frequency;
      }
      T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD_8OHM;

      typedef T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD_8OHM
T_AUDIO_STEREO_SPEAKER_MODE_HANDFREE_8OHM;

      typedef union
      {
        /* headphone mode parameters */
        T_AUDIO_STEREO_SPEAKER_MODE_HEADPHONE headphone;
        /* handheld mode parameters */
        T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD  handheld;
        /* handfree mode parameters */
        T_AUDIO_STEREO_SPEAKER_MODE_HANDFREE  handfree;
        /* handheld 8ohm mode parameters */
        T_AUDIO_STEREO_SPEAKER_MODE_HANDHELD_8OHM handheld_8ohm;
        /* handfree 8ohm mode parameters */
        T_AUDIO_STEREO_SPEAKER_MODE_HANDFREE_8OHM handfree_8ohm;
      }
      T_AUDIO_STEREO_SPEAKER_MODE;
```

Mode Specifies the mode of the microphone: AUDIO_STEREO_SPEAKER_HEADPHONE, AUDIO_STEREO_SPEAKER_HANDHELD, AUDIO_STEREO_SPEAKER_HANDFREE, AUDIO_STEREO_SPEAKER_HANDHELD/HANDFREE_8OHM	
AUDIO_STEREO_SPEAKER_HEADPHONE mode: this mode is only available with the analog base band SYREN.	
	stereo_mono: Specifies the possible stereo-mono conversion: - AUDIO_STEREO (no conversion) - AUDIO_MONO_LEFT (convert to mono and transmit on left channel) - AUDIO_MONO_RIGHT (convert to mono and transmit on right channel) - AUDIO_MONO_LEFT | AUDIO_MONO_RIGHT (convert to mono and transmit on both channels)
	sampling_frequency: Specifies the audio stereo sampling rate frequency - AUDIO_STEREO_SAMPLING_FREQUENCY_48KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_44_1KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_32KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_22_05KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_16KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_11_025KHZ - AUDIO_STEREO_SAMPLING_FREQUENCY_8KHZ **WARNING:** sampling frequency can not be changed after PLL power on.
AUDIO_STEREO_SPEAKER_HANDHELD mode: this mode is only available with the analog base band SYREN.	
	sampling_frequency: See above
	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.
AUDIO_STEREO_SPEAKER_HANDFREE mode: this mode is only available with the analog base band SYREN.	
	sampling_frequency: See above

	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.
AUDIO_STEREO_SPEAKER_HANDHELD/HANDFREE_8OHM mode: this mode is only available with the analog base band SYREN.	
	sampling_frequency: See above
	Note: stereo-mono conversion is set to AUDIO_MONO_LEFT.

9.2.1.6 T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING

Specifies the characteristic of the features involved in the loop between the speaker and the microphone. **For DSP codes >= 33, there is a new version of AEC called NEW AEC**.

- For P2 samples with non TI audio ABB :

```
typedef struct
{
  /* gain of the sidetone */
  INT16                         sidetone_gain;
  /* configuration of the acoustic echo cancellation */
  T_AUDIO_AEC_CFG               aec;
}
T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING;
```

- For TRITON ABB:

```
typedef struct
{
  /* gain of the sidetone */
  INT16                         sidetone_gain;
  /* configuration of the acoustic echo cancellation */
  T_AUDIO_AEC_CFG               aec;
   /* ES configuration */
   T_AUDIO_ES_CFG               es;
}
T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING;
```

- For other configurations :

```
typedef struct
{
  /* gain of the sidetone */
  INT8                          sidetone_gain;
  /* configuration of the acoustic echo cancellation */
  T_AUDIO_AEC_CFG               aec;
}
T_AUDIO_MICROPHONE_SPEAKER_LOOP_SETTING;
```

Where the AEC parameters areas follows::

9.2.1.6.1 T_AUDIO_AEC_CFG

The primary goal of the AEC is to remove the echo stemming from the loudspeaker reverberation to the microphone.

AEC:

```
typedef struct
{
  /* Enable the AEC */
  UINT16    aec_enable;
  /* Mode of the AEC */
  UINT16    aec_mode;
  /* level of the echo cancellation */
  UINT16    echo_suppression_level;
  /* enable the noise suppression */
  UINT16    noise_suppression_enable;
  /* level of the noise suppression */
  UINT16    noise_suppression_level;
}
T_AUDIO_AEC_CFG;
```

NEW AEC :

```
typedef struct
{
  UINT16    aec_enable;
  BOOL      continuous_filtering;
  UINT16    granularity_attenuation;
  UINT16    smoothing_coefficient;
  UINT16    max_echo_suppression_level;
  UINT16    vad_factor;
  UINT16    absolute_threshold;
  UINT16    factor_asd_filtering;
  UINT16    factor_asd_muting;
  UINT16    aec_visibility ;
  UINT16    noise_suppression_enable;
  UINT16    noise_suppression_level;
}
T_AUDIO_AEC_CFG;
```

IMPORTANT NOTE:

The ES features is only present in TCS 3.x software except TCS 3.0. Other software versions do not include T_AUDIO_ES_CFG structure.

For T_AUDIO_ES_CFG details: see 9.2.1.3.4

NOTE:

The noise suppressor is replaced by a new ANR module in TCS 3.x software except TCS 3.0. For these software versions, "noise_suppression_enable" and "noise_suppression_level" parameters don't exist. See 9.2.1.3 (microphone settings) for information on ANR.

Below the detail of each parameters:

Sidetone ***(For P2 samples using a non-TI audio ABB)***

Specifies the gain to add to the loop between the microphone and the speaker. The range is from 0x0001 (-90 dB) to 0xFFFF (6 dB) in unsigned Q15 format, ex:
0dB → (2^15)*10^(0/20) = 0x8000
-6 dB → (2^15)*10^(-6/20) = 0x4026
0x08F5 → 20*log(2293/(2^15)) = -23 dB

This mode is only available in GSM and bluetooth cordless voice path mode.

Sidetone ***(For other configurations)***

Specifies the gain in 3 dB unit to add to the loop between the microphone and the speaker. The range is from –23 dB to 1 dB (3 dB by 3 dB). Note if the variable is equal to AUDIO_SIDETONE_OPEN, there's no loop between the microphone and the speaker.

This mode is only available in GSM and bluetooth cordless voice path mode.

WARNING: IF THE SPEAKER FILTER IS BYPASSED, THE SIDETONE IS OPEN.

es

See 9.2.1.3.4 for details

AEC

aec_enable

Specifies if the AEC module must be enabled (AUDIO_AEC_ENABLE) or disabled (AUDIO_AEC_DISABLE).

Note: AEC is only available in GSM and all DAI mode.

In case of Read Access, the following parameters are valid only if anr_enable = 1.

aec_mode

Specifies the mode of the cancellation: AUDIO_SHORT_ECHO: short echo cancellation, AUDIO_LONG_ECHO: long echo cancellation.

Note: AEC is only available in GSM all DAI mode.

echo_suppression_level

Specifies the additional echo suppression level.

Note: noise suppression is only available in GSM and all mode.

Level name	level (dB)
AUDIO_ECHO_0dB	0
AUDIO_ECHO_6dB	6
AUDIO_ECHO_12dB	12
AUDIO_ECHO_18dB	18

noise_enable

Specifies if the noise suppression module must be enable (AUDIO_NOISE_SUPPRESSION_ENABLE) or disable (AUDIO_NOISE_SUPPRESSION_DISABLE).

Note: noise suppression is only available in GSM and all DAI mode.

Note: The noise suppressor is replaced by ANR in TCS3.x except 3.0.

noise_suppression_level

Specifies the noise suppression limitation level.

Note: AEC is only available in GSM and all DAI mode.

Level name	level (dB)

AUDIO_NOISE_NO_LIMIT	no limitation
AUDIO_NOISE_6dB	-6
AUDIO_NOISE_12dB	-12
AUDIO_NOISE_18dB	-18

Note: The noise suppressor is replaced by ANR in TCS3.x except 3.0.

NEW AEC (detailed format of the parameters can be found in [2])

aec_enable
Specifies if the AEC module must be enabled (AUDIO_AEC_ENABLE) or disabled (AUDIO_AEC_DISABLE).
Note: NEW AEC is only available in GSM and all DAI mode.

In case of Read Access, the following parameters are valid only if aec_enable = 1.

continuous_filtering
Enable (TRUE) or disable (FALSE) continuous mode filtering.

granularity_attenuation
granularity of the smoothed attenuation.

smoothing_coefficient
smoothing coefficient.

max_echo_suppression_level
maximum attenuation level. Some values are defined as constants:
AUDIO_MAX_ECHO_xdB with x being 0, 2, 3, 6, 12, 18, 24.

vad_factor
VAD factor relative to the current estimated energy.

absolute_threshold
VAD absolute offset relative to the current estimated energy.

factor_asd_filtering
modifying factor of d_far_end_noise for filtering decision.

factor_asd_muting
modifying factor of d_far_end_noise for muting decision.

aec_visibility
Enable (AUDIO_AEC_VISIBILITY_ ENABLE) or disable (AUDIO_AEC_VISIBILITY_ DISABLE) AEC visibility. A copy of far_end_pow and far_end_noise is traced in Layer1 every SC_AEC_VISIBILITY_INTERVAL frames. It is intended for debug purposes and can only be disabled by a new AEC request (i.e. going back to idle mode won't disable visibility for next call).

noise_enable
Specifies if the noise suppression module must be enabled (AUDIO_NOISE_SUPPRESSION_ENABLE) or disabled (AUDIO_NOISE_SUPPRESSION_DISABLE).
Note: The noise suppressor is replaced by ANR in TCS3.x except 3.0.

noise_suppression_level
Specifies the noise suppression limitation level.
Note: AEC is only available in GSM and all DAI mode.

Level name	**level (dB)**
AUDIO_NOISE_NO_LIMIT	no limitation
AUDIO_NOISE_6dB	-6
AUDIO_NOISE_12dB	-12
AUDIO_NOISE_18dB	-18

Note: The noise suppressor is replaced by ANR in TCS3.x except 3.0.

AEC 2.x

In TCS3.2 acoustic chain the version used is AEC 2.x.
AEC module only works in DAI acoustic and GSM and Bluetooth headset path modes.
The following AEC settings are to be used for AEC 2.x used in the TCS 3.2 acoustic chain.

AEC settings are inside following structure of the microphone speaker loop settings:

```
typedef struct
{
  T_AEC_CONTROL          aec_control;
   T_AUDIO_AEC_PARAMS    parameters ;
}
T_AUDIO_AEC_CFG;

typedef enum
{
  AEC_STOP  = 0,
```

```
  AEC_START = 1,
  AEC_UPDATE = 2
}
T_AEC_CONTROL;

typedef struct
{
   INT16     mode;
   INT16     mu;
   INT16     cont_filter;
   INT16     scale_input_ul;
   INT16     scale_input_dl;
   INT16     div_dmax;
   UINT16    div_swap_good;
   UINT16    div_swap_bad;
   INT16     block_init;
   INT16     fact_vad;
   UINT16    fact_asd_fil;
   UINT16    fact_asd_mut;
   UINT16     thr_abs;
   INT16     es_level_max;
   UINT16     granularity_att;
   INT16     coef_smooth;
   UINT16     block_size;
}
T_AUDIO_AEC_PARAMS;
```

Name	Type	Format	Range	Comments
mode	INT16	16b/Q0	0x0008	AEC 1.8 with internal VAD and ES (previous solution)
			0x0000	AEC 1.8 with internal VAD and without internal ES
			0x0007	AEC 2.0 with divergence control enabled
			0x0003	AEC 2.0 with divergence control disabled
mu	INT16	16b/Q15	0x5000	Ref. 0.625
cont_filter	INT16	16b/Q0	0x0000	continuous filtering disabled
			0x0001	continuous filtering enabled
scale_input_ul	INT16	16b/Q0	0x0000	Ref. 1, no scaling uplink
			0x0003	Ref. 8, scaling uplink
scale_input_dl	INT16	16b/Q15	0x0000	Ref. 1, no scaling downlink
			0x0003	Ref. 8, no scaling downlink
div_dmax	INT16	16b/Q12	0x537D	Ref. 21373
div_swap_good	UINT16	16b/Q0	0x7FB2	Ref. 32690
div_swap_bad	UINT16	16b/Q0	0x65AD	Ref. 26029
block_init	INT16	16b/Q0	0x07D0	Ref. 2000 samples
fact_vad	INT16	16b/Q15	0x3FFF	Ref. 0.5
fact_asd_fil	UINT16	16b/Q12	0x1000	Ref. 4096
fact_asd_mut	UINT16	16b/Q12	0x1000	Ref. 4096
thr_abs	UINT16	16b/Q0	0x0032	Ref. 50
es_level_max	INT16	16b/Q15	[0x0813,	Ref. -24dB
			…,	
			0x7FFF]	Ref. 0dB
granularity_att	UINT16	16b/Q0	0x00A0	Ref. 160 samples
coef_smooth	INT16	16b/Q15	[0x0CCB,	Ref. 3275
			…,	
			0x7FFF]	Ref. 32767
block_size	UINT16	16b/Q0	0x0001	Ref. 1 sample

9.2.1.7 T_AUDIO_MICROPHONE_SPEAKER_SETTING

Specifies the characteristic of the features that are common to the speaker and the microphone.

```
typedef struct
 {
   /* volume speed control */
   INT16 volume_speed;
   /* audio on/off */
   INT8  audio_onoff;
 }
 T_AUDIO_MICROPHONE_SPEAKER_SETTING;
```

Below the detail of each parameters:

volume_speed ***(only available in case of non-TI audio ABB used with P2 samples)***
speed to change the volume in downlink and in uplink
values are from 0x0001 (low speed) to 0x7FFF (high speed) in signed Q15 format, ex:
0x1 → (2^15)/1 = 32768 samples to reach the volume level
0x2 → (2^15)/2 = 16384 samples to reach the volume level
0x7FFF → (2^15)/32767 = 1 sample to reach the volume level

Audio_onoff ***(only available on Calypso+, Perseus 2 and Locosto samples)***
If set to 1, it starts ABB audio and disable the automatic stop when no DSP audio activity is running.
If set to 0, it will stop ABB audio when there is no DSP (or L1 MCU in case of Locosto) audio activity running.

9.2.2 API functions

9.2.2.1 audio_mode_load

```
T_AUDIO_RET audio_mode_load ( T_AUDIO_MODE_LOAD *p_parameter,
                              T_RV_RETURN       return_path)
```

Description

This function is called to set an audio mode saved in a flash file.

Parameters

- **T_AUDIO_MODE_LOAD**

This parameter specifies the path name of the audio mode Flash file.

```
typedef struct
{
  char  audio_mode_filename[AUDIO_MODE_FILENAME_MAX_SIZE];
}
T_AUDIO_MODE_LOAD;
```

Below the detail of each parameters:

audio_mode_filename
Specifies the file name of the audio mode file. Note that this file name is used by the audio entity to request the data to the File Flash System. Due to the fact that each audio mode flash files are in the same folder (*/aud/*) and have the same extension (*.cfg*). The file name to specified contains only the name of the file without the folder name and the extension. For example: for the file */aud/headset.cfg* the file name is *headset*. Note the maximum size of the path plus the name is 10 characters.

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	An error is occurred during the execution of this function

Event Return

- **AUDIO_MODE_LOAD_DONE**

This event informs that the mobile is configured with the new audio mode.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
}T_AUDIO_MODE_LOAD_DONE;
```

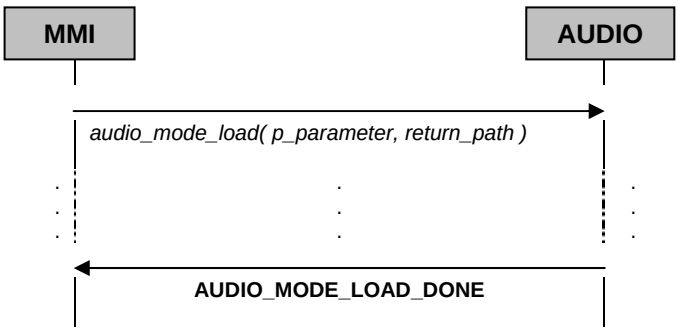
The possible values of *status* are:

value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	A problem occurs during the audio mode configuration

Current restriction of use

None.

Process flow



9.2.2.2 audio_mode_save

```
T_AUDIO_RET audio_mode_save ( T_AUDIO_MODE_SAVE *p_parameter,
                              T_RV_RETURN       return_path)
```

Description

This function is called to save the current audio mode in a flash file.

Note: Only a

Parameters

- **T_AUDIO_MODE_SAVE**

This parameter describes where the current audio mode must be saved.

```
typedef struct
{
  char  audio_mode_filename[AUDIO_MODE_FILENAME_MAX_SIZE];
}
T_AUDIO_MODE_SAVE;
```

Below the detail of each parameters:

audio_mode_filename
Specifies the file name of the audio mode file. Note that this file name is used by the audio entity to save the audio mdoe to the File Flash System. Due to the fact that each audio mode flash files are in the same folder (*/aud/*) and have the same extension (*.cfg*). The file name to specified contains only the name of the file without the folder name and the extension. For example: for the file */aud/headset.cfg* the file name is *headset*. Note the maximum size of the path plus the name is 10 characters.

- **T_RV_RETURN**

C.f. API function *audio_mode_load*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_mode_load*.

Event Return

- **AUDIO_MODE_SAVE_DONE**

This event informs that the audio mode is saved in flash.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
}T_AUDIO_MODE_SAVE_DONE;
```

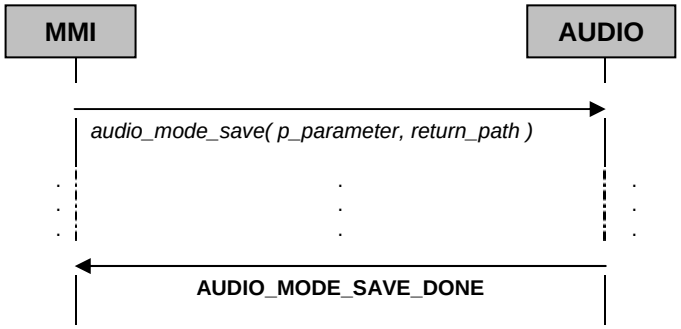
The possible values of *status* are:

Value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	A problem occurs during the audio mode configuration

Current restriction of use

None.

Process flow



9.2.2.3 audio_speaker_volume

This chapter describes how to tune the volume of the speaker.
It is only available with the analog base bands SYREN and TRITON.

```
T_AUDIO_RET audio_speaker_volume ( T_AUDIO_SPEAKER_VOLUME volume,
                                   T_RV_RETURN       return_path)
```

Description

This function is called to change the speaker volume. The volume is save in flash in a file with the same filename of the current audio mode. So each mode can have its own volume.

Moreover this new file flash has the extension "*.vol*" and it's saved in the folder "*aud*\".

Each speaker volume flash file contains a variable structured like below:

```
{
  /* volume of the audio speaker */
  UINT8      audio_speaker_level;
}
T_AUDIO_SPEAKER_LEVEL;
```

NOTE: THIS FEATURES IS ONLY AVAILABE IN GSM, BLUETOOTH CORDLESS VOICE AND ALL DAI PATH MODE (c.f. audio mode configuration section).

Parameters

- **T_AUDIO_SPEAKER_VOLUME**

```
typedef struct {
  UINT8   volume_action;
  UINT8   value;
} T_AUDIO_SPEAKER_VOLUME;
```

Below the detail of each parameters:

volume_action
Specify the action of the audio speaker volume function. The action can be to set directly the volume (AUDIO_SPEAKER_VOLUME_SET) or to increase (AUDIO_SPEAKER_VOLUME_INCREASE) or to decrease (AUDIO_SPEAKER_VOLUME_INCREASE) the current volume corresponding to the current mode.

Value
Specify the value when the action is to set directly the volume (AUDIO_SPEAKER_VOLUME_SET).
The Volume value can be:

Level name	level (dB)
AUDIO_SPEAKER_MUTE	mute the speaker
AUDIO_SPEAKER_VOLUME_0dB	0
AUDIO_SPEAKER_VOLUME_6dB	-6
AUDIO_SPEAKER_VOLUME_12dB	-12
AUDIO_SPEAKER_VOLUME_18dB	-18
AUDIO_SPEAKER_VOLUME_24dB	-24

WARNING: IF THE AUDIO FILTER (c.f. audio mode structure) IS BYPASSED, THE VOLUME TOO.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_mode_load*.

Event Return

- **AUDIO_SPEAKER_VOLUME_DONE**

This event informs that the audio volume was changed.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
}T_AUDIO_SPEAKER_VOLUME_DONE;
```

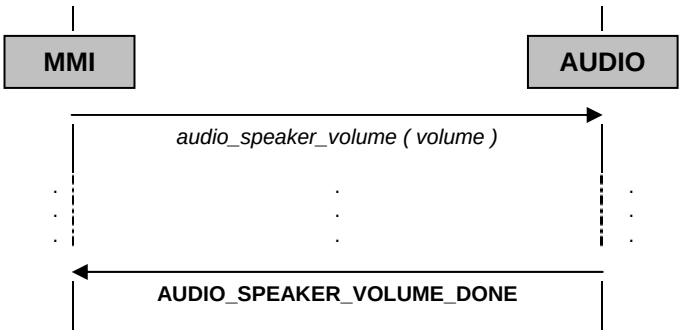
The possible values of *status* are:

Value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	A problem occurs during the audio volume configuration

Current restriction of use

- An audio mode needs to be load before to use this API function.

Process flow



9.2.2.4 audio_stereo_speaker_volume

This chapter describes how to tune the volume of the audio stereo speaker. It is only available with the analog base band SYREN and TRITON.

```
T_AUDIO_RET audio_stereo_speaker_volume ( T_AUDIO_SPEAKER_VOLUME volume,
                                          T_RV_RETURN       return_path)
```

Description

This function is called to change the audio stereo speaker volume. The volume is saved in flash in a file with the same filename of the current audio mode. So each mode can have its own volume.

Moreover this new file flash has the extension "*.volst*" and it is saved in the folder "*aud*\".

Each speaker volume flash file contains a variable structured like below:

```
{
  /* volume of the audio speaker */
  UINT8      audio_stereo_speaker_level_left;
  UINT8      audio_stereo_speaker_level_right;
}
T_AUDIO_STEREO_SPEAKER_LEVEL;
```

Parameters

- **T_AUDIO_STEREO_SPEAKER_VOLUME**

```
typedef struct {
  UINT8   volume_action_left;
  UINT8   value_left;
  UINT8   volume_action_right;
  UINT8   value_right;
} T_AUDIO_STEREO_SPEAKER_VOLUME;
```

Below the detail of each parameters:

volume_action_left Specify the action of the audio speaker volume function for left channel. The action can be to set directly the volume (AUDIO_STEREO_SPEAKER_VOLUME_SET) or to increase (AUDIO_STEREO_SPEAKER_VOLUME_INCREASE) or to decrease (AUDIO_STEREO_SPEAKER_VOLUME_DECREASE) the current volume corresponding to the current mode.
value_left Specify the value when the action is to set directly the volume on left channel. The Volume value can be: • AUDIO_STEREO_SPEAKER_MUTE: mute the speaker • Any value V between 0 and 30: level will be –V dB (i.e. between 0 and -30)
volume_action_right as above for right channel
value_right as above for right channel

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_mode_load*.

Event Return

- **AUDIO_STEREO_SPEAKER_VOLUME_DONE**

This event informs that the audio volume was changed.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_STEREO_SPEAKER_VOLUME_DONE;
```

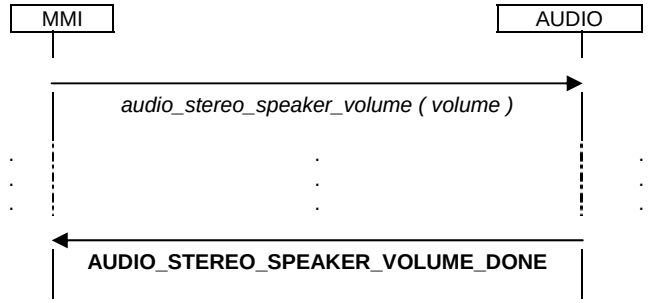
The possible values of *status* are:

Value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	A problem occurs during the audio volume configuration

Current restriction of use

- An audio mode needs to be loaded before to use this API function.

Process flow



9.3 Full access family

This chapter describes all the API functions belong to the full access family.

9.3.1 API functions

9.3.1.1 audio_full_access_write

```
T_AUDIO_RET audio_full_access_write (
                        T_AUDIO_FULL_ACCESS_WRITE     *p_parameter,
                        T_RV_RETURN       return_path)
```

Description

This function is called to configure any value belonging to the audio mode structure.

Parameters

- **T_AUDIO_FULL_ACCESS_WRITE**

```
typedef struct {
  UINT8   variable_indentifier;
  // identifier of the variable to configure
  void    *data;
  // data corresponding to the variable to set
}T_AUDIO_FULL_ACCESS_WRITE;
```

Below the detail of each parameters (note for the description of the data please see the MMI family chapter):

Identifier	Associated data format
AUDIO_PATH_USED	typedef UINT8 T_AUDIO_VOICE_PATH_SETTING;
AUDIO_MICROPHONE_MODE *(1)*	INT8
AUDIO_MICROPHONE_GAIN *(2)*	INT8
AUDIO_MICROPHONE_EXTRA_GAIN *(1)*	INT8
AUDIO_MICROPHONE_OUTPUT_BIAS *(1)*	INT8

AUDIO_MICROPHONE_FIR	typedef struct
	{
	UINT16 coefficient[31];
	}
	T_AUDIO_FIR_COEF;

AUDIO_MICROPHONE_ANR *(6)*	T_AUDIO_ANR_CFG (see 9.2.1.3.3). In case of TCS 3.2, see 9.2.1.3.1
AUDIO_MICROPHONE_ES *(6)*	T_AUDIO_ES_CFG (see 9.2.1.3.4)
AUDIO_MICROPHONE_AGC (7)	T_AUDIO_AGC_CFG (see 9.2.1.3.2)
AUDIO_SPEAKER_MODE *(1)*	INT8
AUDIO_SPEAKER_GAIN *(2)*	INT8
AUDIO_SPEAKER_EXTRA_GAIN *(1)*	INT8
AUDIO_SPEAKER_FILTER *(1)*	INT8
AUDIO_SPEAKER_HIGHPASS_FILTER *(1)*	INT8

AUDIO_SPEAKER_FIR *(5)*	typedef struct
	{
	UINT16 coefficient[31];
	}
	T_AUDIO_FIR_COEF;

AUDIO_SPEAKER_IIR *(6)*	T_AUDIO_IIR_CFG (see 9.2.1.4.5). In case of TCS 3.2, see 9.2.1.4.3
AUDIO_SPEAKER_LIMITER *(6)*	T_AUDIO_LIMITER_CFG (see 9.2.1.4.4)

AUDIO_SPEAKER_AGC (7)	T_AUDIO_AGC_CFG (see 9.2.1.3.2)
AUDIO_MICROPHONE_AGC (7)	T_AUDIO_AGC_CFG (see 9.2.1.3.2)
AUDIO_SPEAKER_DRC (7)	T_AUDIO_DRC_CFG (see 9.2.1.4.2)
AUDIO_SPEAKER_BUZZER *(1)*	INT8
AUDIO_MICROPHONE_SPEAKER_LOOP_SIDETONE *(2)*	INT8

AUDIO_MICROPHONE_SPEAKER_LOOP_AEC	typedef struct
	{
	/* Enable the AEC */
	UINT16 aec_enable;
	/* Mode of the AEC */
	UINT16 aec_mode;
	/* level of the echo cancellation */
	UINT16 echo_suppression_level;
	/* enable the noise suppression */
	UINT16 noise_suppression_enable;
	/* level of the noise suppression */
	UINT16 noise_suppression_level;
	}
	T_AUDIO_AEC_CFG;
	
	noise_suppression_level and noise_suppression_enable not available in TCS3.0 software except TCS3.0 (replaced by ANR)
	
	In case of TCS 3.2, see 9.2.1.6.1

AUDIO_STEREO_SPEAKER_MODE *(1)*	INT8
AUDIO_STEREO_SPEAKER_STEREO_MONO *(1)*	INT8
AUDIO_STEREO_SPEAKER_SAMPLING_FREQUENCY *(1)*	INT8

AUDIO_SPEAKER_VOLUME_LEVEL *(1)*	typedef struct
	{
	/* volume of the audio speaker */
	UINT8 audio_speaker_level;
	}
	T_AUDIO_SPEAKER_LEVEL;

AUDIO_STEREO_SPEAKER_VOLUME_LEVEL *(1)*	typedef struct
	{
	/* volume of the audio speaker */
	UINT8 audio_stereo_speaker_level_left;
	UINT8 audio_stereo_speaker_level_right;
	}
	T_AUDIO_STEREO_SPEAKER_LEVEL;

AUDIO_ONOFF *(3)*	INT8
AUDIO_VOLUME_SPEED *(4)*	INT16

(1) Not available when using a non-TI ABB for audio tasks
(2) When using a non-TI ABB for audio tasks, type of parameter is Int16 instead of Int8
(3) Only available on Calypso+ and Perseus2 samples
(4) Only available when using a non-TI ABB for audio tasks
(5) Not available in TCS3.x software except TCS3.0
(6) Only available in TCS3.x software except TCS3.0
(7) Only available in TCS 3.2

- **T_RV_RETURN**

C.f. API function *audio_mode_load*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_mode_load*.

Event Return

- **AUDIO_FULL_ACCESS_WRITE_DONE**

This event informs that the value was written.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
}T_AUDIO_FULL_ACCESS_WRITE_DONE;
```

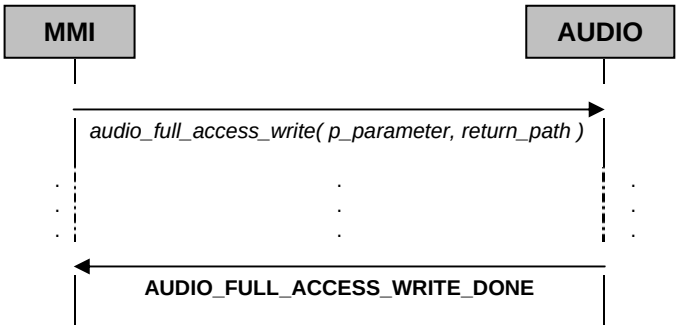
The possible values of *status* are:

value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped
-1	AUDIO_ERROR	A problem occurs during the writing process..

Current restriction of use

None.

Process flow



9.3.1.2 audio_full_access_read

```
T_AUDIO_RET audio_full_access_read (
                        T_AUDIO_FULL_ACCESS_READ      *p_parameter)
```

Description

This function is called to read any value belonging to the audio mode structure.

Parameters

- **T_AUDIO_FULL_ACCESS_READ**

```
typedef struct {
  UINT8  variable_indentifier; // identifier of the variable to read
  VOID   *data                 // data to return
}T_AUDIO_FULL_ACCESS_READ;
```

Below the detail of each parameters (note for the description of the data please see the MMI family chapter):

Identifier	Associated data format
AUDIO_PATH_USED	typedef UINT8 T_AUDIO_VOICE_PATH_SETTING;
AUDIO_MICROPHONE_MODE *(1)*	INT8
AUDIO_MICROPHONE_GAIN *(2)*	INT8
AUDIO_MICROPHONE_EXTRA_GAIN *(1)*	INT8
AUDIO_MICROPHONE_OUTPUT_BIAS *(1)*	INT8

AUDIO_MICROPHONE_FIR	typedef struct
	{
	UINT16 coefficient[31];
	}
	T_AUDIO_FIR_COEF;

AUDIO_MICROPHONE_ANR *(6)*	T_AUDIO_ANR_CFG (see 9.2.1.3.3). In case of TCS 3.2, see 9.2.1.3.1
AUDIO_MICROPHONE_ES *(6)*	T_AUDIO_ES_CFG (see 9.2.1.3.4)
AUDIO_MICROPHONE_AGC(7)	T_AUDIO_AGC_CFG (see 9.2.1.3.2)
AUDIO_SPEAKER_MODE *(1)*	INT8
AUDIO_SPEAKER_GAIN *(2)*	INT8
AUDIO_SPEAKER_EXTRA_GAIN *(1)*	INT8
AUDIO_SPEAKER_FILTER *(1)*	INT8
AUDIO_SPEAKER_HIGHPASS_FILTER *(1)*	INT8

AUDIO_SPEAKER_FIR *(5)*	typedef struct
	{
	UINT16 coefficient[31];
	}
	T_AUDIO_FIR_COEF;

AUDIO_SPEAKER_IIR *(6)*	T_AUDIO_IIR_CFG (see 9.2.1.4.5) In case of TCS 3.2, (see 9.2.1.4.3)
AUDIO_SPEAKER_LIMITER *(6)*	T_AUDIO_LIMITER_CFG (see 9.2.1.4.4)
AUDIO_SPEAKER_AGC(7)	T_AUDIO_AGC_CFG (see 9.2.1.3.2)
AUDIO_MICROPHONE_AGC (7)	T_AUDIO_AGC_CFG (see 9.2.1.3.2)
AUDIO_SPEAKER_DRC(7)	T_AUDIO_DRC_CFG (see 9.2.1.4.2)
AUDIO_SPEAKER_BUZZER *(1)*	INT8
AUDIO_MICROPHONE_SPEAKER_LOOP_SIDETONE *(2)*	INT8
AUDIO_MICROPHONE_SPEAKER_LOOP_AEC	typedef struct { /* Enable the AEC */ UINT16 aec_enable; /* Mode of the AEC */

	UINT16 aec_mode; /* level of the echo cancellation */ UINT16 echo_suppression_level; /* enable the noise suppression */ UINT16 noise_suppression_enable; /* level of the noise suppression */ UINT16 noise_suppression_level; } T_AUDIO_AEC_CFG; noise_suppression_level and noise_suppression_enable not available in TCS3.0 software except TCS3.0 (replaced by ANR) In case of TCS 3.2, see 9.2.1.6.1
AUDIO_STEREO_SPEAKER_MODE *(1)*	UINT8
AUDIO_STEREO_SPEAKER_STEREO_MONO *(1)*	UINT8
AUDIO_STEREO_SPEAKER_SAMPLING_FREQUENCY *(1)*	INT8

AUDIO_SPEAKER_VOLUME_LEVEL *(1)*	typedef struct
	{
	/* volume of the audio speaker */
	UINT8 audio_speaker_level;
	}
	T_AUDIO_SPEAKER_LEVEL;

AUDIO_STEREO_SPEAKER_VOLUME_LEVEL *(1)*	typedef struct
	{
	/* volume of the audio speaker */
	UINT8 audio_stereo_speaker_level;
	}
	T_AUDIO_STEREO_SPEAKER_LEVEL;

AUDIO_ONOFF *(3)*	INT8
AUDIO_VOLUME_SPEED *(4)*	INT16

(1) Not available when using a non-TI ABB for audio tasks
(2) When using a non-TI ABB for audio tasks, type of parameter is Int16 instead of Int8
(3) Only available on Calypso+ and Perseus2 samples
(4) Only available when using a non-TI ABB for audio tasks
(5) Not available in TCS3.x software except TCS3.0
(6) Only available in TCS3.x software except TCS3.0
(7) Only available in TCS 3.2.

- **T_RV_RETURN**

C.f. API function *audio_mode_load*.

Immediate Return

- **T_AUDIO_RET**

C.f. API function *audio_mode_load*.

- Moreover the data pointer of the T_AUDIO_FULL_ACCESS_READ structure points to the data to returned data if the *T_AUDIO_RET* value is equal to *AUDIO_OK*.

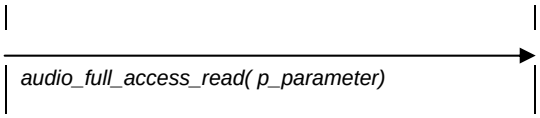
Event Return

None.

Current restriction of use

None.

Process flow



10 AMR play/record

10.1 Overview

This API allows playing an AMR file from FFS or RAM and recording to FFS or RAM. This can be used to create Voice Memos or AMR MMS files.
Played/recorded AMR format is **fully compliant with a mono-channel AMR MMS in storage mode** as defined in RFC 3267. However, the 6-character "**#!AMR\n" MMS header is not always handled:**

- For audio_amr_... APIs, upper layers must insert it at the beginning of the recorded samples or remove it before play (not needed for a simple AMR Voice Memo).
- For audio_mms_... APIs, header is removed before play and inserted before record.

If FFS is used, more processing is required but it is more flexible in terms of size. The user is only limited by the total size of the flash. On the contrary, there is no API to handle RAM like a file-system so AMR feature works with a single contiguous buffer. Therefore, the user must not try to record more samples than can be stored in the RAM buffer. It is recommended to use only Flash for voice memos.

Note: when audio_mms_... APIs have same immediate return, event return, comments, parameters description… than their audio_amr_... counterparts, they are not repeated. For example, to get a full description of audio_mms_record_to_ffs_start(…), a customer must also read audio_amr_record_to_ffs_start(…).

10.2 Immediate return and event return

The API described below allows starting or stopping AMR features. To test the success of the full process, an API caller must check 2 types of information:

- The **immediate return** is simply the value returned by the API (the function called). It indicates "failure" if parameters were wrong or there was not enough memory to send the start message to the audio entity.
- The **event return** is the STATUS event/message sent back to the API caller at the end of the execution. It indicates success or failure of audio entity processing.

A complete execution includes these steps (See Figure 2):

- An entity calls an API function and immediate return is OK. A message is sent to the audio entity.
- The audio entity tries to start the feature. Start fails (for example, not enough memory) or feature is successfully executed and stopped (automatically or through a stop API call).
- STATUS message is sent back to the entity, which called the start API. It indicates success of the execution (feature started and stopped) or failure to start the feature. The optional generated file is only valid if STATUS is OK.

All possible process flows are described in figures below:

- Successful start API call but feature can't start.

- Successful start API call and automatic or requested stop.
- Successful stop API call without any related start API call. No STATUS is sent as no processing was performed in the audio entity.

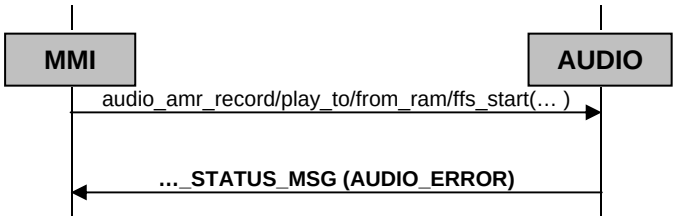


Figure 1: Start fails

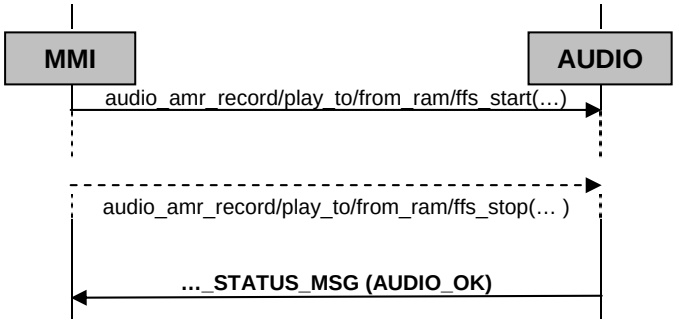


Figure 2: Start feature + automatic or requested stop concluded by STATUS message

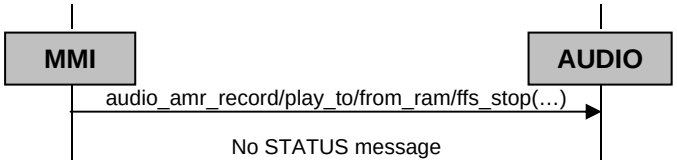


Figure 3: Stop feature with no start

10.3 audio_amr_record_to_ram/ffs_start

```
T_AUDIO_RET audio_amr_record_to_ram_start (
            T_AUDIO_AMR_RECORD_TO_RAM_PARAMETER *p_record_parameter,
            T_RV_RETURN return_path)
```

```
T_AUDIO_RET audio_amr_record_to_ffs_start (
            T_AUDIO_AMR_RECORD_TO_FFS_PARAMETER *p_record_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to start recording an AMR file to RAM/FFS.

Parameters

- **T_AUDIO_AMR_RECORD_TO_RAM_PARAMETER**

Specifies the characteristics for record phase.

```
typedef struct
{
  UINT8   *p_buffer;
  UINT32  memo_duration;
  BOOL    compression_mode;
  UINT16  microphone_gain;
  UINT8   amr_vocoder;
}T_AUDIO_AMR_RECORD_TO_RAM_PARAMETER;
```

- **T_AUDIO_AMR_RECORD_TO_FFS_PARAMETER**

Specifies the characteristics for record phase.

```
typedef struct
{
  char    memo_name[AUDIO_PATH_NAME_MAX_SIZE];
  UINT32  memo_duration;
  BOOL    compression_mode;
  UINT16  microphone_gain;
  UINT8   amr_vocoder;
}T_AUDIO_AMR_RECORD_TO_RAM_PARAMETER;
```

Below the detail of each parameters:

p_buffer Pointer on the buffer where AMR file will be stored. **memo_name** Specifies the file name of the AMR file. The file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 80 characters.
memo_duration Specifies the maximum duration of the AMR file in **byte** unit.
microphone_gain Specifies the gain multiplied to the voice sample from the microphone. The format is Q8.8, for example: if microphone_gain = 0x0100, the gain is 1 and if microphone_gain = 0x0080, the gain is 0,5.
compression_mode Activate or deactivate the compression of the voice recorded. It means that the silence between two voice activity is detected and compressed. Compression can have 2 values: **AUDIO_AMR_COMPRESSION_MODE** or **AUDIO_AMR_NO_COMPRESSION_MODE**.
amr_vocoder Rate of vocoder used. Possible values are: **AUDIO_AMR_VOCODER_4_75 -> 4.75kbps** **AUDIO_AMR_VOCODER_5_15 -> 5.15kbps** **AUDIO_AMR_VOCODER_5_90 -> 5.90kbps** **AUDIO_AMR_VOCODER_6_70 -> 6.70kbps** **AUDIO_AMR_VOCODER_7_40 -> 7.40kbps** **AUDIO_AMR_VOCODER_7_95 -> 7.95kbps** **AUDIO_AMR_VOCODER_10_2 -> 10.2kbps** **AUDIO_AMR_VOCODER_12_2 -> 12.2kbps**

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

- **AUDIO_AMR_RECORD_TO_MEM_STATUS_MSG**

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
  UINT32    record_duration;
}T_AUDIO_AMR_RECORD_STATUS;
```

Below the detail of each parameters:

Status Status of the record. If it is AUDIO_OK, the file is valid, otherwise it is AUDIO_ERROR and file must not be used.
recorded_duration Specifies the size of the file in byte unit.

10.4 audio_amr_record_to_ram/ffs_stop

```
T_AUDIO_RET audio_amr_record_to_ram/ffs_stop (void)
```

Description

This function is called to stop recording an AMR file to RAM/FFS.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return (if start API was called)

- **AUDIO_AMR_RECORD_TO_MEM_STATUS_MSG**

```
typedef struct {
```

```
  T_RV_HDR  os_hdr;
  INT8      status;
  UINT32    record_duration;
}T_AUDIO_AMR_RECORD_STATUS;
```

Below the detail of each parameters:

Status Status of the record. If it is AUDIO_OK, the file is valid, otherwise it is AUDIO_ERROR and file must not be used.
recorded_duration Specifies the size of the file in byte unit.

10.5 audio_amr_play_from_ram/ffs_start

```
T_AUDIO_RET audio_amr_play_from_ram_start (
            T_AUDIO_AMR_PLAY_FROM_RAM_PARAMETER *p_parameter,
            T_RV_RETURN return_path)
```

```
T_AUDIO_RET audio_amr_play_from_ffs_start (
            T_AUDIO_AMR_PLAY_FROM_FFS_PARAMETER *p_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to start playing an AMR file from FFS/RAM..

Parameters

- **T_AUDIO_AMR_PLAY_FROM_RAM_PARAMETER**

```
typedef struct
{
  UINT8   *p_buffer;
  UINT16  buffer_size;
}T_AUDIO_AMR_PLAY_FROM_RAM_PARAMETER;
```

- **T_AUDIO_AMR_PLAY_FROM_FFS_PARAMETER**

```
typedef struct
{
  char memo_name[AUDIO_PATH_NAME_MAX_SIZE];
}T_AUDIO_AMR_PLAY_FROM_FFS_PARAMETER;
```

Below the detail of each parameters:

p_buffer Pointer on the buffer where AMR file will be stored. **memo_name** Specifies the file name of the AMR file. The file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 80 characters.
buffer_size Specifies the duration of the AMR file in **byte** unit.

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

- **AUDIO_AMR_PLAY_FROM_MEM_STATUS_MSG**

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_AMR_PLAY_STATUS;
```

10.6 audio_amr_play_from_ram/ffs_stop

```
T_AUDIO_RET audio_amr_play_from_ram/ffs_stop (void)
```

Description

This function is called to stop playing an AMR file to RAM/FFS.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return (if start API was called)

- **AUDIO_AMR_PLAY_FROM_MEM_STATUS_MSG**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
}T_AUDIO_AMR_PLAY_STATUS;
```

10.7 audio_mms_record_to_ffs_start

```
T_AUDIO_RET audio_mms_record_to_ffs_start (
            T_AUDIO_MMS_RECORD_TO_FFS_PARAMETER *p_record_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to start recording an AMR-MMS file to FFS.

Parameters

- **T_AUDIO_MMS_RECORD_TO_FFS_PARAMETER (identical to T_AUDIO_AMR_RECORD_TO_FFS_PARAMETER)**

Specifies the characteristics for record phase. memo_duration specifies the maximum duration of the file in **byte** unit but does not include the size of the header.

10.8 audio_mms_record_to_ffs_stop

```
T_AUDIO_RET audio_mms_record_to_ffs_stop (void)
```

Description

This function is called to stop recording an AMR-MMS file to FFS.

Event Return (if start API was called)

- **AUDIO_AMR_RECORD_TO_MEM_STATUS_MSG**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
  UINT32     record_duration;
}T_AUDIO_AMR_RECORD_STATUS;
```

Below the detail of each parameters:

Status Status of the record. If it is AUDIO_OK, the file is valid, otherwise it is AUDIO_ERROR and file must not be used.
recorded_duration Specifies the size of the file in byte unit. It does not include the size of the header.

10.9 audio_mms_play_from_ffs_start

```
T_AUDIO_RET audio_mms_play_from_ffs_start (
            T_AUDIO_MMS_PLAY_FROM_FFS_PARAMETER *p_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to start playing an AMR-MMS file from FFS.

Parameters

- **T_AUDIO_MMS_PLAY_FROM_FFS_PARAMETER (identical to T_AUDIO_AMR_PLAY_FROM_FFS_PARAMETER)**

10.10 audio_mms_play_from_ffs_stop

```
T_AUDIO_RET audio_mms_play_from_ffs_stop (void)
```

Description

This function is called to stop playing an AMR file from FFS.

10.11 audio_amr_play_from_ffs_pause/ram_pause

```
T_AUDIO_RET audio_amr_play_from_ffs_pause (void)
```

```
T_AUDIO_RET audio_amr_play_from_ram_pause (void)
```

Description

This function is called to pause the AMR file playing from FFS or RAM. This shall be called only when the AMR file is playing. There is no confirmation message that is sent after file is paused.

10.12 audio_amr_play_from_ffs_resume/ram_resume

```
T_AUDIO_RET audio_amr_play_from_ffs_resume (void)
```

```
T_AUDIO_RET audio_amr_play_from_ram_resume (void)
```

Description

This function is called to resume the AMR file. This shall be called only when the AMR file is paused. There is no confirmation message that is sent after file is resumed to play.

10.13 audio_vm_amr_forward

```
T_AUDIO_RET audio_vm_amr_forward(UINT32 forward_skip_time)
```

Description

This function is called to forward the VM AMR play from FFS/RAM during play.

Parameters

- **UINT32 forward_skip_time**

This parameter gives in seconds the time for which the media that is played must be forwarded.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

There is no specific event that is returned for forward request. When the forward requested, reaches end of file, Audio VM AMR module stops playing the file and sends the confirmation as when stop is requested [see 10.6 and 10.8].

10.14 audio_vm_amr_rewind

```
T_AUDIO_RET audio_vm_amr_rewind(UINT32 forward_skip_time)
```

Description

This function is called to rewind the VM AMR play from FFS/RAM while playing.

Parameters

- **UINT32 rewind_skip_time**

This parameter gives in seconds the time for which the media that is played must be rewind.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

There is no specific event that is returned for rewind request. When the rewind requested, reaches the beginning of file, Audio VM AMR module starts to play again the same file from the beginning.

11 TTY

This chapter describes how to start/stop TTY services. These API can only be called when the mobile is in dedicated mode and vocoder is enabled.

11.1 audio_tty_set_config

```
T_AUDIO_RET audio_tty_set_config (T_AUDIO_TTY_CONFIG_PARAMETER *parameter,
            T_RV_RETURN *return_path)
```

Description

This function is called to change TTY configuration.

Parameters

- **T_AUDIO_TTY_CONFIG_PARAMETER**

Specifies the characteristics of tty configuration.

```
typedef struct {
  UINT8  Mode;
  UINT16 ThresholdRead;
  UINT16 ThreshHoldWrite;

}T_AUDIO_TTY_CONFIG_PARAMETER;
```

- **Mode** parameter defines an operation mode of the Baudot and CTM codecs. These configurations are currently used:

Value	Configuration
TTY_EXT_START (TTY external device End_To_End)	CodecOnCTM = TRUE CodecOnBaudot = TRUE TranscoderOn = TRUE
TTY_STOP	CodecOnCTM = FALSE CodecOnBaudot = FALSE TranscoderOn = FALSE

- **ThresholdRead** and **ThreshHoldWrite** are reserved for future use. They must be set to 0.

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Driver error (entity not started, not enough memory, bad parameters…)

Event Return

- **AUDIO_TTY_STATUS_MSG**

This event is the status sent at the end of TTY execution or if an error occurred.

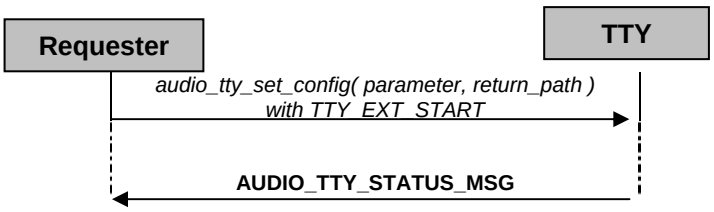
```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_TTY_STATUS;
```

The possible values of *status* are:

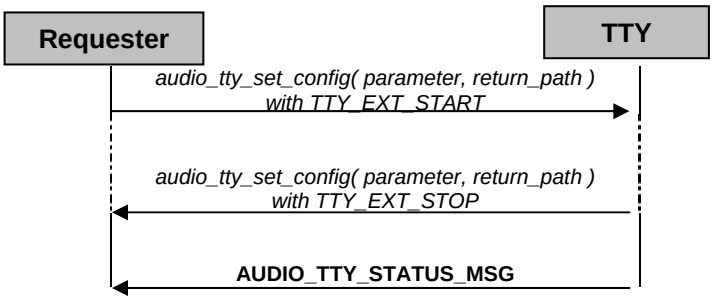
Value	Id	Definition
0	AUDIO_OK	The feature was successfully executed and stopped
-1	AUDIO_ERROR	Driver error

Process flow

Automatic stop:



Requested stop:



12 MP3

MP3 controls for MP3 file playing are:
- Start/Stop: start and stop playing MP3 file.
- Pause: pause playing.
- Resume: resume playing **after a pause**

Rules to respect:
The MMI must respect the following rules to play a MP3 file:
- The MMI isn't allowed to play a new MP3 before receiving an Audio MP3 Status.
- The MMI can receive a stop confirmation (Audio MP3 status) in the following cases:
  - The MMI requested to stop the playing (audio_mp3_stop function) and the Audio entity confirms this action
  - The MP3 decoding is finished and the Audio entity informs the MMI with a stop confirmation message.
  - An error occurred from while playing the MP3 file.
- After a pause request (audio_mp3_pause function), the MMI is allowed to request a resume or a stop playing (audio_mp3_resume or audio_mp3_stop).
- The MMI isn't allowed to use the start command during a pause: resume should be used instead.
- Resume commands has no effect outside pause mode.
- The MMI isn't allowed to request two pauses in a row .
- The MMI isn't allowed to request MP3 information before sending a start request (audio_mp3_start function) or after sending a stop request (audio_mp3_stop function).

This chapter describes how to play a MP3 melody, using the AUDIO SW entity service.

12.1 audio_mp3_start

```
T_AUDIO_RET audio_mp3_start (  T_AUDIO_MP3_PARAMETER  *p_parameter,
                               T_RV_RETURN            *p_return_path)
```

Description
This function is called to start a MP3 melody generation.

Parameters

- **T_AUDIO_MP3_PARAMETER**

```
typedef struct
{
  char    mp3_name[AUDIO_MP3_PATH_NAME_MAX_SIZE]; // File name of the melody
  BOOL    mono_stereo;                            // channel configuration
  UINT32  size_file_start;                        // size in bytes
  BOOLEAN   play_bar_on;                           // Progress-Bar configuration
} T_AUDIO_MP3_PARAMETER;
```

Below the detail of each parameter:

mp3_name Specifies the file name of the MP3 melody. Note that this file name is used by the audio entity to request the melody data to the File Flash System. Moreover, the file name must contain the entire path to access to the melody file.
Mono_stereo Specifies the configuration of the channel. If *Mono_stereo* = AUDIO_MP3_MONO, the channel configuration is Mono If *Mono_stereo* = AUDIO_MP3_STEREO, the channel configuration is Stereo
size_file_start **Specifies the size (in bytes) from where the melody should be started** **If size_file_start = 0, the melody is played from the beginning of the MP3 file** **If size_file_start = XXX, the melody is played from the byte XXX.**
play_bar_on **Specifies whether progress bar indication should be switched ON or not.** **When 0, the progress bar indication will be OFF** **When 1, the progress bar indication will be ON**

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

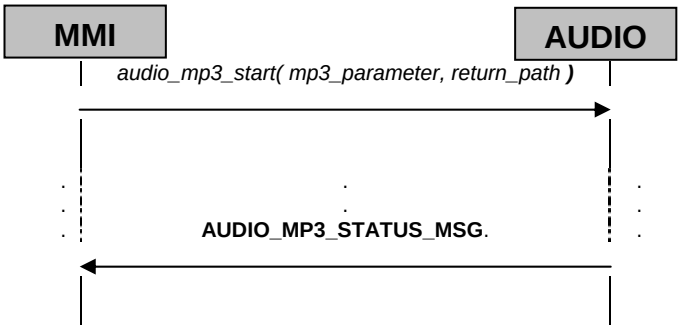
- **AUDIO_MP3_STATUS_BAR**

```
typedef struct
{
    T_RV_HDR os_hdr; /* Message id */
    INT8     status; /* status. This is for now AUDIO_OK always */
    UINT16  u16TotalTimePlayed; /* Current time played in seconds*/
    UINT16 u16totalTimeEst; /*The total song/file duration in seconds.  */
```

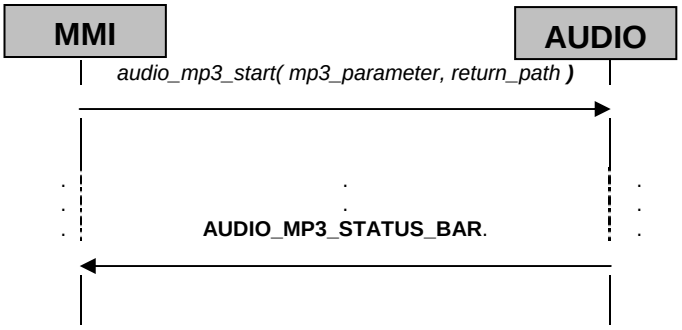
} T_AUDIO_MP3_STATUS_BAR;

status The possible value for status are, AUDIO_OK, indicates the function was successfully executed AUDIO_ERROR, indicates error (bad parameters, not enough memory, feature not compiled…)
u16TotalTimePlayed Specifies the actual total time played till now in milliseconds. This is calculated based on every frames decoded and is an accurate time played till now.
u16totalTimeEst Specifies the estimated total time played in milliseconds. This is estimated in the beginning of the play using the bit-rate of the first MP3 frame and the total size of the file.

Process flow



When Progress Bar is switched ON, AUDIO_MP3_STATUS_BAR message is sent (approximately) every 1 second.



Limitations

1. Configuration of progress bar update periodicity not supported. The update message will be sent at a pre-defined periodicity of ***approximately 1 second*** for both MP3 and AAC. The exact timing of the indication depends on the current frame bit-rate.
2. No separate API to configure the switch on/off of progress bar is supported. Progress bar update can be switched on/off only during the start request.
3. Progress bar update will not function properly when the start request is given from middle of the file (using size_file_start).

12.2 audio_mp3_stop

```
T_AUDIO_RET audio_mp3_stop (UINT32 *size_played)
```

Description

This function is called to stop playing a MP3 melody.

Parameters

- **UINT32 *size_played**

This parameter returns the size of the file that has been played before the audio_mp3_stop function was called. This size is in bytes.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

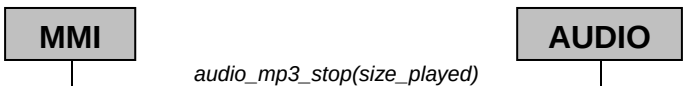
Event Return (if start API was called)

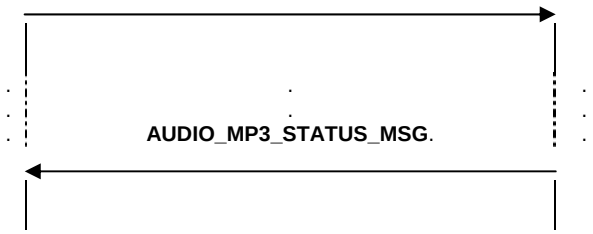
- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT16       status;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id
0	AUDIO_OK
0x0002	C_MP3_SYNC_NOT_FOUND
0x0004	C_MP3_NOT_LAYER3
0x0008	C_MP3_FREE_FORMAT
0x0010	C_MP3_ALG_ERROR
0x0020	C_MP3_DECODING_DELAY
0x04000	C_MP3_CHECK_BUFFER_KO
0x08000	C_MP3_CHECK_BUFFER_DELAY





12.3 audio_mp3_pause

```
T_AUDIO_RET audio_mp3_pause (void)
```

Description

This function is called to pause playing a MP3 melody.

The MP3 melody can be restarted using the audio_mp3_resume function.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

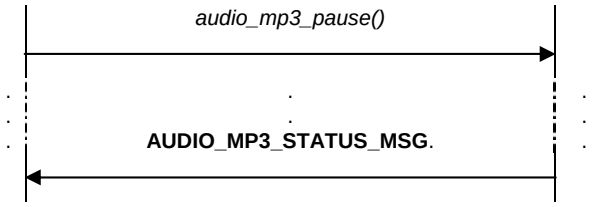
Event Return (if start API was called)

- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)



12.4 audio_mp3_resume

```
T_AUDIO_RET audio_mp3_resume (void)
```

Description

This function is called to resume a MP3 melody, after a pause.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

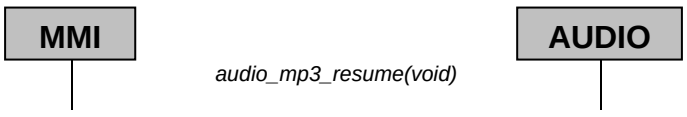
Event Return (if start API was called)

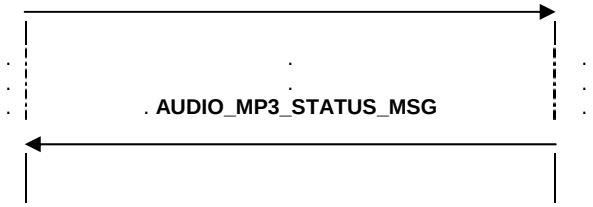
- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)





12.5 audio_mp3_forward

```
T_AUDIO_RET audio_mp3_forward (UINT32 forward_skip_time)
```

Description

This function is called to forward the MP3 melody during play.

Parameters

- **UINT32 forward_skip_time**

This parameter gives in seconds the time for which the media that is played must be forwarded.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

There is no specific event that is returned for forward request. When the forward requested, reaches end of file, Audio MP3 module starts to play again the same file from the beginning.

12.6 audio_mp3_rewind

```
T_AUDIO_RET audio_mp3_rewind (UINT32 rewind_skip_time)
```

Description

This function is called to rewind the MP3 melody during play.

Parameters

- **UINT32 rewind_skip_time**

This parameter gives in seconds the time for which the media that is played must be rewind.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

There is no specific event that is returned for rewind request. When rewind requested, goes before the beginning of file, Audio MP3 module starts again to play the same file from the beginning.

12.7 audio_mp3_info

```
T_AUDIO_RET audio_mp3_info (void)
```

Description

This function is called to request information about the currently decoded MP3 frame .

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return (if start API was called)

- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
  T_MP3_HEADER_INFO info;
} T_AUDIO_MP3_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.

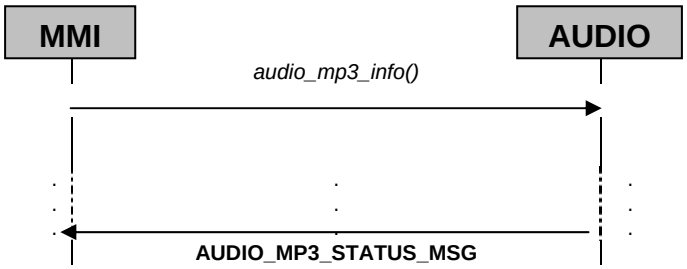
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

info (T_MP3_HEADER_INFO)

Structure containing information about current MP3 frame:

Field	Type	Possible values
frequency	UINT8	C_MP3_HEADER_FREQ_48000 C_MP3_HEADER_FREQ_44100 C_MP3_HEADER_FREQ_32000 C_MP3_HEADER_FREQ_24000 C_MP3_HEADER_FREQ_22050 C_MP3_HEADER_FREQ_16000 C_MP3_HEADER_FREQ_12000 C_MP3_HEADER_FREQ_11025 C_MP3_HEADER_FREQ_8000
bitrate	UINT8	C_MP3_HEADER_BITRATE_320 C_MP3_HEADER_BITRATE_256 C_MP3_HEADER_BITRATE_224 C_MP3_HEADER_BITRATE_192 C_MP3_HEADER_BITRATE_160 C_MP3_HEADER_BITRATE_128 C_MP3_HEADER_BITRATE_112 C_MP3_HEADER_BITRATE_96 C_MP3_HEADER_BITRATE_80 C_MP3_HEADER_BITRATE_64 C_MP3_HEADER_BITRATE_56 C_MP3_HEADER_BITRATE_48 C_MP3_HEADER_BITRATE_40 C_MP3_HEADER_BITRATE_32
mpeg_id	UINT8	C_MP3_HEADER_MPEGID_1 C_MP3_HEADER_MPEGID_2 C_MP3_HEADER_MPEGID_2_5
layer	UINT8	C_MP3_HEADER_LAYER_I C_MP3_HEADER_LAYER_II C_MP3_HEADER_LAYER_III Note: different layer values than LAYER III can be returned in theory. But this is a LAYER III only decoder.
padding	BOOL	TRUE, FALSE
private	UINT8	0, 1
channel	UINT8	C_MP3_HEADER_STEREO C_MP3_HEADER_JSTEREO C_MP3_HEADER_DUAL_MONO C_MP3_HEADER_MONO
copyright	BOOL	TRUE, FALSE

emphasis	UINT8	C_MP3_HEADER_EMPHASIS_NONE C_MP3_HEADER_EMPHASIS_50_15 C_MP3_HEADER_EMPHASIS_CCIT_J17



12.8 How to use the MP3 APIs

12.8.1 "Pause" MP3 in order to play an other melody

If the user wants to "pause" the MP3 melody in order to play an other melody (for example Midi ringer), it is necessary to stop the MP3 melody and then restart it. Indeed, due to Hardware constraints the MP3 can't be in "pause" mode when an other melody is playing.

Example of code – the user is listening to a MP3 melody when a Midi ringer needs to be played:

- Start the MP3 melody from the beginning:

```
strcpy(mp3_parameter.mp3_name,"/mp3/mp3_file");
mp3_parameter.mono_stereo = AUDIO_MP3_MONO;
mp3_parameter.size_file_start = 0;

if (audio_mp3_start(&mp3_parameter, return_path) == AUDIO_ERROR)
 {
        *error_type = FUNCTION_ERROR;
        return (audio_test_regr_return_verdict(*error_type));
}
```

- Midi Ringer must be played

  - First Stop the MP3 melody:

```
if (audio_mp3_stop(size_played) == AUDIO_ERROR)
{
        *error_type = MEMORY_ERROR;
         return (audio_test_regr_return_verdict(*error_type));
 }
```

- Play the Midi ringer
Use the start function to resume the MP3 melody
*mp3_parameter.size_file_start = *size_played;*

```
if (audio_mp3_start(&mp3_parameter, return_path) == AUDIO_ERROR)
{
    *error_type = FUNCTION_ERROR;
    return (audio_test_regr_return_verdict(*error_type));
}
```

12.8.2 “Pause” MP3 and resume it

If the user want to pause the MP3 melody and no melody is played in parallel:

- Start the MP3 melody from the beginning:

```
strcpy(mp3_parameter.mp3_name,"/mp3/mp3_file");
mp3_parameter.mono_stereo = AUDIO_MP3_MONO;
mp3_parameter.size_file_start = 0;

if (audio_mp3_start(&mp3_parameter, return_path) == AUDIO_ERROR)
{
    *error_type = FUNCTION_ERROR;
    return (audio_test_regr_return_verdict(*error_type));
}
```

- Pause the MP3 melody:

```
if (audio_mp3_pause() == AUDIO_ERROR)
{
  *error_type = FUNCTION_ERROR;
  return (audio_test_regr_return_verdict(*error_type));
}
```

- And then resume the MP3 melody thanks to the resume function:

```
if (audio_mp3_resume() == AUDIO_ERROR)
{
    *error_type = FUNCTION_ERROR;
    return (audio_test_regr_return_verdict(*error_type));
}
```

13 AAC (Advanced Audio Coding)

This chapter describes how to play an AAC melody, using the AUDIO SW entity service.

AAC controls for AAC file playing are:

- Start/Stop: start and stop playing AAC file.
- Pause: pause playing.
- Resume: resume playing **after a pause**
- Restart: reinitialize AAC decoder and restart playing **after a pause**. The re-initialization allows upper layer to move in the AAC file, which can implement a fast forward, fast backward solution for ADTS stream. In case of ADIF stream after re-initialization it starts playing from the beginning of the AAC file.

Note: In case of ADIF streams, it is the upper layers' responsibility to refill the buffers with the beginning of file before a restart.

Rules to respect:
The MMI must respect the following rules to play an AAC file:

- The MMI isn't allowed to play a new AAC file before receiving a stop confirmation (Audio AAC status).
- The MMI can receive a stop confirmation (Audio AAC status) in the following cases:
  - The MMI requested to stop the playing (audio_aac_stop function) and the Audio entity confirms this action
  - The AAC decoding is finished and the Audio entity informs the MMI with a stop confirmation message.
  - An error occurred from while playing the AAC file.

- Definitions:
  - Before any AAC activity, the Audio entity is in "idle mode".
  - When the MMI sends an AAC start command (audio_aac_start function), the Audio entity goes to "play mode".
  - When the MMI sends an AAC pause command (audio_aac_pause function), the Audio entity goes to "pause mode".
  - When the MMI sends an AAC resume or an AAC restart command (audio_aac_resume function or audio_aac_restart function), the Audio entity goes back to "play mode".
  - Whenever the MMI receives a stop confirmation (Audio AAC status), the Audio entity goes back to "idle mode".

- The MMI is only allowed to use the start command (_audio_aac_start function) while the Audio entity is in "idle mode".
- The MMI is only allowed to request AAC information (audio_aac_info function) while the Audio entity is in "play mode".
- The MMI is only allowed to use the pause command (audio_aac_pause function) while the Audio entity is in "play mode".
- The MMI is only allowed to use resume and restart commands (audio_aac_resume function or audio_aac_restart function) while the Audio entity is in "pause mode".
- The MMI can send a stop request (audio_aac_stop function) while the Audio entity is in "play mode" or in "pause mode".

- AAC playing isn't compatible with any other melody feature. Please refer to L1_AS328 specification for the cross-functionality table.

13.1 audio_aac_start

```
T_AUDIO_RET audio_aac_start (  T_AUDIO_AAC_PARAMETER  *p_parameter,
                               T_RV_RETURN            return_path)
```

Description

This function is called to start an AAC melody generation.

Parameters

- **T_AUDIO_AAC_PARAMETER**

```
typedef struct
{
  char    aac_name[AUDIO_AAC_PATH_NAME_MAX_SIZE]; // File name of the melody
  BOOLEAN mono_stereo;                            // channel configuration
  UINT32  size_file_start;                        // size in bytes
  BOOLEAN   play_bar_on;            // Progress Bar configuration
} T_AUDIO_AAC_PARAMETER;
```

Below the detail of each parameter:

aac_name Specifies the file name of the AAC melody. Note that this file name is used by the audio entity to request the melody data to the File Flash System. Moreover, the file name must contain the entire path to access to the melody file.
Mono_stereo Specifies the configuration of the channel. If *Mono_stereo* = AUDIO_AAC_MONO, the channel configuration is Mono. If *Mono_stereo* = AUDIO_AAC_STEREO, the channel configuration is Stereo.
size_file_start Specifies the size (in bytes) from where the melody should be started. If size_file_start = 0, the melody is played from the beginning of the AAC file. If size_file_start = XXX, the melody is played from the byte XXX. **Note : in Case of ADIF streams, the size_file_start must be equal to 0.**
play_bar_on **Specifies whether progress bar indication should be switched ON or not.** **When 0, the progress bar indication will be OFF** **When 1, the progress bar indication will be ON**

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return

- **AUDIO_AAC_STATUS**

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT32      status;
  UINT8     aac_format;
} T_AUDIO_AAC_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

The possible values of aac_format are:

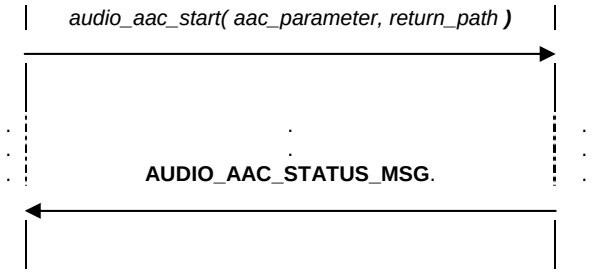
Value	Id	Definition
0	AUDIO_AAC_ADIF	Specifies the ADTS format of the AAC stream.
1	AUDIO_AAC_ADTS	Specifies the ADIF format of the AAC stream.

- **AUDIO_AAC_STATUS_BAR**

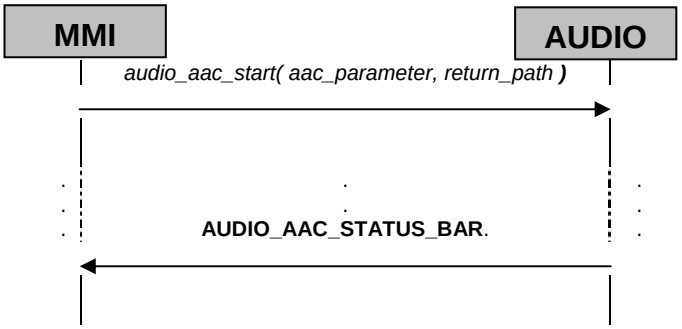
```
typedef struct
{
    T_RV_HDR os_hdr; /* Message id */
    INT8     status; /* status. This is for now AUDIO_OK always */
    UINT16  u16TotalTimePlayed; /* Current time played in seconds*/
    UINT16 u16totalTimeEst; /*The total song/file duration in seconds.  */
} T_AUDIO_AAC_STATUS_BAR;
```

status The possible value for status are, AUDIO_OK, indicates the function was successfully executed AUDIO_ERROR, indicates error (bad parameters, not enough memory, feature not compiled…)
u16TotalTimePlayed Specifies the actual total time played till now in milliseconds. This is calculated based on every frames decoded and is an accurate time played till now.
u16totalTimeEst Specifies the estimated total time played in milliseconds. This is estimated in the beginning of the play using the bit-rate of the first AAC frame and the total size of the file.

Process flow



When Progress Bar is switched ON, AUDIO_AAC_STATUS_BAR message is sent (approximately) every 1 second.



Limitations

1. Configuration of progress bar update periodicity not supported. The update message will be sent at a pre-defined periodicity of ***approximately 1 second*** for both MP3 and AAC. The exact timing of the indication depends on the current frame bit-rate.
2. No separate API to configure the switch on/off of progress bar is supported. Progress bar update can be switched on/off only during the start request.
3. Progress bar update will not function properly when the start request is given from middle of the file (using size_file_start).

13.2 audio_aac_stop

```
T_AUDIO_RET audio_aac_stop (UINT32 *size_played)
```

Description

This function is called to stop playing an AAC melody.

Parameters

- **UINT32 *size_played**

This parameter returns the size of the file that has been played before the audio_aac_stop function was called. This size is in bytes.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return (if start API was called)

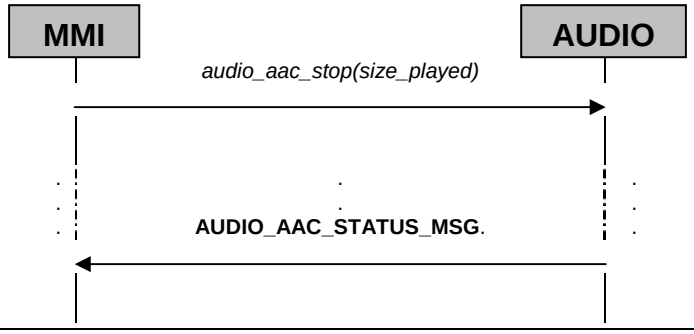
- **AUDIO_AAC_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT32       status;
} T_AUDIO_AAC_STATUS;
```

The possible values of status are:

Value	Id
0	No error
Else following bits equal to 1 means:	
bit 1 (0x0000 0002)	AAC_SYNC_WORD_NOT_FOUND
bit 2 (0x0000 0004)	AAC_ADTS_HEADER_HAS_INVALID_SYNCWORD
bit 3 (0x0000 0008)	AAC_ADTS_LAYER_DATA_ERROR
bit 4 (0x0000 0010)	AAC_NUM_CHANNELS_EXCEEDED
bit 5 (0x0000 0020)	AAC_PREDICTION_DETECTED
bit 6 (0x0000 0040)	AAC_LFE_CHANNEL_DETECTED
bit 7 (0x0000 0080)	AAC_GAIN_CONTROL_DETECTED
bit 8 (0x0000 0100)	AAC_CHANNEL_ELEMENT_PARSE_ERROR
bit 9 (0x0000 0200)	AAC_PULSE_DATA_NOT_ALWD_SHORT_BLK
bit 10 (0x0000 0400)	AAC_MAX_SFB_TOO_LARGE_SHORT
bit 11 (0x0000 0800)	AAC_MAX_SFB_TOO_LARGE_LONG
bit 12 (0x0000 1000)	AAC_ERROR_ON_DATA_CHANNEL
bit 13 (0x0000 2000)	AAC_COUPLING_CHANNEL_DETECTED
bit 14 (0x0000 4000)	AAC_ADTS_PROFILE_ERROR
bit 15 (0x0000 8000)	AAC_ADIF_PROFILE_ERROR
bit 16 (0x0001 0000)	AAC_INVALID_ELEMENT_ID
bit 17 (0x0002 0000)	AAC_SAMP_FREQ_NOT_SUPPORTED
bit 18 (0x0004 0000)	AAC_DSP_DELAY
bit 19 (0x0008 0000)	AAC_MCU_COPY_DELAY
bit 20 (0x0010 0000)	AAC_MCU_FILL_DELAY
bit 21 (0x0020 0000)	AAC_ERR_DMA_DROP
bit 22 (0x0040 0000)	AAC_ERR_DMA_TOUT_SRC
bit 23 (0x0080 0000)	AAC_ERR_DMA_TOUT_DST
bit 24 (0x0100 0000)	AAC_WARNING_DMA_IT_MASKED

Several errors can occur at the same time. The error code returned is a logical or of the different error codes that have occurred.



13.3 audio_aac_pause

```
T_AUDIO_RET audio_aac_pause (void)
```

Description

This function is called to pause playing an AAC melody.

The AAC melody can be restarted using the audio_aac_resume function or the audio_aac_restart function. AAC can also be stopped while in pause using the audio_aac_stop function.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

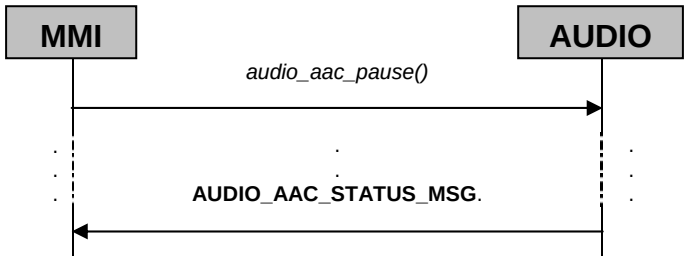
Event Return (if start API was called)

- **AUDIO_AAC_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT32       status;
} T_AUDIO_AAC_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)



13.4 audio_aac_resume

```
T_AUDIO_RET audio_aac_resume (void)
```

Description

This function is called to resume an AAC melody, after a pause.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

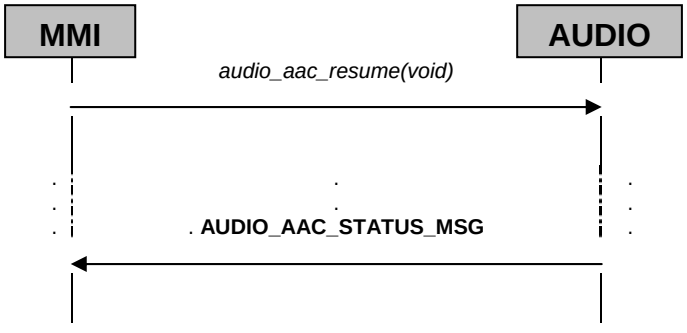
Event Return (if start API was called)

- **AUDIO_AAC_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT32      status;
} T_AUDIO_AAC_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)



13.5 audio_aac_restart

```
T_AUDIO_RET audio_aac_restart (UINT32 size_file_restart)
```

Description

This function is called to restart an AAC melody, after a pause.

Difference with resume is: the AAC decoder is reinitialized. Input buffers containing subsets of AAC files are refilled. It allows upper layers to restart playing at any point in the AAC file.

Note : In case of ADIF streams, it is the upper layers' responsibility to refill the buffers with the beginning of the file before a restart.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

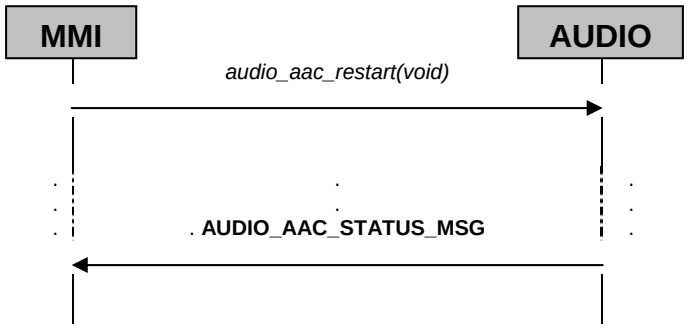
Event Return (if start API was called)

- **AUDIO_AAC_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT32      status;
} T_AUDIO_AAC_STATUS;
```

The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)



13.6 audio_aac_info

```
T_AUDIO_RET audio_aac_info (void)
```

Description

This function is called to request information about the currently decoded AAC raw data block. The MMI is only allowed to call this function while the Audio entity is in “play mode”.

Immediate Return

- **T_AUDIO_RET**

The immediate value returned is defined as:

```
typedef INT8 T_AUDIO_RET;
```

The possible values are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

Event Return (if start API was called)

- **AUDIO_MP3_STATUS**

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT32       status;
  T_L1A_AAC_INFO_CON info;
} T_AUDIO_AAC_INFO_STATUS;
```

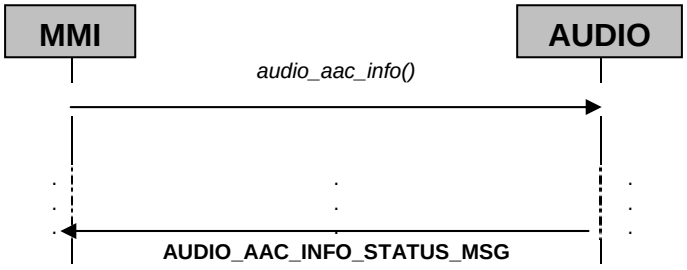
The possible values of status are:

Value	Id	Definition
0	AUDIO_OK	The API function was successfully executed.
-1	AUDIO_ERROR	Error (bad parameters, not enough memory, feature not compiled…)

info (T_L1A_AAC_INFO_CON)

Structure containing information about current AAC frame:

frequency	UINT16	0 : AAC_STREAM_FREQ_48000 1 : AAC_STREAM _FREQ_44100 2 : AAC_STREAM _FREQ_32000 3 : AAC_STREAM _FREQ_24000 4 : AAC_STREAM _FREQ_22050 5 : AAC_STREAM _FREQ_16000 6 : AAC_STREAM _FREQ_12000 7 : AAC_STREAM _FREQ_11025 8 : AAC_STREAM _FREQ_8000 **Note:** The hardware does not support sampling frequency of 64000 Hz, 88200 Hz and 96000 Hz.
bitrate	UINT32	Values ranging from 16000 bps to 576000 bps.
channel	UINT8	1 : AAC_STREAM _MONO 2 : AAC_STREAM _STEREO
aac_format	UINT8	0 : AAC_ADIF_FORMAT 1 : AAC_ADTS_FORMAT



13.7 How to use the AAC APIs

13.7.1 “Pause” AAC in order to play an other melody

If the user wants to “pause” the AAC melody in order to play an other melody (for example Midi ringer), it is necessary to stop the AAC melody and then start it again. Indeed, due to Hardware constraints the AAC can’t be in “pause” mode when an other melody is playing.
Example of code – the user is listening to a AAC melody when a Midi ringer needs to be played:

- Start the AAC melody from the beginning:

```
strcpy(aac_parameter.aac_name,"/aac/aac_file");
aac_parameter.mono_stereo = AUDIO_AAC_MONO;
aac_parameter.size_file_start = 0;

if (audio_aac_start(&aac_parameter, return_path) == AUDIO_ERROR)
 {
     *error_type = FUNCTION_ERROR;
     return (audio_test_regr_return_verdict(*error_type));
 }
```

- Midi Ringer must be played

- First Stop the AAC melody:

```
if (audio_aac_stop(size_played) == AUDIO_ERROR)
{
     *error_type = MEMORY_ERROR;
     return (audio_test_regr_return_verdict(*error_type));
}
```

- Play the Midi ringer
- Use the start function to resume the AAC melody

```
aac_parameter.size_file_start = *size_played;

if (audio_aac_start(&aac_parameter, return_path) == AUDIO_ERROR)
 {
   *error_type = FUNCTION_ERROR;
   return (audio_test_regr_return_verdict(*error_type));
 }
```

13.7.2 “Pause” AAC and resume it

If the user want to pause the AAC melody and no melody is played in parallel:

- Start the AAC melody from the beginning:

```
strcpy(aac_parameter.aac_name,"/aac/aac_file");
aac_parameter.mono_stereo = AUDIO_AAC_MONO;
aac_parameter.size_file_start = 0;

if (audio_aac_start(&aac_parameter, return_path) == AUDIO_ERROR)
 {
     *error_type = FUNCTION_ERROR;
     return (audio_test_regr_return_verdict(*error_type));
 }
```

- Pause the AAC melody:

```
if (audio_aac_pause() == AUDIO_ERROR)
{
  *error_type = FUNCTION_ERROR;
  return (audio_test_regr_return_verdict(*error_type));
}
```

- And then resume the AAC melody thanks to the resume function:

```
if (audio_aac_resume() == AUDIO_ERROR)
{
   *error_type = FUNCTION_ERROR;
   return (audio_test_regr_return_verdict(*error_type));
}
```

14 Voice memorization on PCM

This chapter describes how to use the voice memorization on PCM api's, using the AUDIO software entity service.

14.1 audio_vm_pcm_record_start

```
T_AUDIO_RET audio_vm_pcm_record_start (
            T_AUDIO_VM_PCM_RECORD_PARAMETER *p_record_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to initiate the recording for voice memorization on PCM.

Parameters

- **T_AUDIO_VM_PCM_RECORD_PARAMETER**

Specifies the parameters using during the voice memorization phase.

```
typedef struct {
  char           memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];
  UINT32         memo_duration;       // Maximum duration of the voice memo
  UINT16         microphone_gain;     // Recording gain applies to microphone
  UINT16         network_gain;        // Gain applies to the network voice
}T_AUDIO_VM_PCM_RECORD_PARAMETER;
```

Below the detail of each parameter:

memo_name Specifies the file name of the voice memo. Note that this file name is used by the audio entity to write the memo data to the Flash file System. Moreover, the file name must contain the entire path of the memo file. Note the maximum size of the path plus the name is 20 characters.
memo_duration Specifies the duration of the voice memo in seconds.
microphone_gain Specifies the gain multiplied to the voice sample from the microphone. The values allowed are 0(to DISABLE recording of PCM samples from the microphone) and 0x20(to ENABLE recording of PCM samples from the microphone).
network_gain Specifies the gain multiplied to the voice sample from the network (if the mobile is in dedicated mode). The values allowed are 0(to DISABLE recording of PCM samples from the network) and 0x20(to ENABLE recording of PCM samples from the network).

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_VM_PCM_RECORD_STATUS_MSG**

This event indicates that the melody task is stopped or an error is occurred.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
  UINT16     recorded_duration;
}T_AUDIO_VM_PCM_RECORD_STATUS;
```

Below the detail of the parameter:

recorded_duration Specifies the size in seconds' unit of the recorded data.

The possible values of *status* are:

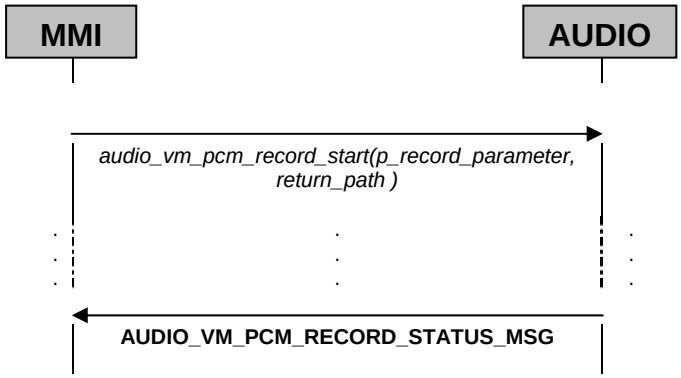
Value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed

Current restriction of use

The following restriction of use MUST BE followed by the entity. **If it isn't the case, the good functionality of the complete system isn't guarantee. Note: these following restrictions are only available in the latest version of the software.**

- An entity isn't allowed to call this API function if the following audio features is running:
  - A melody E1.
  - A tone.
  - A speech recognition (enrollment, update, update-check, recognition).
  - A voice memorization playing.
  - Note: this restriction is managed by the audio entity, therefore the Voice memorization (PCM) recording isn't started if the speech recognition or voice memorization playing or tone or melody E1 is running.
- All directories included in the pathname must be declared before
-

Process flow



14.2 audio_vm_pcm_record_stop

```
T_AUDIO_RET audio_vm_pcm_record_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the current voice memorization PCM record.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

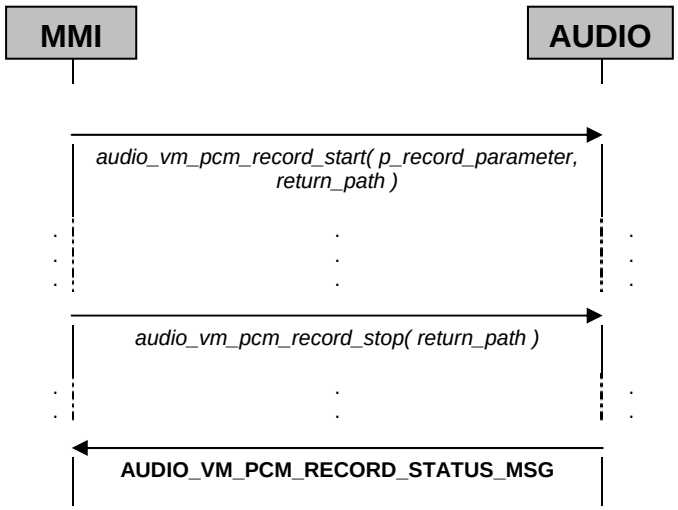
- **AUDIO_VM_RECORD_STATUS_MSG**

C.f. API function *audio_vm_pcm_record_start*.

Current restriction of use

C.f. API function *audio_vm_pcm_record_start*.

Process flow



14.3 audio_vm_pcm_play_start

```
T_AUDIO_RET audio_vm_pcm_play_start (T_AUDIO_VM_PCM_PLAY_PARAMETER
            *p_parameter, T_RV_RETURN return_path)
```

Description

This function is called to initiate the voice memorization PCM playing phase.

Parameters

- **T_AUDIO_VM_PCM_PLAY_PARAMETER**

Specifies the parameters using during the voice memorization phase.

```
typedef struct {
  char    memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];
  UINT32  memo_duration;    /* maximum duration of the voice memo played*/
  UINT16  speaker_gain;     /* play gain applied to speaker. Sent as up
                               load_ul_gain to L1*/
  UINT16  network_gain;     /* play gain applied to play to the network. Sent
                               as upload_dl_gain to L1*/
}T_AUDIO_VM_PCM_PLAY_PARAMETER;
```

Below the detail of each parameters:

memo_name Specifies the file name of the voice memo. Note that this file name is used by the audio entity to request the memo data to the File Flash System. Moreover, the file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 20 characters.
memo_duration Specifies the duration of the voice memo to be played in seconds.
microphone_gain Specifies the gain multiplied to the voice sample played to the speaker. The values allowed are 0(to DISABLE playing of PCM samples to the speaker) and 0x20(to ENABLE playing of PCM samples to the speaker).
network_gain Specifies the gain multiplied to the voice sample from the network (if the mobile is in dedicated mode). The values allowed are 0(to DISABLE playing of PCM samples to the network) and 0x20(to ENABLE playing of PCM samples to the network).

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_VM_PCM_PLAY_STATUS_MSG**

This event indicates that the voice memorization playing task is stopped or an error is occurred.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_VM_PCM_PLAY_STATUS;
```

The possible values of *status* are:

Value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed

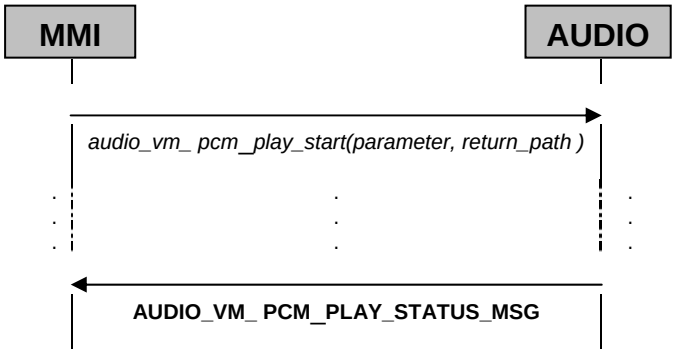
Current restriction of use

The following restriction of use MUST BE followed by the entity. **If it isn't the case, the good functionality of the complete system isn't guarantee. Note: these following restrictions are only available in the latest version of the software.**

- An entity isn't allowed to call this API function if the following audio features is running:
  - A melody E1.
  - A speech recognition (enrollment, update, update-check, recognition).
  - A voice memorization recording.

  Note: this restriction is managed by the audio entity, therefore the voice memorization PCM playing isn't started if the speech recognition or voice memorization playing is running.
- All directories included in the pathname must be declared before

Process flow



14.4 audio_vm_pcm_play_stop

```
T_AUDIO_RET audio_vm_pcm_play_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the current voice memorization PCM play.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

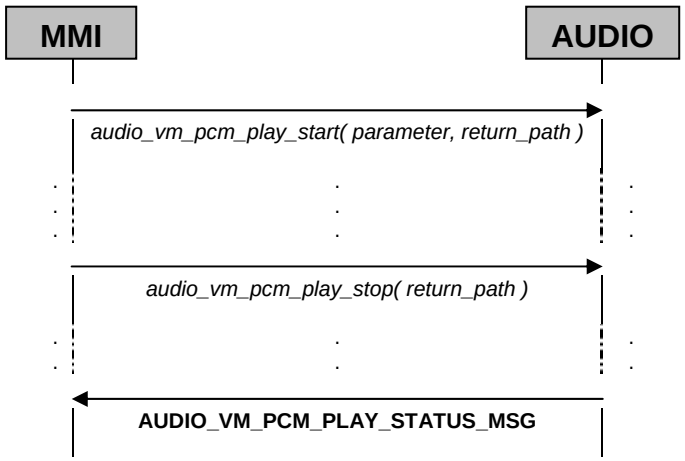
- **AUDIO_VM_PCM_PLAY_STATUS_MSG**

C.f. API function *audio_vm_pcm_play_start*.

Current restriction of use

C.f. API function *audio_vm_pcm_play_start*.

Process flow



15 Voice Buffering on PCM

This chapter describes how to use the voice memorization buffering, using the AUDIO software entity service.

15.1 audio_voice_buffering_pcm_record_start

```
T_AUDIO_RET audio_voice_buffering_pcm_record_start (
```

```
            T_AUDIO_VBUF_PCM_RECORD_PARAMETER *p_record_parameter,
            T_RV_RETURN return_path)
```

Description

This function is called to initiate the recording of voice as PCM samples from the microphone or from the network.

Parameters

- **T_AUDIO_VBUF_PCM_RECORD_PARAMETER**

Specifies the parameters used during the voice buffering recording phase.

```
typedef struct {
  char           memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];
  UINT32          memo_duration;       // maximum duration of the voice memo
  UINT16          microphone_gain;     // recording gain applies to microphone
  UINT16          network_gain;        // gain applies to the network voice
}T_AUDIO_VBUF_PCM_RECORD_PARAMETER;
```

Below the detail of each parameter:

memo_name Specifies the file name of the voice memo. Note that this file name is used by the audio entity to request the memo data to the File Flash System. Moreover, the file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 20 characters.
memo_duration Specifies the duration of the voice buffering in seconds.
microphone_gain Specifies the gain multiplied to the voice sample from the microphone. The values allowed are 0(to DISABLE recording of PCM samples from the microphone) and 0x20(to ENABLE recording of PCM samples from the microphone).
network_gain Specifies the gain multiplied to the voice sample from the network (if the mobile is in dedicated mode). The values allowed are 0(to DISABLE recording of PCM samples from the network) and 0x20(to ENABLE recording of PCM samples from the network).

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_VBUF_PCM_RECORD_STATUS_MSG**

This event indicates that the melody task is stopped or an error is occurred.

```
typedef struct {
  T_RV_HDR   os_hdr;
  INT8       status;
  UINT16     recorded_duration;
}T_AUDIO_VBUF_PCM_RECORD_STATUS;
```

Below the detail of the parameter:

recorded_duration Specifies the size in seconds' unit of the recorded data.

The possible values of *status* are:

Value	Id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed

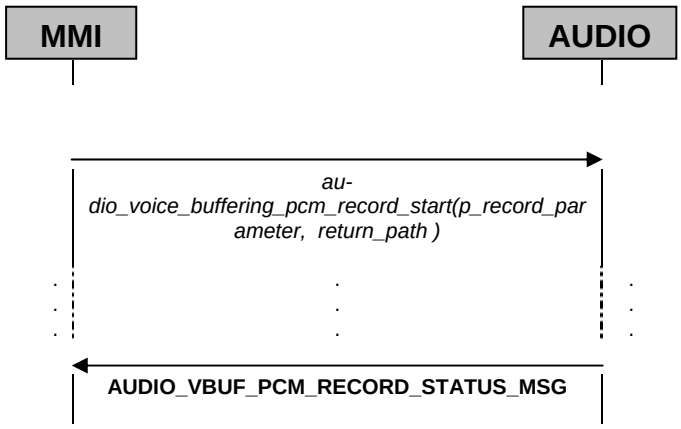
Current restriction of use

The following restriction of use MUST BE followed by the entity. **If it isn't the case, the good functionality of the complete system isn't guarantee. Note: these following restrictions are only available in the latest version of the software.**

- An entity isn't allowed to call this API function if the following audio features is running:
  - A melody E1.
  - A tone.
  - A speech recognition (enrollment, update, update-check, recognition).
  - A voice memorization playing.
  - Note: this restriction is managed by the audio entity, therefore the Voice memorization recording isn't started if the speech recognition or voice memorization playing or tone or melody E1 is running.

- All directories included in the pathname must be declared before

Process flow



15.2 audio_voice_buffering_pcm_record_stop

```
T_AUDIO_RET audio_voice_buffering_pcm_record_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the current voice buffering record.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

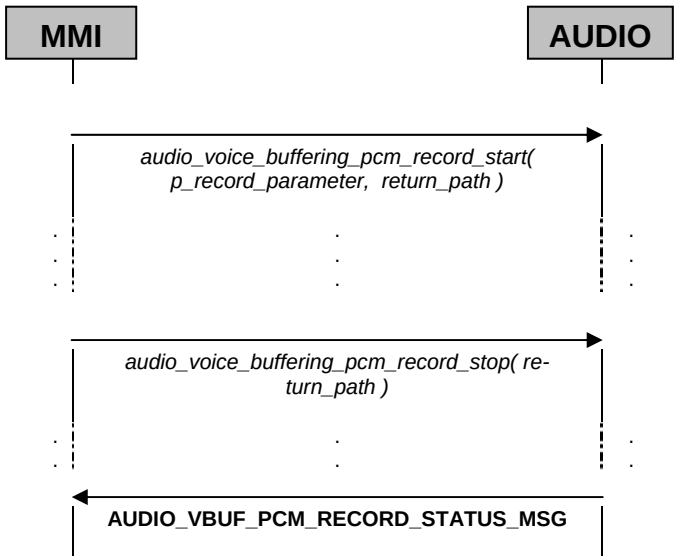
- **AUDIO_VBUF_PCM_RECORD_STATUS_MSG**

C.f. API function *audio_vm_pcm_record_start*.

Current restriction of use

C.f. API function *audio_vm_pcm_record_start*.

Process flow



15.3 audio_voice_buffering_pcm_play_start

```
T_AUDIO_RET audio_voice_buffering_pcm_play_start
            (T_AUDIO_VBUF_PCM_PLAY_PARAMETER *p_parameter, T_RV_RETURN
            return_path)
```

Description

This function is called to initiate the voice buffering playing phase.

Parameters

- **T_AUDIO_VBUF_PCM_PLAY_PARAMETER**

Specifies the parameters using during the voice memorization phase.

```
typedef struct {
  char    memo_name[AUDIO_MEMO_PATH_NAME_MAX_SIZE];
  UINT32  memo_duration;    /* maximum duration of the voice memo played*/
  UINT16  speaker_gain;     /* play gain applied to speaker. Sent as up
                               load_ul_gain to L1*/
  UINT16  network_gain;     /* play gain applied to play to the network. Sent
                               as upload_dl_gain to L1*/
} T_AUDIO_VBUF_PCM_PLAY_PARAMETER;
```

Below the detail of each parameters:

memo_name Specifies the file name of the voice memo. Note that this file name is used by the audio entity to request the memo data to the File Flash System. Moreover, the file name must contain the entire path to declare the memo file. Note the maximum size of the path plus the name is 20 characters.
memo_duration Specifies the duration of the voice memo to be played in seconds.
speaker_gain Specifies the gain multiplied to the voice sample played to the speaker. The values allowed are 0(to DISABLE playing of PCM samples to the speaker) and 0x20(to ENABLE playing of PCM samples to the speaker).
network_gain Specifies the gain multiplied to the voice sample from the network (if the mobile is in dedicated mode). The values allowed are 0(to DISABLE playing of PCM samples to the network) and 0x20(to ENABLE playing of PCM samples to the network).

- **T_RV_RETURN**

C.f. previous section (return mechanism).

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

- **AUDIO_VBUF_PCM_PLAY_STATUS_MSG**

This event indicates that the voice memorization playing task is stopped or an error is occurred.

```
typedef struct {
  T_RV_HDR  os_hdr;
  INT8      status;
}T_AUDIO_VBUF_PCM_PLAY_STATUS;
```

The possible values of *status* are:

Value	id	Definition
0	AUDIO_OK	The audio features was successfully executed and stopped.
-1	AUDIO_ERROR	The audio features was not successfully executed

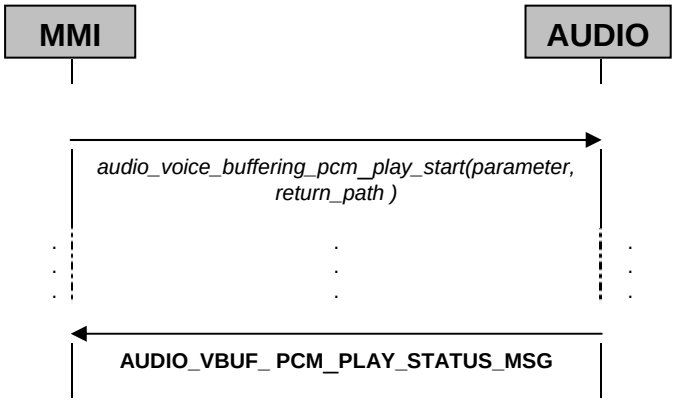
Current restriction of use

The following restriction of use MUST BE followed by the entity. **If it isn't the case, the good functionality of the complete system isn't guarantee. Note: these following restrictions are only available in the latest version of the software.**

- An entity isn't allowed to call this API function if the following audio features is running:
  - A melody E1.
  - A speech recognition (enrollment, update, update-check, recognition).
  - A voice memorization recording.

  Note: this restriction is managed by the audio entity, therefore the voice memorization playing isn't started if the speech recognition or voice memorization playing is running.
- All directories included in the pathname must be declared before

Process flow



15.4 audio_voice_buffering_pcm_play_stop

```
T_AUDIO_RET audio_voice_buffering_pcm_play_stop (T_RV_RETURN return_path)
```

Description

This function is called in order to stop the current voice memorization play.

Parameters

- **T_RV_RETURN**

C.f. section *return mechanism*.

Immediate Return

C.f. API function *audio_keybeep_start*.

Event Return

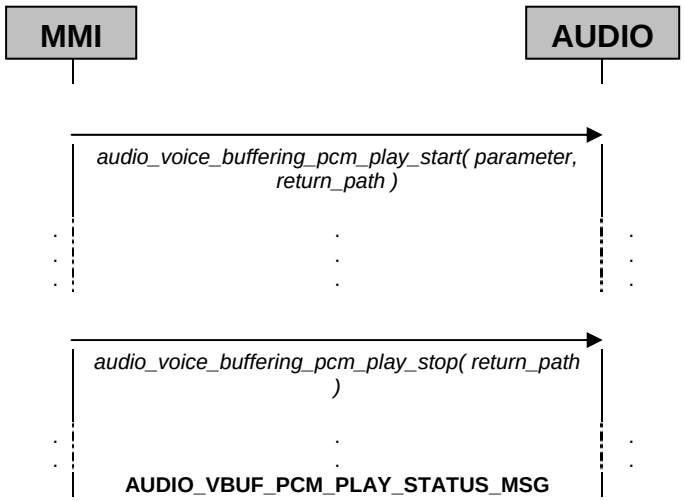
- **AUDIO_VBUF_PCM_PLAY_STATUS_MSG**

C.f. API function *audio_vm_pcm_play_start*.

Current restriction of use

C.f. API function *audio_vm_pcm_play_start*.

Process flow



Appendices

A. Acronyms

AAC	Advanced Audio Coding
ADTS	Audio Data Transport Stream
ADIF	Audio Data Interchange Format
AEC	Acoustic Echo Cancellation
FFS	Flash File System
FIR	Finite Impulse Response filter
MMI	Man Machine Interface
RFS	Riviera File System
SR	Speech Recognition

B. Glossary