BP30
Audio Driver Specification

Edition 2006

Published by Neonseven s.r.l.,
Viale Stazione di Prosecco, 15

34010 Sgonico (Trieste) Italy

For questions on technology, delivery and prices please contact the Neonseven Offices in Italy Sgonico and Gorizia

Attention Please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact Neonseven.

Neonseven technologies may only be used in life-support devices or systems with the express written approval of Neonseven, if a failure of such technologies can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Table of Contents

1 Document Mission/Scope

1.1 Mission

This document describes the interface to the audio driver on BP30 platform. The interface is a function interface.

1.2 Scope

This document is addressed to any SW developer who needs to use the audio drivers on BP30 platform.

2 List of Acronyms

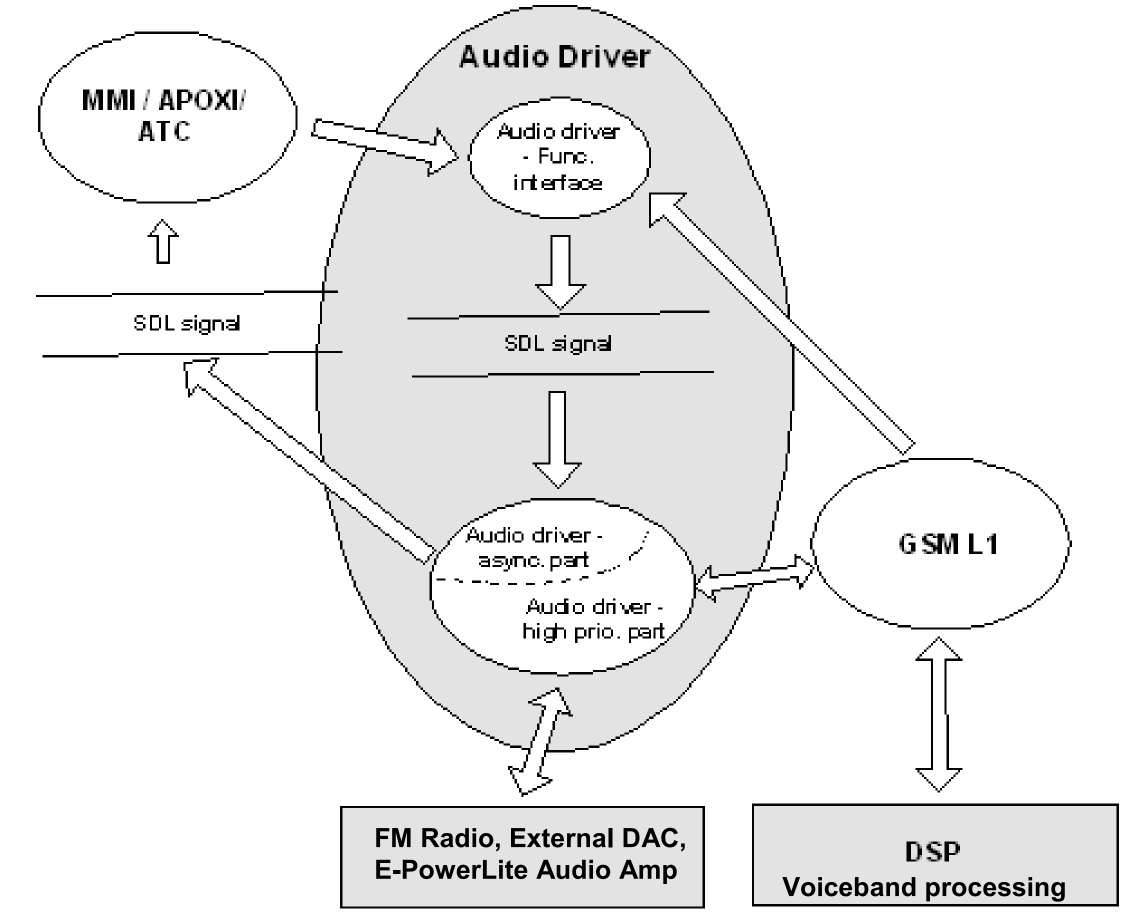
Abbreviation / Term	Explanation / Definition
ADC	Analog to Digital Converter
AFE	Audio Front End
AGC	Automatic Gain Control unit
AMR	Adaptive Multirate Speech-Codec
DAC	Digital to Analog Converter
DAI	Digital Audio Interface
DSP	Digital Signal Processor
EC	Echo Canceller unit
EFR	Enhanced Full Rate speech codec
FMR	Frequency Modulation Radio
FR	Full Rate speech codec
HF	HandsFree algorithm
HR	Half Rate speech codec
LMS	Least Mean Square algorithm
MIDI	Musical Instrument Digital Interface
MP3	MPEG Layer III
NR	Noise Reduction unit
PCM	Pulse Code Modulation
TCH	Traffic CHannel
VM	Voice Memo

3 Introduction

The audio driver is composed of a set of modules developed by DWD audio group [1] and adapted and tuned to BP30 platform by N7. These modules manage the HW audio resources of the platform. These resources are:

- Speech
- Tone generator
- FM radio
- VM recorder
- VM player
- MP3 player
- MIDI player
- PCM player

Every resource can be controlled by a set of interface functions. This introduction presents the interface function operation in a general way. Following paragraphs will illustrate in detail the sets of function provided for every resource. On figure 1 the interfaces are illustrated. When the MMI needs an audio service, it calls the appropriate function in the audio driver.



Since some of the functions, which the audio driver offers, uses the same HW blocks (from now on resources), some resource handling is necessary. This is accomplished by letting the MMI allocate the resource before using it. When a resource is allocated by a process, no other process will then be able to use the resource. For each of the resources a dedicated set of functions exists, these are all described in next chapters. When a resource has been successfully allocated, a handle is returned. This handle must be used whenever calling functions dedicated to the resource.

If the audio driver for some reason needs to inform a process that has allocated a resource, a SDL signal is sent to that process. This information could be the signaling of an occurred error or of the changed status of a state machine. There exist only one SDL-signal and name of the signal is AUD_DRIVER_RSP.

The parameters of the signal are:

Handle: The handle returned from AUD_allocate_resource(…)
Caller_id: The parameter passed in AUD_allocate_resource(…)
Func_id: The resource
Return_code: a return code which explains why the signal is sent. Return codes from –1 to –99 are errors, and the client should react on these codes. Return code from –100 to –127 are information code, and the client does not necessarily have to react.
Parm1 to parm6. These parameters are used in combination with the return code. Some return codes supply extra information in these parameters (i.e.: FM radio could return the frequency of a new found station the level or receiving , the signaling of mono or stereo transmission).
Even though only one signal exist it should be possible from the handle, func_id and return_code it should be possible for the client to figure out how to handle the signal.
In

there is an example that shows the principle in using the audio driver. The example enables audio for a normal speech call. All the audio driver interface functions used in the example are described in detail in next paragraphs.

```
/** This function activates or deactivates audio channel for tch
** and informs the clients that a handle was allocated or released              */

GLOBAL SDL_Void mn_aud_tch_on_off ( T_MN_CLIENT_STORAGE * clients_ptr,  SDL_Boolean           on_off )
{
      SINT8 aud_rc;
  if( on_off ) /*  need to activate audio channel*/
  {
      if( mnc_aud_speech_handle == 0 ) /* no handle available */
      {
           aud_rc = AUD_allocate_resource(0,  aud_resource_speech,  aud_priority_high);      /*allocate speech */
           if( aud_rc <= (SINT8)0 )                                   /*check if driver signals an error*/
           {
               ms_warn( MN_ERR_AUD_SPEECH );              /* this function report an error*/
               return (SDL_Void) 0;
           }
           mnc_aud_speech_handle = (UINT8)aud_rc;
           send_speech_handle_to_clients( clients_ptr, mnc_aud_speech_handle,on_off);  /* informs the clients that a handle was allocated */
      }
       if (mn_aud_tch_enabled == SDL_False )
       {
           if( AUD_speech_enable( mnc_aud_speech_handle) < 0 )       /*use the handle with a function for this resource*/
           {
               ms_warn( MN_ERR_AUD_SPEECH );
               return (SDL_Void) 0;
```

4 Interface description

In this chapter a detailed description, of the all interface functions which the audio driver to the "outside" world, is given.

4.1 Resource management

This section contains the description of general functions.
Important note:
If the interface function relies on a pointer in its parameters, e.g. pointing to melody data, this pointer must remain intact and constant through the life span of the data.

4.1.1 AUD_allocate_resource

Prototype: SINT8 AUD_allocate_resource(UINT16 id, aud_resource_enum resource, aud_priority_enum priority)
Parameters: aud_resource_enum resource
aud_priority_enum priority
Returns: Return code, see chapter 0

Description: Reserve the requested resource if it is free. If the resource is already allocated, an error code is returned. If the resource is available, it is checked that using the resource, does not conflict with any of the other allocated resources, if it does an error code is returned. If the priority is set to high, the resource is always allocated, even though another module already has allocated it (the module which already has the control of the resource will then be informed that the resource it has allocated, has been taken over by another module). Only two priority levels exist - normal and high. The id parameter is the id of the calling task. The audio will use this if it needs to send information back to the calling task. If everything is OK, (i.e. the resource can be allocated) a handle is returned (a number > 0). This handle must be used whenever a request for an audio service is made.

Enum: aud_priority_enum	
Enum name	Description
aud_priority_normal	Normal priority
aud_priority_high	High priority (normally used for test)
aud_priority_end	

Enum: aud_resource_enum		
Enum name	Description	Available in BP30 platform
aud_resource_speech	Speech encoder/decoder	Yes
aud_resource_tone_generator	Internal tone generator	Yes
aud_resource_ringer	External ringer (dedicated Yamaha chip)	No
aud_resource_vibrator	Vibrator (if drived by external ringer, synchronous with ringer)	No *
aud_resource_amplifier	External amplifier for speaker phone functionality	Yes
aud_resource_radio	FM/AM radio	Yes
aud_resource_record_vm	Voice memo recorder	Yes
aud_resource_playback_vm	Voice memo player	Yes
aud_resource_playback_mp3	Internal mp3 player	Yes
aud_resource_PCM_channel	I2Sy interface	No **
aud_resource_midi_player	Internal midi player	Yes
aud_resource_tty	Tty/Ctm functionality	Yes
aud_resource_playback_pcm	Playback PCM samples	Yes
aud_resource_record_pcm	Recording PCM samples	No
aud_resource_mapi	Yamaha ext. ringer Music API interface	No
aud_resource_end	Enum end	

Notes: * Vibro in BP30 PLATFORM is available with his own driver
** I2Sy is available only in DAI configuration

4.1.2 AUD_release_resource

Prototype: SINT8 AUD_release_resource(UINT8 handle);
Parameters: UINT8 handle: The parameter returned by AUD_allocate_resource(…)
Returns: Return code, see chapter 0.
Description: The resource is released, so it can be used by another task. If the resource is not disabled/stopped, this is done automatically. It is checked that the handle and resource 'match', if not an error code is returned.

4.1.3 AUD_init

Prototype: void AUD_init(void)
This function is now automatically called at system power up, and it should not be called from application side.

4.1.4 AUD_get_downlnk_parallel_path_configuration

Prototype: UINT32 AUD_get_downlnk_parallel_path_configuration(aud_downlink_source_enum path)

Parameters: aud_downlink_source_enum path

Returns: Parallel path configuration

Description: This function gets the information about what are the downlink path's, which can be run in parallel with the given downlink 'path'. In the return value, the position of bit '1' (Position numbering will start from '0') indicates the enum value of the downlink path (In aud_downlink_source_enum), which can be run in parallel with 'path'.

Enum: aud_downlink_source_enum	
Enum name	Description
aud_normal_earpiece	Handset speaker
aud_mono_headset	Mono headset speaker
aud_stereo_headset	Stereo headset speakers (not supported on BP30 PLATFORM)
aud_backspeaker	Handsfree speaker / Backspeaker (mono)
aud_I2S1_tx	Bluetooth device
aud_downlink_source_end	Enum end

4.1.5 AUD_get_uplink_parallel_path_configuration

Prototype: UINT32 AUD_get_uplink_parallel_path_configuration(aud_uplink_source_enum path)

Parameters: aud_uplink_source_enum path

Returns: Parallel path configuration

Description: This function gets the information about what are the uplink path's, which can be run in parallel with the given uplink 'path'. In the return value, the position of bit '1' (Position numbering will start from '0') indicates the enum value of the uplink path (In aud_uplink_source_enum), which can be run in parallel with 'path'.

Enum: aud_uplink_source_enum	
Enum name	Description
aud_handset_mic	Handset microphone
aud_headset_mic	Headset microphone
aud_I2S1_rx	Bluetooth device
aud_uplink_source_end	Enum end

4.2 Path Management

The MMI will have 100% freedom to setup any kind of combination for the Uplink and Downlink audio path, and besides this, the MMI have the possibility to select between 1-100 volume steps for any resource and a master volume control with 1-100 steps controlling all.
Due to the limitation in firmware, it has to be guaranteed that always at least one transducer (mic/ speaker / bt) is active, when moving from Idle mode to non-idle mode of operation.
The volume steps for the resources works independently of each other.
Default settings for the different resources and master volume are set – but these values are not optimized for any given platform.

If the MMI make a request to the Audio-driver, that will result in a conflict (due to double usage of HW block, missing support in specific HW etc.) the audio-driver will send an error back to the MMI, and no change of the audio setup will be made.

4.2.1 AUD_add_uplinkpath

Prototype: SINT8 AUD_add_uplinkpath(aud_uplink_source_enum path)
Parameters: aud_uplink_source_enum path
Returns: Return code, see chapter 0.
Description: Beside the control of the resources the application must also set the audio uplink path. With this function, the application can add the new uplink path for the audio. The first added path always has the highest priority. The paths added later on, are considered to have lesser priority when compared to the first selected path. Path's can also run in parallel. This depends upon the path and its hardware blocks and their inter dependencies.
For ex: Say we have 3 path's, 'A', 'B' and 'C' and path 'A' can run in parallel with path 'C'. Then, if path 'A' is already added to the driver, path 'C' can also be added and enabled together.
In general, when a new path is added, it will be checked with all the other path's which are already added to the driver. If any one path in the buffer cannot run in parallel to the new path, the new path will not be added to the target buffer. On the other hand, if it is allowed, the new path will be added.
Currently, the settings programmed for the first selected path are used in the driver. If the first selected path is removed, the next path in the buffer will be set. Now this path, which is now moved to the first position in the buffer, will have the highest priority and so on. The uplink path is selected with the path parameter from the list in the 'aud_uplink_source_enum'.

Enum: aud_uplink_source_enum	
Enum name	Description
aud_handset_mic	Handset microphone
aud_headset_mic	Headset microphone
aud_I2S1_rx	Bluetooth device
aud_uplink_source_end	Enum end

4.2.2 AUD_remove_uplinkpath

Prototype: SINT8 AUD_remove_uplinkpath(aud_uplink_source_enum path)
Parameters: aud_uplink_source_enum path
Returns: Return code, see chapter 0.
Description: With this function the application can remove the source for the audio. The audio sources are selected with the path parameter from the list in the 'aud_uplink_source_enum'. See description in section 4.2.1. Please note that, when the enum value, 'aud_uplink_source_end' is used as the parameter to this function, all the current uplink paths will be deleted.

4.2.3 AUD_add_downlinkpath

Prototype: SINT8 AUD_add_downlinkpath(aud_downlink_source_enum path)
Parameters: aud_downlink_source_enum path
Returns: Return code, see chapter 0.
Description: With this function, the application can add the new downlink path for the audio. The first selected path always has the highest priority. The paths added later on, are considered to have lesser priority when compared to the first selected path. Path's can also run in parallel. This depends upon the path and its hardware blocks and their inter dependencies.
For ex: Say we have 3 path's, 'A', 'B' and 'C' and path 'A' can run in parallel with path 'C'. Then, if path 'A' is already added to the driver, path 'C' can also be added and enabled together.
In general, when a new path is added, it will be checked with all the other path's which are already added to the driver. If any one path in the buffer cannot run in parallel to the new path, the new path will not be added to the target buffer. On the other hand, if it is allowed, the new path will be added.
Currently, the settings programmed for the first selected path are used in the driver.
If the first added path is removed, the next path in the buffer will be set. Now this path, which is now moved to the first position in the buffer, will have the highest priority and so on. The downlink paths are selected with the path parameter from the list in the aud_downlink_source_enum.

Enum: aud_downlink_source_enum	
Enum name	Description
aud_normal_earpiece	Handset speaker
aud_mono_headset	Mono headset speaker
aud_stereo_headset	Stereo headset speakers (NOT supported)
aud_backspeaker	Handsfree speaker / Backspeaker

aud_I2S1_tx	Bluetooth device
aud_downlink_source_end	Enum end

Note: When activating any external_ringer downlink paths, audio_driver will automatically reassign first non-external ringer, active path to primary (first added) path to avoid having external ringer paths influence BaseBand chip audio volumes.

4.2.4 AUD_remove_downlinkpath

Prototype: SINT8 AUD_remove_downlinkpath(aud_downlink_source_enum path)
Parameters: aud_uplink_source_enum path
Returns: Return code, see chapter 0.
Description: With this function the application can remove the destination for the audio. The audio destination are selected with the path parameter from the list in the aud_downlink_source_enum see description in section 4.2.3.
Please note that, when the enum value, 'aud_downlink_source_end' is used as the parameter to this function, all the current downlink paths will be deleted.

4.3 Volume and muting concept

Based on the resource-concept, it is possible to adjust the individual audio resources (MP3, internal midi, tone generator etc.) volume in 100 steps using the AUD_set_resource_volume (….) function.
The volume step for the individual resource will be made so it only affects the selected resource.

AUD_set_master_volume(….) controls the volume for all resources, it provides 100 steps equivalent to the 100 steps for the resource volume control, i.e. that an increase of n steps in Master volume can be "leveled out" by a decrease of n steps for a specific resource – in terms of dB.

The analogy compared to a "well known" scenario, is like a mixer - where each input has its own volume control, and a Master control exists affection all inputs.

Likewise a similar mute concept controlling each resource and a Master Mute function is available. Working after the same philosophy like volume control – with mute of the individual resources using AUD_mute_resource () and all resources using AUD_mute_master ().

Important: The new volume concept is controlled by a compiler define AUD_MASTER_VOLUME_CONCEPT which must be set for the complete audio driver.

It is necessary to set each volume with as high a volume as possible. Just letting the master volume be fixed at a high level and then only using the resource volume to control everything is not recommended.
Instead an approach like having a speech scenario, where speech is the main objective, setting the speech resource volume as high as possible and the using the master volume to regulate it is the best solution. In case an ringer has to be played in this setup, the correct volume setting for the ringer then has to be calculated – and eventually it might be necessary to regulate the different resources.

4.3.1 AUD_set_resource_volume

Prototype: SINT8 AUD_set_resource_volume (UINT8 handle, aud_volume_enum volume)
Parameters: UINT8 handle: The parameter returned by AUD_allocate_resource(…)
Note: Except speech resource.
aud_volume_enum volume
Returns: Return code, see chapter 0.

Description: Change the speaker volume. 100 steps are available, with a 0.25 dB change for each step (0.5dB for each step when handle is for external Yamaha polyringer and sound should be played from backspeaker). The audio driver only stores the volume in RAM. If the volume should be remembered after a power off, it is up to the MMI to store the volume level in the EEPROM. The handle defines which resource volume to be set.

Enum: aud_ volume_enum	
Enum name	Description
aud_volume_1	Volume step one
aud_volume_2	Volume step two
--------------	------------
aud_volume_99	Volume step ninety nine
aud_volume_100	Volume step hundred
aud_volume_special	Special volume step for test
aud_speech_volume_end	Enum end

4.3.2 AUD_set_master_volume

Prototype: SINT8 AUD_set_master_volume(aud_volume_enum volume)
Parameters: aud_volume_enum volume
Returns: Return code, see chapter 0.
Description: Change the volume for all resources (except external Yamaha polyphonic ringer where volume should be changed by AUD_set_resource_volume(..)). 100 steps are available, with a 0.25 dB change for each step. The audio driver only stores the volume in RAM. If the volume should be remembered after a power off, it is up to the MMI to store the volume level in the EEPROM.

Enum: aud_ volume_enum	
Enum name	Description
aud_volume_1	Volume step one
aud_volume_2	Volume step two
--------------	------------
aud_volume_99	Volume step ninety nine
aud_volume_100	Volume step hundred

aud_volume_special	Special volume step for test
aud_speech_volume_end	Enum end

4.3.3 AUD_mute_resource

Prototype: SINT8 AUD_mute_resource (UINT8 handle, aud_mute_enum enable_disable, aud_ul_dl_direction_enum direction)
Parameters: UINT8 handle: The parameter returned by AUD_allocate_resource(…)
aud_mute_enum enable_disable
aud_ul_dl_direction_enum direction
Returns: Return code, see chapter 0.
Description: Controls muting of individual resources. Using “enable” makes the muting of a resource active. Disable muting restores the last volume setting given for the specific resource.
Direction is currently only active for speech and voice memo, but should always be set to a valid value in all cases.

Enum : aud_mute_enum	
Enum name	Description
aud_mute_enable	Mutes the given resource
aud_mute_disable	Restores the last given volume setting for a resource

Enum: aud_direction_enum	
Enum name	Description
aud_direction_uplink,	Mute direction uplink
aud_direction_downlink,	Mute direction downlink
aud_direction_updownlink,	Mute direction uplink and downlink
aud_direction_end	

4.3.4 AUD_mute_master

Prototype: SINT8 AUD_mute_master (aud_mute_enum enable_disable)
Parameters: aud_mute_enum enable_disable
Returns: Return code, see chapter 0.
Description: Controls muting of master volume. Using "enable" makes the muting active. Disable muting restores the last volume setting.

Enum : aud_mute_enum	
Enum name	Description
aud_mute_enable	Mutes all enabled resources.
aud_mute_disable	Restores the last given volume setting.

4.4 Speech

The functions in this section, all relates to the speech resource.

4.4.1 AUD_speech_enable

Prototype: SINT8 AUD_speech_enable(UINT8 handle)
Parameters: UINT8 handle: The parameter returned by AUD_allocate_resource(…)
Returns: Return code, see chapter 0.
Description: Enable uplink and downlink audio path for speech. The volume level will be level last set with AUD_speech_set_volume level.

4.4.2 AUD_speech_disable

Prototype: SINT8 AUD_speech_disable (UINT8 handle)
Parameters: UINT8 handle: The parameter returned by AUD_allocate_resource(…)
Returns: Return code, see chapter 0.
Description: Powers down the audio path, both uplink and downlink.

4.4.3 AUD_set_EC_NR

Prototype: SINT8 AUD_set_EC_NR (UINT8 EC_on,UINT8 NR_on);
Parameters: UINT8 EC_on, UINT8 NR_on (both treated as booleans)

Returns: Return code, see chapter 0.
Description: This function can be used to add a mask to the actual setting for the selected UL path. This means, if the path allows the Echo Cancellation or the Noise Reduction to be turned on, this mask can be used to either turn it on or off. If the path sets EC or NR to off, this mask as no effect, i.e. it can not turn it on.

4.5 DSP tone generator

All the functions in this section deals with the DSP tone generator. The DSP tone generator is used for supervisory tones and info tones (see aud_tone_id_enum).

4.5.1 AUD_tone_start

Prototype: SINT8 AUD_tone_start(UINT8 handle, aud_tone_id_enum tone_id, UINT16 nof_repeats, SINT16 mix_factor)

Parameters: UINT8 handle: The parameter returned by AUD_allocate_resource(…)
aud_tone_id_enum tone_id
UINT16 nof_repeats
SINT16 mix_factor

Return codes see 0 and (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_playback_finish	If the nof_repeats parameter is different from 0, the AUD_DRIVER_RSP signal with this return code will be send when the tone has played to the end. If nof_peats is 0 the tone generator must be stopped by the function AUD_tone_stop(..), this will not result in sending the SDL signal.

Description: Start to play the specified tone (tone_id). Setting nof_repeats to 0, means repeats forever. If a number different from 0 is selected for the nof_repeats, the audio driver sent a message to the calling module that the tone is finish. The calling module must still release the resource the normal way. If a tone is currently playing and a new tone is requested, the present tone will be disabled and the new tone is started, i.e. no suspend/resume functionality. The mix_factor is a kind of volume. Valid range is 0-0x7FFF, where 0 is no sound, and 0x7FFF is max. A value of 0x4000 will result in a volume level equal to speech, whereas a lower value than 0x4000 results in tones being playing played with a lower volume than speech.

Enum: aud_tone_id_enum	
Enum name	Description
aud_tone_DTMF_0	DTMF key 0
aud_tone_DTMF_1	DTMF key 1
aud_tone_DTMF_2	DTMF key 2
aud_tone_DTMF_3	DTMF key 3
aud_tone_DTMF_4	DTMF key 4
aud_tone_DTMF_5	DTMF key 5
aud_tone_DTMF_6	DTMF key 6
aud_tone_DTMF_7	DTMF key 7
aud_tone_DTMF_8	DTMF key 8
aud_tone_DTMF_9	DTMF key 9

aud_tone_DTMF_hash	DTMF hash key
aud_tone_DTMF_asterix	DTMF asterix key
aud_tone_key_tone_1	key tone 1
aud_tone_key_tone_2	key tone 2
aud_tone_key_tone_3	key tone 3
aud_tone_key_tone_4	key tone 4
aud_tone_key_tone_5	key tone 5
aud_tone_sv_subscriber_busy	supervisory tone: subscriber busy
aud_tone_sv_congestion	supervisory tone: congestion
aud_tone_sv_radio_path_ack	supervisory tone: radio path ack
aud_tone_sv_radio_path_not_avail	supervisory tone: radio path not avail
aud_tone_sv_error_info	supervisory tone: error info tone:
aud_tone_sv_call_waiting	supervisory tone: call waiting
aud_tone_info_free_tone	info tone: free tone
aud_tone_info_connection	info tone: connection
aud_tone_info_disconnect	info tone: disconnect
aud_tone_info_device_in	info tone: device in
aud_tone_info_device_out	info tone: device out
aud_tone_info_msg_full	info tone: msg full
aud_tone_info_ussd	info tone: ussd
aud_tone_info_minute_minder	info tone: minute minder
aud_tone_info_error_1	info tone: error 1
aud_tone_info_error_2	info tone: error 2

aud_tone_info_sms_in_call	info tone: sms in call
aud_tone_info_broadcast_in_call	info tone: broadcast in call
aud_tone_info_alarm_in_call	info tone: alarm in call
aud_tone_info_low_bat_in_call	info tone: low battery in call
aud_tone_info_power_off	info tone: power off
aud_tone_info_power_on	info tone: power on
aud_tone_info_single_beep	info tone: single beep
aud_tone_info_positive_acknowledgement	info tone: positive acknowledgement
aud_tone_info_negative_acknowledgement	info tone: negative acknowledgement
aud_tone_info_auto_redial	info tone: auto redial
aud_tone_info_network_attention	info tone: network attention
aud_tone_info_dial_tone	info tone: dial tone
aud_tone_info_low_bat	info tone: low battery
aud_tone_id_end	End of enum

4.5.2 AUD_tone_start_user_tone

Prototype: SINT8 AUD_tone_start_user_tone(UINT8 handle, void *tone_data, aud_tone_type_enum type, UINT32 nof_tones, UINT16 nof_repeats, SINT16 mix_factor);

Parameters: UINT8 handle: The parameter returned by AUD_allocate_resource(…)
void *tone_data
aud_tone_type_enum type
UINT32 nof_tones
UINT16 nof_repeats
SINT16 mix_factor

Description: In principle the same function as AUD_tone_start_user_tone(..). Instead of using a predefined tone, a pointer to a user-defined tone is given. The format of the tone data can be either single tone, dual tone or triple tone, this is determined by the type parameter. Structs are defined in aud_drv.h (aud_single_user_tone_type, aud_dual_user_tone_type and aud_triple_user_tone_type). Below a single tone sequence is showed.

{frequency_1, amplitude_1, duration_1},
:
:
{frequency_N, amplitude_N, duration_N}

Frequency is aUINT16 and valid range is [350 3500]. Duration is a UINT16 and valid range is [10 8000] msec, the resolution is 5 msec. Valid range for amplitude is [0 0x7FFF] (0x4000 is the same as speech).

Enum: aud_tone_type _enum	
Enum name	Description
aud_single_user_tone	Play tone with one frequency
aud_dual_user_tone	Play tone with two frequency's
aud_triple_user_tone	Play tone with tree frequency's

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_playback_finish	If the nof_repeats parameter is different from 0, the AUD_DRIVER_RSP signal with this return code will be send when the tone has played to the end. If nof_peats is 0 the tone generator must be stopped by the function AUD_tone_stop(..), this will not result in sending the SDL signal.

4.5.3 AUD_tone_stop

Prototype: SINT8 AUD_tone_stop(UINT8 handle);
Parameters: UINT8 handle: The parameter returned by AUD_allocate_resource(…)
Returns: Return code, see chapter 0.
Description: Immediately stops the tone generator.

4.5.4 AUD_key_tone

Key tones are handled special, because it is not necessary to allocate a resource prior to playing the key tone, even though the DSP tone generator is also used to play the key tones. This is because just a small delay will be noticed by the user, so a very fast response is required, also the key tone must always be played, no matter what else is going. This is considered a problem since the key tone is only active for short period this is no problem.

Prototype: SINT8 AUD_key_tone(aud_tone_id_enum key_tone, SINT16 mix_factor);
Description: Plays the select key beep, using the DSP tone generator. If the DSP tone generator is already in use e.g. by playing a supervisory tone, this tone will be suspended until the key tone has been played. After the key tone is finish the originally tone will be resumed. For the mix_factor parameter, see the description in the AUD_tone_start function.
Parameters: aud_tone_id_enum id (see table under AUD_tone_start)
SINT16 mix_factor
Returns: Return code, see chapter 0.
Description: Plays the select key beep, using the DSP tone generator. If the DSP tone generator is already in use e.g. by playing a supervisory tone, this tone will be suspended until the key tone has been played. After the key tone is finish the originally tone will be resumed. For the mix_factor parameter, see the description in the AUD_tone_start function.

4.6 Voice memo

The voice memo is used for recording and playback of voice. The voice memo functionality can be used in both idle- and Tch26 mode. In Tch26 mode, it is possible to record and playback both in uplink and downlink direction.

4.6.1 AUD_vm_start_recording

Prototype: SINT8 AUD_vm_start_recording (UINT8 handle, aud_vm_mode_enum vm_mode, aud_media_enum media_type, aud_dsp_format_enum format, UINT8 rate, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size, UINT32 offset)

Description: This function is used to setup and start voice recording. When recording to FFS the file should be opened in streaming mode (size must be 'streaming'). If the recorded message is going to be send as an audio file in a MMS message (format should be AMR), necessary header information must be added by the application. The audio driver only stores the sampled data.

Note: When PCM codec is used for recording the speech data by allocating the resource, 'aud_resource_record_vm', will block the allocation of the resource, 'aud_resource_record_pcm'. Meaning, when voice memo interface is used for recording the data in PCM format, usage of the PCM interface for the same is not allowed.

Parameters:

Table 1 UINT8 handle: The parameter returned by AUD_allocate_resource(…)

- aud_vm_mode_enum vm_mode: used to tell whether the recording shall start in idle (standby) mode or tch26 (ongoing conversation) mode, see table below:

Enum: aud_vm_mode_enum	
Enum name	Description
aud_vm_mode_standby	The voice memo shall be recorded in the Stand-by (idle) mode
aud_vm_mode_tch	The voice memo shall be recorded in the Tch26 mode (there is an ongoing conversation). Both uplink and downlink will be recorded

- UINT16 media_type: The recorded sound data can be saved in a RAM buffer or directly to FFS. This parameter tells which media to use. If FFS is selected, it is the clients responsibility to make sure that a file is created and opened in streaming mode, so it is ready for use.

Enum: aud_media_enum	
Enum name	Description
aud_media_ffs	Save in the Flash File System
aud_media_mmc	Save in MultiMediaCard; Not supported in BP30 platform.
aud_media_ram	Save in RAM
aud_media_test_ram	For test only
aud_media_I2S	Save through I2S; Not supported in BP30 platform
aud_media_mmf	Save by Multi Media Framework ;Not supported in BP30 platform

aud_media_mmf_test	For test only ;Not supported in BP30
aud_media_vr	Voice Recognition; Not supported in BP30 platform
aud_media_end	End of enum

- aud_dsp_format_enum format: different speech codec's can be used for compressing the speech data. Currently 3 codec's are available, see table.

Enum: aud_dsp_format_enum	
Enum name	Description
aud_dsp_format_amr	The AMR codec shall be used for compression. This is format used in MMS messages. If header information is needed for MMS this must be added by the application. Selecting AMR as format also implies that a sampling must be chosen, see the 'rate' parameter
aud_dsp_format_full_rate	The full rate codec shall be used for compression.
aud_dsp_format_pcm	The PCM codec shall be used for compression.

Note: When using the PCM codec (aud_dsp_format_pcm), the recorded PCM data format Would be fixed to,

1. 8Khz sample rate
2. 16 bit Signed data in 2's complement.

- UINT8 rate: This is the sampling rate if AMR is chosen as format. If full rate is selected as format this parameter is don't care (not used), for full rate the memory consumption is constant (34 bytes pr. Speech frame). The rate parameter must be between 0-7. The sampling rate and memory consumption can be seen in table below. If format is 'full rate' this parameters is don't care, i.e. not used.

Rate	Sampling rate [kBit/sec]	Bytes pr. Speech frame	Bytes pr. Sec.	Bytes pr. Minute
0	4,75	13	650	38,1 KB
1	5,15	14	700	41,0 KB
2	5,90	16	800	46,9 KB
3	6,70	18	900	52,7 KB
4	7,40	20	1000	58,6 KB
5	7,95	21	1050	61,5 KB
6	10,20	27	1350	79,1 KB
7	12,20	32	1600	93,8 KB
Full rate	Full rate	34	1700	99,6 KB

- UINT16* file_handle: If FFS is selected as media this parameter is the file handle returned from the creating/opening process of the FFS file. If RAM is selected as media this parameter is the pointer to the RAM-buffer.
- UINT32 buffer_size: If RAM is selected as the media, this parameter tells the size of the RAM-buffer. This is to prevent the recording function from overwriting RAM cells. The recording will automatically stop when no more space is available in the buffer and the AUD_DRIVER_RSP signal will be sent to

the client, where parm1 will tell the exact number of bytes used. The vm state machine will go its idle state, but the resource is still allocated. If media is FFS this parameter is don't care, i.e. it is not used and can be set to any value.
- start_offset: If RAM or FFS is selected as media this parameter is the number of bytes which is left blank in the start of the buffer. This can be used if some header info should be present in the start of the data.

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_storage_problems	For some reason the FFS has reported an error. This problem can be caused by application, FFS or audio driver. The error code from FFS can be seen in parm1 of the AUD_DRIVER_RSP signal. . The voice memo state machine will return to the idle state, but the resource is still allocated.
aud_rc_performace_problems	Data are delivered either to slow or to fast to the audio driver. This problem will only occur if media is FFS. The problem can be both in FFS and/or in the audio driver.
aud_rc_ram_buffer_used	All the space in the RAM-buffer has been used, and the recording has stopped. Parm1 in the AUD_DRIVER_RSP signal will contain the exact number of bytes used. The voice memo state machine will return to the idle state, but the resource is still allocated. This situation of course only occurs if the media is RAM
aud_rc_vm_missing_dsp_resources	

4.6.2 AUD_vm_stop_recording

Prototype: SINT8 AUD_vm_stop_recording(UINT8 handle)

Parameters:
- handle: : The parameter returned by AUD_allocate_resource(…)

Description: This function stops voice-memo recording process.

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_recording_finish	If media is FFS, this signal is send when the recorder has finished writing to the FFS. When this signal is received by the application, the application can safely close the FFS file. This RC should be merged with

	aud_rc_vm_bytes_used_in_ram_buffer

4.6.3 AUD_vm_start_playback

Prototype: SINT8 AUD_vm_start_playback(UINT8 handle, AUD_vm_mode_enum vm_mode, aud_media_enum media_type, aud_dsp_format_enum format, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size, UINT16 nof_repeats, UINT32 offset)

Description: This function is used to start playback of a voice memo. If the voice memo is received in a MMS message all header info etc. must be removed before playback, only the raw sampled data should be 'send' to this function. If data is stored in FFS, the file must be opened in streaming mode.
Note: When using the PCM codec (aud_dsp_format_pcm) for playback of PCM data, the PCM data format should be,

1. 8Khz sample rate
2. 16 bit Signed data in 2's complement.

Parameters:

- handle: : The parameter returned by AUD_allocate_resource(…)
- mode: see description in AUD_vm_start_recording(..).
- media: see description in AUD_vm_start_recording(..).
- format: see description in AUD_vm_start_recording(..).
- file_handle: see description in AUD_vm_start_recording(..).
- buffer_size: If RAM is selected as media this parameter tells the number of bytes which should be played back. If media is FFS this parameter is don't care, i.e. it can be set to any value.
- nof_repeats: If RAM or FFS is selected as media this parameter is the number of times the sound data is played, e.g. to play it once the parameter should be 1. If it is 0 the sound data will be repeated infinitely, i.e. a stop request must be sent to the driver before playback stops. For the streaming case this parameter is don't care, i.e. the application must control the number of repeats by itself.
- start_offset: If RAM or FFS is selected as media this parameter is the number of bytes which should be skipped in the start. This can be used if some header info is present in the start of the data.

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_storage_problems	For some reason the FFS has reported an error. This problem can be caused by application, FFS or audio driver. The error code from FFS can be seen in parm1 of the AUD_DRIVER_RSP signal. . The voice memo state machine will return to the idle state, but the resource is still allocated.
aud_rc_playback_started	The voice memo playback has started.
aud_rc_performance_problems	Data are delivered either to slow or to fast to the audio driver. This problem will only occur if media is FFS. The problem can be both in FFS and/or in the audio driver.
aud_rc_playback_finish	The voice memo has played to the end. The voice memo state machine has returned to idle the idle state, but the resource is still allocated
aud_rc_playback_loop	If the VM data is being played for N times,this RC signal is received N-1 times

4.6.4 AUD_vm_stop_playback

Prototype: SINT8 AUD_vm_stop_playback(UINT8 handle)

Parameters: UINT8 handle

Returns (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_playback_stopped	The voice memo has been stopped by the application.

Description: This function is used to switch voice-memo playback off. This function doesn't send SDL signal.

4.6.5 AUD_vm_suspend

Prototype: SINT8 AUD_vm_suspend(UINT8 handle, UINT8 slot_id)
Parameters:

- handle: : The parameter returned by AUD_allocate_resource(…)
- slot_id : The slot in which the driver should store the data needed for resuming playback. 5 slots are available: 0-4.

Returns: Return code, see chapter 0.
Description: This function suspends the playback of voice memo. It is possible to have up to 5 playbacks suspended at the same time. The parameter slot_id is used to tell the driver, to which "slot" it should store the data needed to resume the playback. The audio driver doesn't keep track of the slot used, i.e. if a request for suspending to a slot that already is in use, the audio driver simply overwrites the info in that slot, without any warning to the client. When playback has been suspended, a SDL signal is sent to the client.
Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_playback_suspended	The playback has been suspended
aud_rc_request_error	If suspend is requested when there is no playing data

4.6.6 AUD_vm_resume

Prototype: SINT8 AUD_vm_resume(UINT8 handle, UINT8 slot_id)
Parameters:

- handle: : The parameter returned by AUD_allocate_resource(…)
- slot_id The slot in which the driver should store the data needed for resuming playback. 5 slots are available: 0-4.

Returns: Return code, see chapter 0.

Description: This function resumes playback for the selected slot. If the selected slot doesn't contain any data, a return code is sent to the client. When playback starts after the resume, a SDL signal is sent to the client

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_suspend_resume_error	The selected slot did not contain any suspend data
aud_rc_playback_started	When playback resumes, this signal is sent.

4.6.7 AUD_vm_get_total_playtime

Prototype: SINT8 AUD_vm_get_total_playtime(UINT8 handle, aud_media_enum media_type, aud_dsp_format_enum format, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size, UINT32 offset)

Parameters:

- handle: : The parameter returned by AUD_allocate_resource(…)
- media: see description in AUD_vm_start_recording(..).
- format: see description in AUD_vm_start_recording(..).
- file_handle: see description in AUD_vm_start_recording(..).
- buffer_size: see description in AUD_vm_start_recording.
- offset : see descption in AUD_vm_start_playback()

Returns: None
Description: This function returns the total playtime for the selected data. It is possible to request the playtime both in idle and while playback is ongoing. If the playtime is requested while playback is ongoing, it will always be the playtime for the melody being played which is returned, no matter what the parameters are (i.e. parameters are ignored). The playtime is returned in msec in the AUD_DRIVER_RSP signal in parm1. This function does of course not work in the streaming case. It is not possible to ask for the total playtime on suspended melodies.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_total_playtime	Parm1 contains the total playtime in msec. If parm1 has the value 0xFFFFFFFF, it means that the playtime for some reason could not be calculated.
aud_rc_request_error	If media is streaming, this code is returned

4.6.8 AUD_vm_get_play_position

Prototype: SINT8 AUD_vm_get_play_position(UINT8 handle)

Parameters:
handle: The parameter returned by AUD_allocate_resource()
Returns: None

Description: This function returns the play position of the current playing Voice memo data. The current play position is communicated to the application through the SDL function as first parameter (parm1). The information is passed in 'milliseconds' to the application. The function works only on the data that are currently being played, i.e. it is not possible to get the elapsed time on suspended data. It is not possible to use the function in the streaming case. If used when playback is inactive, an error code is returned in a SDL signal.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_elapsed_time	The actual play position is returned in parm1.
aud_rc_request_error	If playback is done in streaming mode (media is MMF), this code is returned or if there is no playing data

4.6.9 AUD_vm_set_play_position

Prototype: SINT8 AUD_vm_set_play_position(UINT8 handle, UINT32 pos, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size)

Parameters:

- handle: : The parameter returned by AUD_allocate_resource(…)
- pos: The parameter gives the information of the postion to be set. The input is in milliseconds
- file_handle: Currently not used.
- buffer_size: Currently not used.
-

Returns: **Return code, see chapter 0.**

Description: This function sets the play position of the Voice memo playback. The play position is in milliseconds. The play position is referring to position from the start of the data. The function can be used both before playback is started and after playback is started. The function works only on the melody that is currently being played, i.e. it is not possible to set the play position on suspended melodies. It is not possible to use the function in the streaming case.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_unknown_position	The requested play position is not possible.
aud_rc_request_error	If playback is done in streaming mode (media is MMF), this code is returned.
aud_rc_parameter_out_of_range	If the set position is beyond the file size, this RC signal is sent.

4.7 PCM player

In this chapter, there is a detailed description of the PCM functions used for playback and record of PCM data.

4.7.1 AUD_pcm_intern_start_playback

Prototype: SINT8 AUD_pcm_intern_start_playback (UINT8 handle, aud_pcm_mode_enum mode, aud_pcm_sample_rate_enum sample_rate, UINT8 bit_rate, aud_media_enum media_type, aud_pcm_format_enum format, UINT16 DWD_HUGE *file_handle, , UINT32 buffer_size, UINT16 nof_repeats, UINT32 start_offset);

Description: This function is used to start playback of PCM data.

Parameters:

- handle: The parameter returned by AUD_allocate_resource(…).
- Mode: Tells whether the data are stereo, mono or dual mono. See aud_pcm_mode_enum table:

Enum: aud_pcm_mode_enum	
Enum name	Description
aud_pcm_mode_stereo	PCM data are in stereo
aud_pcm_mode_mono	PCM data are in mono
aud_pcm_mode_dual_mono	PCM data are in mono, but they should be played in both left and right channel

- Sample_rate: The sample rate for PCM data, see aud_pcm_sample_rate_enum for the available sample rates:

Enum: aud_pcm_sample_rate_enum	
Enum name	Description
aud_pcm_sample_rate_8khz	8 kHz
aud_pcm_sample_rate_11khz	11.025 kHz
aud_pcm_sample_rate_12khz	12 kHz
aud_pcm_sample_rate_16khz	16 kHz
aud_pcm_sample_rate_22khz	22.05 kHz
aud_pcm_sample_rate_24khz	24 kHz
aud_pcm_sample_rate_32khz	32 kHz
aud_pcm_sample_rate_44khz	44.1 kHz
aud_pcm_sample_rate_48khz	48 kHz

- Bit_rate: The number of bits per sample, max. is 16 bit. For ADPCM this parameter is don't care.
- media_type: The sound data can be played from a RAM buffer, directly from FFS or via a streaming interface. This parameter tells which media to use. If FFS is selected, it is the clients responsibility to make sure that a file is created and opened in streaming mode, so it is ready for use.

Enum: aud_media_enum	
Enum name	Description
aud_media_ffs	The data to be played are stored in FFS.

aud_media_mmc	The data are located on a MMC card. Not supported on BP30
aud_media_ram	The data are located in a RAM buffer
aud_media_mmf	Streaming interface; Not supported on BP30 platform

- Format: The format of the data: either PCM, ADPCM or WAVE, see enum:

Enum: aud_pcm_format_enum	
Enum name	Description
aud_pcm_format_pcm	PCM data
aud_pcm_format_adpcm	ADPCM data. The codec is the Intel DVI. NOT SUPPORTED YET.
aud_pcm_format_wave	Wave file. Off course only the sample rates, bit rates etc used for PCM/ADPCM is supported

- *file_handle: If FFS is selected as media this parameter is the file handle returned from the creating/opening process of the FFS file. If RAM is selected as media, this parameter is the pointer to the RAM-buffer. If media is MMF, this parameter is not used
- buffer_size: If RAM is selected as media this parameter tells the number of bytes which should be played back. If media is FFS or MMF this parameter is don't care, i.e. it can be set to any value.
- nof_repeats: If RAM or FFS is selected as media this parameter is the number of times the sound data is played, e.g. to play it once the parameter should be 1. If it is 0 the sound data will be repeated infinitely, i.e. a stop request must be sent to the driver before playback stops. For the streaming case this parameter is don't care, i.e. the application must control the number of repeats by itself.
- start_offset: If RAM or FFS is selected as media this parameter is the number of bytes which should be skipped in the start. This can be used if some header info is present in the start of the data.

Returns: Return codes, see 0..

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_storage_problems	For some reason the storage system has reported an error. This problem can be caused by application, storage system or audio driver. The error code from storage system can be seen in parm1 of the AUD_DRIVER_RSP signal. The PCM state machine will return to the idle state, but the resource is still allocated.
aud_rc_performance_problems	Data are not delivered properly to the audio driver. This problem will only occur if media is FFS or MMC. The problem can be both in FFS and/or in the audio driver.
aud_rc_playback_finish	The PCM has played to the end. The PCM state

	machine has returned to idle the idle state, but the resource is still allocated
aud_rc_unknown_position	If an invalid offset, is set with the AUD_pcm_intern_set_play_position, before the playback is started, this return code is sent. After the code is sent the playback will start from position 0 (start of file/data)
aud_rc_syntax_error	If an error is detected in wave header this return code is sent. Playback is not started.
aud_rc_playback_started	This code is sent when playback starts.

4.7.2 AUD_pcm_intern_stop_playback

Prototype: SINT8 AUD_pcm_intern_stop_playback (UINT8 handle)

Description: This function is used to switch PCM playback off.

Parameters: handle: The parameter returned by AUD_allocate_resource(…).

Returns: Return code, see chapter 0.

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_playback_stopped	This code will be send when audio driver has stopped the playback. **Note:** Parameter 4 will be 1 if the playback was stopped by driver instead of application

4.7.3 AUD_pcm_intern_start_recording

Prototype: SINT8 AUD_pcm_intern_start_recording(UINT8 handle, aud_pcm_sample_rate_enum sample_rate, aud_media_enum media_type, aud_pcm_format_enum format, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size, UINT32 start_offset)

Description: This function is used to start the recording process in PCM/ADPCM mode. The data delivered are the raw samples from the DSP.

Note: This function will invoke 'AUD_vm_start_recording' internally and hence, when a resource, 'aud_resource_record_pcm' is allocated, it is not possible to allocate the resource, 'aud_resource_record_vm'.

Parameters:

- handle: The parameter returned by AUD_allocate_resource(…).
- Sample_rate: See description of aud_pcm_sample_rate_enum in AUD_pcm_internal_start_playback. Currently only aud_pcm_sample_rate_1 (8 kHz) and aud_pcm_sample_rate_4 (16 kHz) are possible.
- Media_type: See description in AUD_pcm_intern_start_recording(…).
- Format: See description of aud_pcm_format_enum in AUD_pcm_internal_start_recording(..).
- file_handle: If FFS is selected as media, this parameter is the file handle returned from the creating/opening process of the FFS file. If RAM is selected as media, this parameter is the pointer to the RAM-buffer. If MMF is selected as media this parameter is don't care.
- buffer_size: If RAM is selected as the media, this parameter tells the size of the RAM-buffer. This is to prevent the recording function from overwriting RAM cells. The recording will automatically stop when

no more space is available in the buffer and the AUD_DRIVER_RSP signal will be sent to the client, where parm1 will tell the exact number of bytes used. The PCM player state machine will go its idle state, but the resource is still allocated. If media is FFS or MMS this parameter is don't care, i.e. it is not used and can be set to any value.

- start_offset: If RAM or FFS is selected as media this parameter is the number of bytes which is left blank in the start of the buffer. This can be used if some header info should be present in the start of the data.

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_storage_problems	For some reason the storage system has reported an error. This problem can be caused by application, storage system or audio driver. The error code from storage system can be seen in parm1 of the AUD_DRIVER_RSP signal. The PCM state machine will return to the idle state, but the resource is still allocated.
aud_rc_performance_problems	Data are not delivered properly to the audio driver. This problem will only occur if the storage system is used as media (e.g. FFS, MMC). The problem can be both in the storage system and/or in the audio driver.
aud_rc_ram_buffer_used	If media is RAM, this return code is sent if all bytes in the RAM buffer have been used. The exact number of bytes used in the RAM buffer is given in parm1.
aud_rc_recording_started	This code is sent when the recording starts

4.7.4 AUD_pcm_intern_stop_recording

Prototype: SINT8 AUD_pcm_intern_stop_recording (UINT8 handle)

Description: This function stops the PCM recording process.
Note: This function is internally mapped to 'AUD_vm_stop_recording'

Parameters: handle: The parameter returned by AUD_allocate_resource(…).

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_recording_finish	If media is RAM, this signal is AUD_DRIVER_RSP signal is sent to tell the number bytes used in RAM-buffer. The exact number of bytes used in the RAM buffer is given in parm1.

	If the storage system is used as media, this signal is send when the recorder has finished. When this signal is received by the application, the application can safely close the file.

4.7.5 AUD_pcm_intern_suspend

Prototype: SINT8 AUD_pcm_intern_suspend (UINT8 handle, UINT8 slot_id);

Description: This function suspends the playback of the PCM data. The recording process cannot be suspended; only playback can be suspended. It is possible to have up to 5 melodies suspended at the same time. The parameter slot_id is used to tell the driver, to which “slot” it should store the data needed to resume the playback. The audio driver doesn’t keep track of the slot used, i.e. if a request for suspending to a slot that already is in use, the audio driver simply overwrites the info in that slot, without any warning to the client. When playback has been suspended a SDL signal is sent to the client. If playback is in streaming mode (media = MMF), the application must not use/change the buffer from which playback is currently being played.

Parameters:
- handle: The parameter returned by AUD_allocate_resource(…).
- slot_id: The slot in which the driver should store the suspend data. 5 slots are available: 0-4.

Returns: Return codes, see 0.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_playback_suspended	The playback has been suspended
aud_rc_no_playback	There is currently no playback, which make the command obsolete

4.7.6 AUD_pcm_intern_resume

Prototype: SINT8 AUD_pcm_intern_resume (UINT8 handle, UINT8 slot_id);

Description: This function resumes playback for the selected slot. If the selected slot doesn’t contain any data, a return signal is sent to the client. When playback starts after the resume, a SDL signal is sent to the client.

Parameters:
- handle: The parameter returned by AUD_allocate_resource(…).
- slot_id: The slot in which the suspended data are stored. 5 slots are available.

Returns: Return codes, see 0..

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_suspend_resume_error	The selected slot did not contain any suspend data
aud_rc_playback_started	When playback starts, this signal is sent.

4.7.7 AUD_pcm_intern_stop_suspend

Prototype: SINT8 AUD_pcm_intern_stop_suspend(UINT8 handle, UINT8 slot_id)

Parameters:
handle: The parameter returned by AUD_allocate_resource()
slot_id : This indicates which slot id has to be suspended. The slot ids upto 5 instances is supported. This slot id ranges from 0 to 4 for suspended data

Returns: Return code, see chapter 0.

Description: : This function clears all the suspended data contents for the specified slot_id so that the slot_id can be used for another instance.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_playback_stopped	The suspended data with slot id has been cleared

4.7.8 AUD_pcm_intern_get_total_playtime

Prototype: SINT8 AUD_pcm_intern_get_total_playtime UINT8 handle, aud_media_enum media_type, aud_pcm_format_enum format, aud_pcm_sample_rate_enum sample_rate, ud_pcm_mode_enum mode, UINT8 bit_rate, UINT16 DWD_HUGE *file_handle, UINT16 buffer_size, UINT32 offset);

Description: This function returns the total playtime for the selected data. It is possible to request the playtime both in idle and while playback is ongoing. If the playtime is requested while playback is ongoing, it will always be the playtime for the melody being played which is returned, no matter what the parameters are (i.e. parameters are ignored). The playtime is returned in msec in the AUD_DRIVER_RSP signal in parm1. This function does of course not work in the streaming case. It is not possible to ask for the total playtime on suspended melodies.

Parameters:

- handle: The parameter returned by AUD_allocate_resource(…).
- Media_type: See description in AUD_pcm_internal_start_playback(…). Aud_media_mmf is not valid.
- Format: See description in AUD_pcm_intern_start_playback(…).
- Sample_rate: See description in AUD_pcm_intern_start_playback(…)
- mode:
- bit_rate:
- file_handle: See description in AUD_pcm_intern_start_playback(…)
- buffer_size: See description in AUD_pcm_intern_start_playback(…)
- offset : See description in AUD_pcm _intern_start_playback(…)

Returns: Return codes, see 0.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_total_playtime	Parm1 contains the total playtime in msec. If parm1 has the value 0xFFFFFFFF, it means that the playtime for some reason could not be calculated.

aud_rc_request_error	If media is streaming, this code is returned

4.7.9 AUD_pcm_intern_get_play_position

Prototype: SINT8 AUD_pcm_intern_get_play_position (UINT8 handle);

Description: This function returns the elapsed time (play position). The play position is returned in msec in the AUD_DRIVER_RSP signal in parm1. The function works only on the melody that is currently being played, i.e. it is not possible to get the elapsed time on suspended melodies. It is not possible to use the function in the streaming case. If used when playback is inactive, an error code is returned in a SDL signal.

Parameters:

- handle: The parameter returned by AUD_allocate_resource(…).

Returns: Return codes, see 0.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_elapsed_time	The actual play position is returned in parm1.
aud_rc_no_playback	If playback isn't active, this code is returned
aud_rc_request_error	If playback is done in streaming mode (media is MMF), this code is returned

4.7.10 AUD_pcm_intern_set_play_position

Prototype: SINT8 AUD_pcm_intern_set_play_position (UINT8 handle, UINT32 pos, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size);

Description: With the function it is possible to change the position from which playback should start. The function can be used both before playback is started and after playback is started. The playback position must be given in msec. The function works only on the melody that is currently being played, i.e. it is not possible to set the play position on suspended melodies. It is not possible to use the function in the streaming case.

Parameters:

- handle: The parameter returned by AUD_allocate_resource(…).
- pos: The position in msec from which playback should start.
- File_handle: Currently not used.
- Buffer_size: Currently not used.

Returns: Return codes, see chapter 0 .

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment

aud_rc_unknown_position	The requested play position is not possible.
aud_rc_request_error	If playback is done in streaming mode (media is MMF), this code is returned.

4.8 FM radio

The interface to the FM radio is part of the audio driver. Since the audio is routed directly from FM radio hardware to the relevant output transducer(s), this interface is only concerned with controlling the radio. As such, it is not directly involved in audio management, but is nevertheless included in the audio driver, because the FM radio is managed as an audio resource. For instance, it is covered by the standard functions for setting resource volume.
Note that since the audio is not routed through the baseband chip, the master volume setting will not affect the FM radio volume.

A separate document describes the FM radio interface [5].

4.9 MP3 player

MP3 is the short form of MPEG –1 Layer 3 Audio coding standard. MPEG stands for Moving Pictures Experts Group. The basic MP3 functionalities can be supported in Idle and Tch26 mode.

4.9.1 AUD_mp3_start

Prototype: SINT8 AUD_mp3_start (UINT8 handle, aud_media_enum media_type, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size, UINT32 id_offset, UINT32 start_frame, UINT16 nof_repeats)
Parameters:
handle: The parameter returned by AUD_allocate_resource()
media_type: The recorded MP3 data has to be saved in a RAM buffer or directly to FFS. This parameter conveys the information which media has to be used to read the MP3 encoded input data. If FFS is selected, it is the client's responsibility to make sure that the MP3 data is stored in FFS file, which is created and opened in streaming mode, so that it is ready for use.

Enum: aud_media_enum	
Enum name	Description
aud_media_ffs	Play a file in FFS.
aud_media_mmf	Not Supported in BP30 platform
aud_media_mmf_test	Not Supported in BP30 platform
aud_media_ram	Play from a RAM buffer (test)

* file_handle: If FFS is selected as media type, this parameter is the handle returned from creating / opening process of FFS file. If RAM is selected as media this parameter is the pointer to t he RAM buffer.
buffer_size: If RAM is selected as the media, this parameter tells the size of the RAM buffer. If media is FFS, this parameter is not used.
id_offset: This gives the information about ID3V2 offset. If the id_offset is 0, the input is a raw data. If id_offset is a 'X' bytes, the input is an MP3 file with header information. The id_offset gives the information about where the MP3 frame starts.

start_frame: This parameter gives information about whether the MP3 should start playing from the start position or it has to be forwarded in idle mode. If start_frame = 0, MP3 playback start from the beginning of the data. If start_frame = X, the MP3 data is forwarded by X frames and then start playback of MP3 data from X frames.
nof_repeats: If RAM or FFS is selected as media this parameter is the number of times the sound data is played, e.g. to play it once the parameter should be 1. If it is 0 the sound data will be repeated infinitely, i.e. a stop request must be sent to the driver before playback stops. For the streaming case this parameter is don't care, i.e. the application must control the number of repeats by itself.

Return codes (sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_storage_problems	For some reason the FFS has reported an error. This problem can be caused by application, FFS or audio driver. The error code from FFS can be seen in parm1 of the AUD_DRIVER_RSP signal. . The voice memo state machine will return to the idle state, but the resource is still allocated.
aud_rc_playback_started	The voice memo playback has started.
aud_rc_performance_problems	Data are delivered either to slow or to fast to the audio driver. This problem will only occur if media is FFS. The problem can be both in FFS and/or in the audio driver.
aud_rc_playback_finish	The voice memo has played to the end. The voice memo state machine has returned to idle the idle state, but the resource is still allocated
aud_rc_playback_loop	If the VM data is being played for N times,this RC signal is received N-1 times

Description: This function is used to start MP3 data to be played. If the MP3 encoded data is stored in the FFS media, the file must be opened in the streaming mode.

4.9.2 AUD_mp3_stop

Prototype: SINT8 AUD_mp3_stop(UINT8 handle)
Parameters: handle : The parameter returned by AUD_allocate_resource()
Description: This function is used to stop MP3 Player operations.
Returns(sent in AUD_DRIVER_RSP signal)

Return code	Comment
aud_rc_playback_stopped	The voice memo has been stopped by the application.

4.9.3 AUD_mp3_suspend

Prototype: SINT8 AUD_mp3_suspend(UINT8 handle, UINT16 slot_id)
Parameters:
UINT8 handle : The parameter returned by AUD_allocate_resource()
slot_id : This indicates which slot id has to be suspended. The slot ids upto 5 instances is supported.

Description: This interface functions suspends the MP3 player operation with the given slot_id.
Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_playback_stopped	The voice memo has been stopped by the application.

4.9.4 AUD_mp3_resume

Prototype: SINT8 AUD_mp3_resume(UINT8 handle, UINT8 slot_id)
Parameters:
handle : The parameter returned by AUD_allocate_resource()
UINT8 slot_id : This indicates which slot id has to be suspended. The slot ids upto 5 instances is supported
Description: This interface function resumes the MP3 play of the slot_id, which has been suspended.
Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_suspend_resume_error	The selected slot did not contain any suspend data
aud_rc_playback_started	When playback resumes, this signal is sent.

4.9.5 AUD_mp3_fastforward

Prototype: SINT8 AUD_mp3_fastforward (UINT8 handle, UINT32 frame, UINT16 slot_id)

Parameters:
handle: The parameter returned by AUD_allocate_resource()
frame : Number of frames of MP3 data to be moved forwarded.
slot_id: This can have between 0-4 or " 999" . If value is between 0-4, it is assumed that the suspended MP3 data is being forwarded and then played. If the valuei s "999", the current playing data is forwarded
Returns: Return code, see chapter 0.
Description: This function is used to forward the MP3 data to be played. If the forward frame number is less than 0 or is greater than the file size, it is communicated to the Application using SDL communication. The description of the SDL communication is specified in Error message table. If the slot_id is equal to 999, the audio driver forwards the current playing MP3 data. Fastforward is supported in the Suspended mode. The application needs to mention the slot_id to resume the operation.

4.9.6 AUD_mp3_backward

Prototype: SINT8 AUD_mp3_backward (UINT8 handle, UINT32 frame, UINT16 slot_id)
Parameters:
handle: The parameter returned by AUD_allocate_resource()
frame : Number of frames of MP3 data to be moved forwarded.
Returns: Return code see 0.
Description: This function is used to support MP3 backward functionality. The data to be played can be moved backward and the user can select the play the data from that frame. If the slot_id == 999, the audio driver forwards the current playing MP3 data. Backward function is supported in the Suspended mode. The application needs to mention the slot_id to resume the operation.

4.9.7 AUD_mp3_get_total_playtime

Prototype: SINT8 AUD_mp3_get_total_playtime (UINT8 handle, aud_media_enum media_type, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size);

Parameters:
handle: The parameter returned by AUD_allocate_resource()
media_type: The recorded MP3 data has to be saved in a RAM buffer or directly to FFS. This parameter conveys the information which media has to be used to read the MP3 encoded input data. If FFS is selected, it is the client's responsibility to make sure that the MP3 data is stored in FFS file, which is created and opened in streaming mode, so that it is ready for use.
* file_handle: If FFS is selected as media type, this parameter is the handle returned from creating / opening process of FFS file. If RAM is selected as media this parameter is the pointer to t he RAM buffer.
buffer_size: If RAM is selected as the media, this parameter tells the size of the RAM buffer. If media is FFS, this parameter is not used.
Description: This function returns the total playtime in Milliseconds. This function is called only in idle mode. If this is called in playback mode, an error message is communicated to the application using SDL communication.
Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_total_playtime	Parm1 contains the total playtime in msec. If parm1 has the value 0xFFFFFFFF, it means that the playtime for some reason could not be calculated.
aud_rc_request_error	If media is streaming, this code is returned

4.9.8 AUD_mp3_get_play_position

Prototype: SINT8 AUD_mp3_get_play_position (UINT8 handle)

Parameters:
-handle: The parameter returned by AUD_allocate_resource()
Description: This function returns the play position of the current playing MP3 data. The current play position is communicated to the application through the SDL function as first parameter. This function is called only in Playback mode. The information is passed in 'milliseconds' to the application.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_elapsed_time	The actual play position is returned in parm1.
aud_rc_no_playback	If playback isn't active, this code is returned
aud_rc_request_error	If playback is done in streaming mode (media is MMF), this code is returned

4.9.9 AUD_mp3_set_play_position

Prototype: SINT8 AUD_mp3_set_play_position (UINT8 handle, UINT32 pos, UINT16 DWD_HUGE *file_handle, UINT32 buffer_size)

Parameters:
handle: The parameter returned by AUD_allocate_resource()
pos: The parameter gives the information about the MP3 start position. This is in milliseconds.

file_handle: If FFS is selected as media type, this parameter is the handle returned from creating / opening process of FFS file. If RAM is selected as media this parameter is the pointer to t he RAM buffer.
buffer_size: If RAM is selected as the media, this parameter tells the size of the RAM buffer. If media is FFS, this parameter is not used.

Description: This function sets the play position of the MP3 data to be played. The play position is in milliseconds. The play position is referring to position from the start of the data. If the play position is already played, the MP3 data is continued playing. In this situation, an error message is sent through SDL communication. The MP3 data is shifted to the play position and playback is resumed from that position.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_unknown_position	The requested play position is not possible.
aud_rc_request_error	If playback is done in streaming mode (media is MMF), this code is returned.
aud_rc_parameter_out_of_range	If the set position is beyond the file size, this RC signal is sent.

4.9.10 AUD_mp3_stop_suspend

Prototype: SINT8 AUD_mp3_stop_suspend(UINT8 handle, UINT8 slot_id)

Parameters:
handle: The parameter returned by AUD_allocate_resource()
slot_id : This indicates which slot id has to be suspended. The slot ids upto 5 instances is supported. This slot id ranges from 0 to 4 for suspended data

Description: : This function clears all the suspended data contents for the specified slot_id so that the slot_id can be used for another instance.

Return codes (sent in AUD_DRIVER_RSP signal):

Return code	Comment
aud_rc_playback_stopped	The voice memo has been stopped by the application.

4.10 Polyphonic ringer

A set of interface function is dedicated to the polyphonic ringer. This could be an external dedicated chip (not supported on BP30 platform) or an internal polyphonic ringer. BP30 resource for polyphonic melodies is the MIDI player implemented in the DSP. So interface for polyphonic ringer. (AUD_ringer_* functions) calls again MIDI interface (corresponding AUD_midi_* functions). In this document functions are simply listed with their prototypes. For further description of the midi player refer to dedicated N7 document [6].

4.10.1 AUD_ringer_start

Prototype: SINT8 AUD_ringer_start(UINT8 handle, aud_ringer_id_enum tone_id, UINT16 nof_repeats, aud_ringer_device_enum device);

4.10.2 AUD_ringer_start_user_tone

Prototype: SINT8 AUD_ringer_start_user_tone(UINT8 handle, UINT16 huge *ringer_data, UINT16 nof_repeats, UINT32 size, aud_ringer_tone_format_enum format, aud_ringer_device_enum device, UINT8 channel, UINT8 channel_volume);

4.10.3 AUD_ringer_stop

Prototype: SINT8 AUD_ringer_stop(UINT8 handle, UINT8 channel);

4.10.4 AUD_ringer_suspend

Prototype: SINT8 AUD_ringer_suspend(UINT8 handle, UINT8 SlotID, UINT8 channel)

4.10.5 AUD_ringer_resume

Prototype: SINT8 AUD_ringer_resume(UINT8 handle, UINT8 SlotID, UINT8 channel)

4.10.6 AUD_ringer_stop_suspend

Prototype: SINT8 AUD_ringer_stop_suspend(UINT8 handle, UINT8 SlotID)

4.11 Info function

In this section, info functions are described. By info functions means a function returning information about internals in the audio driver. No resource allocation is needed for these functions.

4.11.1 AUD_info_hw_available

Prototype: UINT32 AUD_info_hw_available (void)

Parameters: None

Returns:

Description: Returns which resources there are available. The return code is decoded in the following way:

- **Bit 0: SPEECH**
- **Bit 1: TONE_GENERATOR**
- **Bit 2: POLYPHONIC RINGER**
- Bit 3: EXTERNAL_VIBRATOR (by external ringer)
- Bit 4: EXTERNAL_AMPLIFIER (by external ringer)
- **Bit 5: FM_RADIO**
- **Bit 6: RECORDING_VM**
- **Bit 7: PLAYBACK_VM**
- **Bit 8: PLAYBACK_MP3**
- Bit 9: I2S_PCM_CHANNEL
- **Bit 10: INTERNAL_MIDI**
- Bit 11: TTY
- **Bit 12: PLAYBACK_PCM**
- Bit 13: RECORD_PCM

(resources in bold are available on GLOBE6 platform)
Defines in aud_drv.h exist for this decoding (AUD_hw_parms).

4.12 Layer 1

4.12.1 aud_drv_check_timeout

Prototype: void aud_drv_check_timeout(void);
Parameters: None

Returns: None
Description: In order for the audio driver to work properly this function must be called from GSM layer 1 upon all frame interrupts.

4.13 Test/PC

This section describes the test functions supported by the audio driver. No resource allocation is needed for these functions.

4.13.1 AUD_dai_mode

Prototype: void aud_dai_mode(aud_dai_mode_enum dai_mode)
Parameters: aud_dai_emun dai_mode

DAI_mode
aud_dai_mode_normal
aud_dai_mode_codec_test
aud_dai_mode_acoustic_test
aud_dai_mode_loopback

Returns: None
Description: It sets the phone in the requested DAI mode. DAI is only used at type approval.

The DAI (Digital Audio Interface) is activable on I2Sy logic serial port. IS2y can be connected to physical port I2S1 or I2S2. By default on BP30 platform DAI runs on I2S1 and Blue Tooth devices are connected by logic port I2SX on physical port I2S2. Phisical ports connection can be swapped if necessary to have DAI on I2S2 port pins. (To have a version with DAI on I2S2 port, change in AUD_init function value of variable.

aud_dsp_swap_i2s_parms.swap=1; → *aud_dsp_swap_i2s_parms.swap=0;*

DAI signals are connected to I2S1 or I2S2 pins as shown in the table below:

signal	I2S1 (swap=1)	I2S2 (swap=0)
WA0_DAI	WA0_I2S1	WA0_I2S2
CLK0_DAI	CLK0_ I2S1	CLK0_ I2S2
TX_DAI	TX_ I2S1	TX_ I2S2
RX_DAI	RX_ I2S1	RX_ I2S2

If necessary remove HW components connected to these pins that could limit signal levels or speed.
Signal level range is 0-2.7V.
Phone under test can be switched in DAI mode by the Phone Tool :follow the menu “modes”, select “audio”, choose tab “E-GOLDLITE”; in the "DAI mode" combo box choose the DAI mode (i.e. for acoustic tests choose “acoustic test”; for different mode meanings, refer to [3]).
The phone tool will send to the phone the AT command to switch in DAI mode.
DAI modality (Normal mode, Codec, acoustic test, loopback) can be changed more that once in the same session, sending the correspondent command from the "DAI mode" combo box, followed by a DAI reset (sent by the external test set, i.e. UPL).

5 Return codes

In this chapter, there is a description of all the return code, which can be sent from the audio driver. In **Error! Reference source not found.** are the return codes, which is directly returned by a call to an interface function. In *Table 2* are listed the return codes which are sent in SDL-signal to the client.

Return codes, which can be returned from interface function calls	
Return code name	Description
aud_rc_ok	All is OK
aud_rc_resource_in_use	The requested resource is already allocated by another client
aud_rc_resource_conflict	The requested resource cannot be allocated because this conflicts with another already allocated resource
aud_rc_handle_not_used	The used handle is not valid
aud_rc_no_hw_support	The request cannot be performed because the needed HW is not present
aud_rc_sharing_violation	Access to a resource is done with wrong handle
aud_rc_parameter_out_of_range	One of the parameters are not valid
aud_rc_audio_driver_disabled	The audio driver has not been properly initialized

***Table 1** Return codes from the interface functions*

Return codes, which can be returned in SDL signal from the audio driver	
Return code name	Description
aud_rc_format_not_supported	The format is not supported.
aud_rc_missing_dsp_resources	Currently the DSP is busy with high priority task, and there for doesn't have the resources for completing the requested task. Note: Current Baseband chip soundplayer returning this return code, will be in idle and stop playing.
aud_rc_no_playback	Currently there is no playback, and there for the request is obsolete. E.g. If a suspend request is sent while no playback is active, this code is returned
aud_rc_unknown_position	If a ...set_position request is out of range this code is returned.
aud_rc_request_error	The request from the client is not possible in the given situation. E.g. Request to set the play position when playback is running in streaming mode.
aud_rc_syntax_error	The file requested for playback, is not valid. E.g. It could be syntax error in the header of a wave file.
aud_rc_tone_error	The requested tone could for some reason not be played
aud_rc_storage_problems	An error is reported by the storage system (e.g. FFS, MMC). The error code from the storage system can be seen in parm1.
aud_rc_performance_problems	Data are delivered either to fast or to slow to the audio driver, from the storage system
aud_rc_ram_buffer_used	When all data in the RAM buffer has been used, this code is returned. Parm1 contains the exact number of bytes recorded.
aud_rc_suspend_resume_error	A suspend or resume request was not possible. E.g. A client try to resume from a slot which does not contain any suspended data.
aud_rc_info	General info from the driver. The exact info depends on the sub-system (MP3, FM-radio etc)

aud_rc_playback_finish	All the requested data has been played
aud_rc_recording_finish	The recording process is finish. If recording to a file in the storage system, the client must wait for this code before the file can be closed. If recording to RAM, parm1 will contain the exact number of bytes recorded
aud_rc_playback_started	Send when playback is started. This code is also sent after a successful resume
aud_rc_playback_stopped	This code is returned after a stop request is received. Note: Parm4 will be 1 if the playback was stopped by driver instead of application
aud_rc_playback_loop	If the data should be played more than 1 time, this code will be send each time the data are played, except for the last time, here the aud_rc_playback_finish code is send
aud_rc_playback_suspended	Send when playback is suspended
aud_rc_recording_started	Sent when the recording process is started
aud_rc_elapsed_time	When the current play position is requested, this code is returned, with the result in parm1.
aud_rc_total_playtime	When the total playtime is request, this code is returned, with the result in parm1. If parm1 is 0xFFFFFFFF, it means that the playtime for some reason could not be calculated
aud_rc_current_frame	This is returned when there is request for current frame calculation. Parm1 contains the channel
aud_rc_backwardforward_info	Error – end of file or start of file reached
aud_rc_not_supported	If certain SW is not supported, e.g. Mp3, PCM etc
aud_rc_user_message	Unused
aud_path_removal_success	Used in Generic Module Test System only
aud_path_addition_success	Used in Generic Module Test System only
aud_path_addition_conflict	Used in Generic Module Test System only
aud_path_removal_error	Used in Generic Module Test System only

***Table 2** Return codes sent in SDL signals from the audio driver*

6 Audio parameters

Every audio path both in uplink and downlink can be characterized changing the values of a set of parameters. From the physical point of view these parameters influence the behaviour of the Audio Front End and the Audio Scheduler, both located in the voiceband dedicated portion of the TEAK DSP.
For every path, the sets of parameters path is divided in two subset, one is *AUD_setting* and its values can be choosen in development phase to characterize the phone type acoustic behaviour, the other one is *eep_static* and can be usefull to tune acoustically every single phone (acoustic calibration). These sets of parameters are saved in two structures of data in files aud_data.c (*AUD_setting*) and *eep.c (eep_static*).

6.1 AUD_setting

Structure Aud_setting contains parameter for battery charger, battery capacity estimation, liquid crystal display and audio part and it is defined as eep_default_type in file eep.h.

```
typedef struct
{
audio_uplink_parms_type    aud_audio_uplink_parms[AUDIO_UPLINK_PATHS + 1];
audio_downlink_parms_type  aud_audio_downlink_parms[AUDIO_DOWNLINK_PATHS+1 ]; }
aud_setting_type;
```

Audio parameters in EEP_default are organized in sub-structures eepaud_default_audio_uplink_parms_type and eepaud_default_audio_uplink_parms_type.
Space for 3 sets of uplink parameters and 5 sets of downlink in correspondence with the enumeration path in paragraph 4.2.1, 4.2.3. Both are still divided in sub-structures as appears in the tables below. For action of audio parameters in DSP audio scheduler, please refer to Audio driver presentation document or to E-GOLDlite Design Specification, par.16.1.4.11 :Voiceband Processing.

audio_uplink_parms		Range	Description
signed int16	scal_mic	0x0000-0x7FFF	Microphone digital gain
signed int16	delta0	0x0000-0x7FFF	Uplink to I2S2-Tx digital gain
signed int16	lambda0	0x0000-0x7FFF	Uplink to sample buffer digital gain
signed int16	lambda1	0x0000-0x7FFF	I2S2-Rx to sample buffer digital gain
signed int16	gamma0	0x0000-0x7FFF	Sample buffer to speech encoder gain
signed int16	gamma1	0x0000-0x7FFF	VM decoder to speech encoder gain
signed int16	alpha0	0x0000-0x7FFF	Sample buffer to VM encoder gain
signed int16	in1[5]	-1..1d	Biquad Filter In#1 [a1,b1,a2,b2,a0]
signed int16	in2[5]	-1..1d	Biquad Filter In#2 [a1,b1,a2,b2,a0]
signed int16	hf_algorithm_init	0x0000-0x7FFF	Control word for HF activation
signed int16	hf_algorithm_restart	0x0000-0x7FFF	Control word for HF restart
signed int16	step_width	0x0000-0x7FFF	LMS adaptation speed (step size)
signed int16	lms_length	0x0000-0x7FFF	LMS filter length (num. of coefficients)
signed int16	lms_offset	2 ... 400d	LMS filter offset (num.of skipped taps)
signed int16	block_length	0 ... 400d	LMS block update vector length
signed int16	rxtx_relation	{2, 4, 5, 8}	speaker to micro signal power relation
signed int16	add_atten	-960…960d	AGC additional attenuation
signed int16	min_atten	0…960d	AGC minimal attenuation
signed int16	max_atten	0…960d	AGC maximal attenuation

signed int16	nr_sw_2	0…960d	Noise reduction parameters
signed int16	nr_u_fak_0	0x0000-0x7FFF	Noise reduction parameters
signed int16	nr_u_fak	0x0000-0x7FFF	Noise reduction parameters
signed int16	mic_mute	0x0000-0x0001	Microphone muting (1 =active)
unsigned int16	mic_gain	0x0000-0x7FFF	Analog microphone gain
unsigned int16	tx_dither	0x0000-0x0001	*Unused*
unsigned int32	HardwareDependencies	0x0000-0x7FFF	Bitmap for HW dependencies of the path
unsigned char	ParallelPaths[4]	0x0000-0x0004	Array of indexes of parallel paths
unsigned char	Num_of_parallelPaths	0x0000-0x0004	Number of parallel paths

audio_downlink_parm		**Range**	**Description**
signed int16	gain_out	0x0000-0x7FFF	Downlink digital gain
signed int16	mix_fact_speech	0x0000-0x7FFF	Speech mixer factor
signed int16	mix_fact_tone	0x0000-0x7FFF	Tone generator mixer factor
signed int16	tone_amp	0x0000-0x7FFF	Tone generator amplitude
signed int16	delta1	0x0000-0x7FFF	Sample buffer to I2S2-Tx digital gain
signed int16	kappa0	0x0000-0x7FFF	Sample buffer to downlink digital gain
signed int16	kappa1	0x0000-0x7FFF	I2S2-Rx to downlink digital gain
signed int16	alpha1	0x0000-0x7FFF	Speech decoder to VM encoder dig.gain
signed int16	beta0	0x0000-0x7FFF	Speech decoder to Sample buffer d.gain
signed int16	beta1	0x0000-0x7FFF	VM decoder to Sample buffer dig.gain
signed int16	out1[5]	-1..1d	Biquad Filter Out#1 [a1,b1,a2,b2,a0]
signed int16	out2[5]	-1..1d	Biquad Filter Out#2 [a1,b1,a2,b2,a0]
signed int16	hf_algorithm_init	0x0000-0x7FFF	Control word for HF activation
signed int16	hf_algorithm_restart	0x0000-0x7FFF	Control word for HF restart
signed int16	step_width	0x0000-0x7FFF	LMS adaptation speed (step size)
signed int16	lms_length	0x0000-0x7FFF	LMS filter length (num. of coefficients)
signed int16	lms_offset	2 ... 400d	LMS filter offset (num.of skipped taps)
signed int16	block_length	0 ... 400d	LMS block update vector length
signed int16	rxtx_relation	{2, 4, 5, 8}	speaker to micro signal power relation
signed int16	add_atten	-960…960d	AGC additional attenuation
signed int16	min_atten	0…960d	AGC minimal attenuation
signed int16	max_atten	0…960d	AGC maximal attenuation
signed int16	nr_sw_2	0…960d	Noise reduction parameters
signed int16	nr_u_fak_0	0x0000-0x7FFF	Noise reduction parameters
signed int16	nr_u_fak	0x0000-0x7FFF	Noise reduction parameters
signed int16	scal_rec[9]	0x0000-0x7FFF	Volume steps for downlink digital gain
signed int16	side_tone_fact[9]	0x0000-0x7FFF	Volume steps for digital side tone gain
unsigned int16	afe_rxgainlo[9]	0x0000-0x7FFF	Volume steps for analog handset gain
unsigned int16	afe_rxgainpa[9]	0x0000-0x7FFF	Volume steps for analog headset gain
unsigned int16	zero_detect	0x0000-0x0001	*Unused*
unsigned int32	HardwareDependencies	0x0000-0xFFFF	Bitmap for HW dependencies of the path
unsigned char	ParallelPaths[4]	0x0000-0x7FFF	Array of indexes of parallel paths
unsigned char	Num_of_parallelPaths	0x0000-0x0004	Number of parallel paths

Gain parameters are linear with signal amplitude. For ex., if 0x1FFF =0db, then max value 0x7FFF (4x1FFF) means 20*log(4)= +12dB . If gain is 0 output amplitude is 0.

Gain Parameters	Default (reset)	Value for 0 dB gain	Max gain
Scal_In	0x2000	0x1FFF	+12dB
Scal_Out	0x2000	0x1FFF	+12dB
Side_Ton	0x2000	0x3FFF	+6dB
Scal_Mic	0x2000	0x1FFF	+12dB
Gain_Out	0x2000	0x1FFF	+12dB
Scal_Rec	0x2000	0x1FFF	+12dB
Speech_Mix	0x1000	0x7FFF	0dB
Tone_Mix	0x0000	0x7FFF	0dB
Delta0	0x0000	0x7FFF	0dB
Delta1	0x0000	0x7FFF	0dB
Kappa0	0x7FFF	0x7FFF	0dB
Kappa1	0x0000	0x7FFF	0dB
Lambda0	0x7FFF	0x7FFF	0dB
Lambda1	0x0000	0x7FFF	0dB
Alpha0	0x0000	0x7FFF	0dB
Alpha1	0x0000	0x7FFF	0dB
Beta0	0x7FFF	0x7FFF	0dB
Beta1	0x0000	0x7FFF	0dB
Gamma0	0x7FFF	0x7FFF	0dB
Gamma1	0x0000	0x7FFF	0dB
Scal_SAPP	0x7FFF	0x3FFF	+6dB
Scal_Ext	0x7FFF	0x3FFF	+6dB
Mix_AFE	0x0000	0x3FFF	+6dB
Mix_I2S1	0x0000	0x3FFF	+6dB

All the parameters are coded as unsigned 15 bit except biquad filter parameters that are signed 16 bit in 2's complement. So for biquad filters parameters [a1, b1, a2, b2, a0] range is -1..+1 and values are

0x7FFF = +1
0x8000 = -1
0x0000 = 0

The hardware dependence of every path can be fixed with bit mask HardwareDependencies in structure *EEP_default.* (See par. 5.1). Bit mask is defined as shown in the table below and it is decoded by function *configure_afe* in file *Aud_lib.c.*

External DAC(Max 9850)	AFE_MIC 2 -- On/Off	AFE_MIC 1 -- On/Off	AFE_MIC Supply -- On/Off	EPp1EPn1 -- On/Off	HeadsetAmp -- On/Off	E Power AMP-- ON/Off	I2S2 for external DAC -- On/Off	I2Sx-ON	AFE -- On/Off	AFE_EPpa1EPpa2EPref	AFE_EPpa1EPpa2_On	AFE_OnlyEPpa1	AFE_EPpa1EPpa2Stereo	AFE_EPpa1EPpa2EPp1EPn1_mono	AFE_EPpa1EPpa2_mono	AFE_EPp1EPn1_mono	DAC left	DAC
18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

6.2 EEP_static

Constant array EEP_static conteins calibration parameters whose values are calculated during production test to compensate dispersion of characteristics of every single terminal.
Structure of this array is defined in file eep.h.
It reserves space for audio calibration parameters:
  eepaud_static_scal_in_parms_type   aud_scal_in_parms[15];
  eepaud_static_scal_out_parms_type   aud_scal_out_parms[15];

```
typedef struct
{
   signed int16        scal_in;
}eepaud_static_scal_in_parms_type;

typedef struct
{
  signed int16        scal_out;
}eepaud_static_scal_out_parms_type;
```

So uplink digital gain *scal_in* can be calculated and saved for every uplink path and downlink digital gain *scal_out* for every downlink path.
Index to be used for these tables should be values of aud_uplink_source_enum, aud_downlink_source_enum (see par.4.2.1,4.2.3)
If the phone is not calibrated in audio, these values remain as defined in source code:
scal_in=0x4000, scal_out=0x2000 for every path.

7 References

7.1 External

ETSI Specs

N.A.

Relevant FTA TestCases

N.A.

Others

[1] Interface specification template, Rev. 0.9.60 Author: Jan A. Mylund , 09/02/2006
[2] E-GOLDRadio Design Specification, Rev. 1.05, 2005-08-02
[3] E-GOLDRadio Firmware Manual, Rev. 1.00, June 2005

7.2 Internal

Title	**Doc ID**
[4] Audio driver presentation	BH02.S2.PP.000019
[5] FM Radio Driver specification	AH01.SW.TS.000013
[6] Midi driver Specification	AH01.SW.TS.000022

8 Document change report

	Change Reference		**Record of changes made to previous released version**	
Rev	**Date**	**CR**	**Section**	**Comment**
1.0	05/01/2006	N.A.	N.A.	Document created
1.1	21/06/2004			
1.2	16/02/2006	N.A.	Document updated to BP30 Platform	

9 Approval

Revision	**Approver(s)**	**Date**	**Source/signature**
1.1	Stefano Godeas	21/06/2004	Document stored on server
1.2	Stefano Godeas	16/02/2006	Document stored on server