Freescale Audio decoder Library wrapper API Specification

Version: 1.0

Date: Sep. 9, 2011

Freescale Semiconductor, No. 192 Liangjing Rd., Pu Dong New Area, Shanghai, 201203, PRC

Revision History

Version	Date	Revised By	Description of Changes
1.0	09/09/2011	Lyon Wang	Initial version.

1. Introduction

1.1. In this document

This document is the common API specification for the wrapper of following core decoder libraries:
- AAC-LC
- AAC plus (HE-AAC / HE-AACv2)
- AC3
- MP3
- Ogg-vorbis
- WMA10
- nb-amr
- FLAC
- Real-audio

The calling sequence of the API functions is also explained.

The API is declared in the header file “fsl_unia.h”.

1.2. References

Specification of supported media formats.
Specification of supported media format API documents

2. API Functions

2.1. Query a Codec's Interface

int32 tUniACodecQueryInterface (uint32 id, void ** func);

Description:

A core codec wrapper library shall be built as a shared library and this function must be implemented as the “entry function”

It’s the 1st function to call after the codec shared library is opened. And it can return all other API function pointers implemented by the core audio codec wrapper, such as creating/deleting the core codec

All other API functions shall obey the prototypes defined in the left part of this section.

Arguments:

Id	[in]	The ID of the API function to query.
func	[out]	The related API function pointer. It must obey the prototype of the function ID. If the related API is optional and not implemented by the core codec, set this value to NULL.

Return value:

ACODEC_SUCCESS	Success
Other error codes	Failure

2.2. Creation and Deletion

All functions in this section must be implemented.

2.2.1. Get core Codec Version

typedef const char * (*UniACodecVersionInfo)();

Description:

Function to get the core codec version.

API ID: ACODEC_API_GET_VERSION_INFO

Return value:

The version string of core audio codec.

2.2.2. Codec Wrapper creation

typedef UniACodec_Handle (*UniACodecCreate)(UniACodecMemoryOps * memOps);
Description:
Function to create the core codec.
API ID: ACODEC_API_CREATE_CODEC

Arguments:

memOps	[in]	Memory operation callback table. It implements the functions to malloc, calloc, realloc and free memory.

Return value:

UniACodec_Handle	Created pua_handle point
NULL	Failure

2.2.3. Codec Wrapper delete

typedef int32 (*UniACodecDelete) (UniACodec_Handle pua_handle);

Description:
function to delete the CODEC. This is the last API to call.
API ID: ACODEC_API_DELETE_CODEC

Arguments:

pua_handle	[in]	Handle of codec wrapper.

Return value:

ACODEC_SUCCESS	Success

Other error codes	Failure

2.3. Parameter Setting and Getting.

Many parameters are needed to setting into the Cdoec. Vary from each codecs. The unify API will gather these different type of parameter and using parameter set and get API for this.

2.3.1. Parameter Setting

```
typedef int32 (*UniACodecSetParameter) (UniACodec_Handle pua_handle
                                        UA_ ParaType ParaType,
                                        UniACodecParameter * parameter,
                                        );
```

Description:
Function to setting the parameter to codec wrapper according to the parameter type.
API ID: ACODEC_API_SET_PARAMETER

Arguments:

pua_handle	[in]	Handle of core codec wrapper
ParaType	[in]	The parameter type to set
parameter	[in]	The point of parameter structure

Return value:

ACODEC_SUCCESS	Success
Other error codes	Failure

2.3.2. Parameter Getting

```
typedef int32 (*UniACodecGetParameter) (UniACodec_Handle pua_handle
                                        UA_ ParaType ParaType,
                                        UniACodecParameter * parameter,
                                        );
```

Description:
Function to setting the parameter to codec wrapper according to the parameter type.
API ID: ACODEC_API_GET_PARAMETER

Arguments:

pua_handle	[in]	Handle of core codec wrapper
ParaType	[in]	The parameter type to set
parameter	[out]	The point of parameter structure

Return value:

void *	The data point return according to parameter
NULL	Failure

2.4. Process Frame

2.4.1. Frame decoding

```
typedef int32 (*UniACodec_decode_frame) (UniACodec_Handle pua_handle,
                                         uint8 * InputBuf,
                                         uint32 InputSize,
                                         uint32 *offset
```

uint8 ** OutputBuf
uint32 * OutputSize);

Description:
Function to decode frame.
API ID: ACODEC_API_DEC_FRAME

Arguments:

pua_handle	[in]	Handle of core codec wrapper
InputBuf	[in]	Input Buffer of bitstream, If set to NULL means decoder will decoding all data buffer within the wrapper
InputSize	[in]	The input buffer size for core decoder wrapper
offset	[in/out]	The offset of input buffer to core decoder wrapper, and will be updated by decpder wrapper. When offset is equal to input size, means the input buffer is all consumed
OutputBuf	[in/out]	Output Buf for decoder output, if set it to NULL, decoder wrapper will malloc output buffer inside the decoder.
OuputSzie	[in/out]	Output size for decoder output, if set it to NULL, decoder wrapper

Return value:

ACODEC_SUCCESS	Success
Other return	Decoding error

2.5. Reseting

2.5.1. Cdoec wrapper reset

typedef int32 (*UniACodecReset) (UniACodec_Handle pua_handle)

Description:
Function to reset codec wrapper (flushing inner buffer etc.).
API ID: ACODEC_API_RESET_CODEC

Arguments:

pua_handle	[in]	Handle of codec wrapper.

Return value:

ACODEC_SUCCESS	Success
Other error codes	Failure

2.6. Oher

2.6.1. Get last Error

typedef char * (*UniACodec_get_last_error) (UniACodec_Handle pua_handle);
Description:
Function to get the error string of the codec
API ID: ACODEC_API_GET_LAST_ERROR
Arguments:

pua_handle	[in]	Handle of codec wrapper.

Return value:
The error string of the decoder error;

3. Data Types & Constants

3.1. Error Codes

Common Error Codes	Comments
ACODEC_SUCCESS	success
ACODEC_PARA_ERROR	Invalid parameter
ACODEC_INSUFFICIENT_MEM	Not enough memory allocate for core codec
ACODEC_ERROR_STREAM	Decoding error
ACODEC_ERR_UNKNOWN	Unknown failure, not captured by codec logic
ACODEC_PROFILE_NOT_SUPPORT	Stream profile not support
ACODEC_NOT_ENOUGH_DATA=0x100	Not enough input data flag
ACODEC_CAPIBILITY_CHANGE = 0X200	Output format change flag
ACODEC_END_OF_STREAM=0X300	Reaching the end of stream, no more data.

3.2. API Function ID List

```
enum /* API function ID */
{
   ACODEC_API_GET_VERSION_INFO  = 0x0,
   /* creation & deletion */
   ACODEC_API_CREATE_CODEC     = 0x1,
   ACODEC_API_DELETE_CODEC     = 0x2,
   /* reset */
   ACODEC_API_RESET_CODEC = 0x3,

   /* set parameter */
   ACODEC_API_SET_PARAMETER  = 0x10,
   ACODEC_API_GET_PARAMETER  = 0x11,

   /* process frame */
   ACODEC_API_DEC_FRAME    = 0x20,
   //ACODEC_API_ENC_FRAME    = 0x21,

   ACODEC_API_GET_LAST_ERROR = 0x1000,

};
```

3.3. Handle of Codec Wrapper

```
typedef void * UniACodec_Handle;
```

3.4. Structure of Parameter

```
    typedef struct
```

```
{
    union {
        uint32 samplerate;
        uint32 channels;
        uint32 bitrate;
        uint32 depth;
        uint32 blockalign;
        uint32 version;
        bool downmix;
        bool framed;
        UniACodecParameterBuffer codecData;

        char ** codecDesc;
        UniAcodecOutputPCMFormat outputFormat;
    };
}UniACodecParameter;
```

3.5. Other Constants

4. Callback Functions

4.1. Memory operation callback

```
typedef struct
{
    void* (*Calloc) (uint32 numElements, uint32 size);
    void* (*Malloc) (uint32 size);
    void  (*Free) (void * ptr);
    void* (*ReAlloc)(void * ptr, uint32 size); /* necessary for index scanning!*/

}UniACodecMemoryOps; /* callback operation callback table */
```

Description
Callback functions of memory operations for codec wrapper.

5. API Calling Sequence

This section describes the API Calling Sequence. Without explicitly explanation, the user shall go ahead only if previous step succeeds.

Please refer to the test application code for more details.

- **Open the codec wrapper shared library, and query its interface to get all implemented function pointers.**

  UniACodecQueryInterface

- **Check core codec version (optional)**

  ACodecVersionInfo ()

- **Create the codec**

  ACodecCreate ()

  As long as the codec is created successfully, it must be deleted as the final step to free the resources.

- **Set the needed parameter for according parameter type**

  ACodecSetParameter()

- **Proecess audio frame**

  ACodec_decode_frame()

- **Get ouput parameter needed from the core codec**

  ACodecGetParameter()

- **Delete the core codec**

  ACodecDelete()

6. API Calling Flowchart