

Android New Audio Decoder Porting USER GUIDE

TCCxxxx_ALL-AG-5441_V1.0.0E

May 30, 2012

Telechips

Revision History

Date	Version	Description
2012-05-30	1.00	Initial Release

TABLE OF CONTENTS

Contents

1 Introduction.....	1-4
2 Preparation Steps	2-5
3 How to modify.....	3-6
3.1 Specify the audio decoder name	3-6
3.2 Port_Setting	3-7
3.3 Declaration decoder structure	3-8
3.4 Modify the API functions.....	3-9
3.4.1 ThirdPartyAudio_OpenDecoder	3-9
3.4.2 ThirdPartyAudio_InitDecoder.....	3-10
3.4.3 ThirdPartyAudio_DecodeFrame.....	3-12
3.4.4 ThirdPartyAudio_FlushDecoder	3-14
3.4.5 ThirdPartyAudio_CloseDecoder.....	3-15
4 How to modify the makefile	4-16
4.1 Using source code.....	4-16
4.1.1 Modify the makefile for your decoder library	4-16
4.1.2 Modify the makefile for component	4-18
4.2 Using Pre-built library.....	4-20

1 Introduction

In recent years, many multimedia codecs are connected to higher software layer through the OpenMAX IL standard. In our Android SDK, this standard has also been used to handle the multimedia contents.

This manual describes how to make a new OpenMAX IL component to operate audio decoder within the telechips's android system.

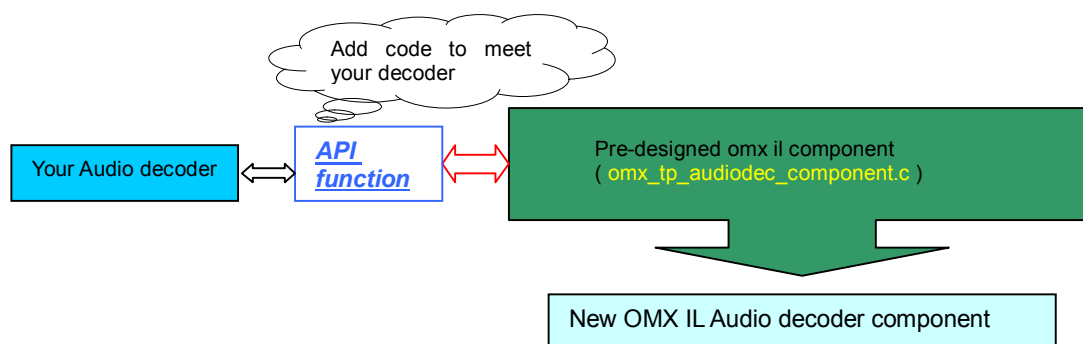
You might want to add a new audio decoder or replace the previous decoder with new one of good performance.

For this, Our SDK provides the pre-designed omx component and the API functions as below Figure 1.1. This pre-designed component is connected to your decoder through the API function.

You can make a component for your decoder by using these API functions, to which you can add a code for your audio decoder that you want to use.

You can also make an omx il compatible component regardless of this manual. However, by using the API functions and the pre-designed component provided you can make a new audio component easily without understanding of OMX IL. Moreover, you can use the complex functionality such as seeking as well as the basic operation such as play back or timestamp calculation, without effort for additional coding.

This document will address only how to connect your audio decoder with the API function.



- Figure 1.1 -

By simply adding your code to the appointed section of the API functions you can make the OMX IL component for your audio decoder.

You need only to add the interface code for your decoder into the API function files below.

Below files are for the interface between the [omx_tp_audiodec_component.c](#) and 3rd-party audio decoder.

File name	
third_party_audio_dec_api.c	To connect pre-designed code and your decoder's functions
third_party_audio_dec_api.h	Definitions, declarations and etc needed to compile the third_party_audio_dec_api.c
third_party_predefine.h	To declare parameter type needed automatically

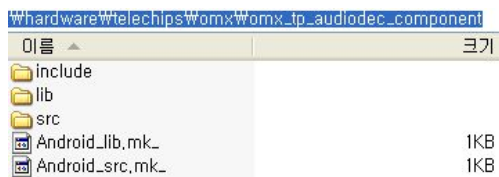
- Table 1.1 API function files -

These file correspond to the [API function](#) in the Figure 1.1.

2 Preparation Steps

OMX IL component code to merge 3rd-party audio decoder easily is located in the below directory.

[\hardware\telechips\omx\omx_tp_audiodec_component](#)

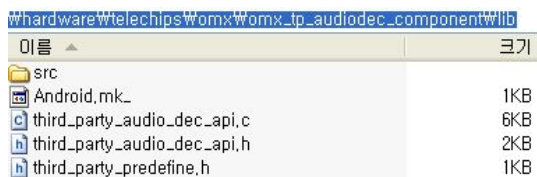


이름	크기
include	
lib	
src	
Android_lib.mk_	1KB
Android_src.mk_	1KB

The structure of this directory is as below table.

Root-directory	Sub-directory	Description
src	omx_tp_audiodec_component.c	omx component source code for 3 rd -Party audio decoder
include	omx_tp_audiodec_component.h	omx component header file
lib	third_party_audio_dec_api.c	API codes for 3 rd -Party audio decoder
	third_party_audio_dec_api.h	Required definitions and declarations
	third_party_predefine.h	For automatic declarations
	Android.mk_	makefile to build 3 rd -Party library, see 4.1.1
Android_lib.mk_		makefile to build component, see 4.2
Android_src.mk_		makefile to build component, see 4.1.2

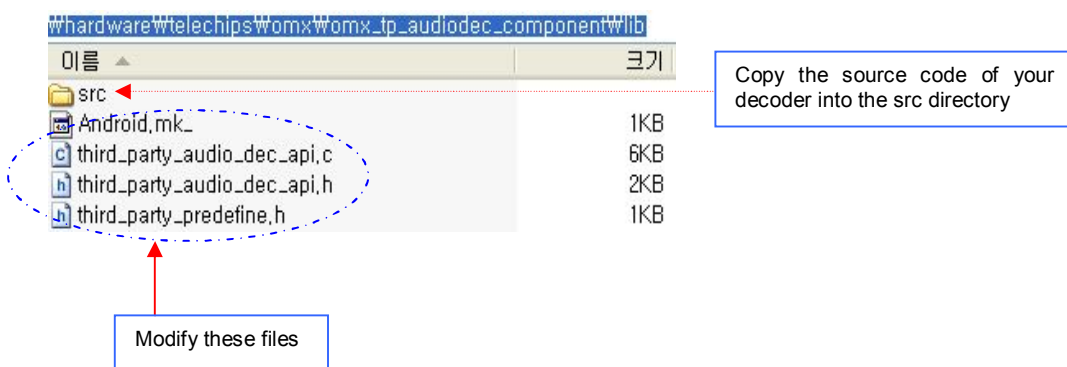
The API functions for the 3rd-Party audio decoder is located in the lib directory as below figure.



이름	크기
src	
Android.mk_	1KB
third_party_audio_dec_api.c	6KB
third_party_audio_dec_api.h	2KB
third_party_predefine.h	1KB

Copy the source code of your decoder that you want to use into the below directory.

[\hardware\telechips\omx\omx_tp_audiodec_component\lib\src](#)



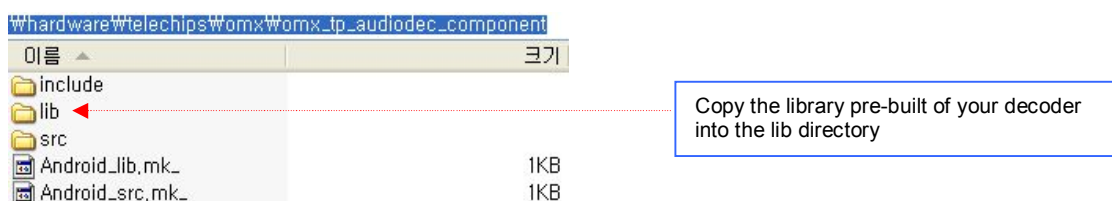
Copy the source code of your decoder into the src directory

Modify these files

If your decoder is in the form of library.

Copy the pre-built library of your decoder into the below directory.

[\hardware\telechips\omx\omx_tp_audiodec_component\lib](#)



Copy the library pre-built of your decoder into the lib directory

3 How to modify

This chapter describes how to modify the API function related files in the table 1.1 to fit your own decoder. These files are for the interface between the [omx_tp_audiodec_component.c](#) and 3rd-party audio decoder.

3.1 Specify the audio decoder name

For automatical declaration of the necessary variables used in the [omx_tp_audiodec_component.c](#) to fit your decoder, you must specify the audio decoder name.

Open the [third_party_predefine.h](#) file.

Write your decoder name into this file as below.

```
#ifndef _THIRD_PARTY_PRE_DEFINE_H_
#define _THIRD_PARTY_PRE_DEFINE_H_

#define CODEC_NAME          xxxdec          // TODO: modify
#define CODEC_TYPE          XXX             // TODO: modify

#endif /* _THIRD_PARTY_PRE_DEFINE_H_ */
```

CODEC_NAME

specify the codec name, write in small letters (write freely, there are no special rule)

CODEC_TYPE

Change this field to suit your own decoder.

You must use only the specified value defined in the **OMX_Audio.h**.

This file is located in the [frameworks/base/include/media/stagefright/openmax](#) directory.

The codec types are defined as below form.

[OMX_AUDIO_PARAM_XXXTYPE](#)

Copy the red colored part, except 'TYPE' character, into the [third_party_predefine.h](#).

For example, in case of FLAC decoder, the codec type is defined in the **OMX_Audio.h** as follows:

```
/** FLAC params */
typedef struct OMX_AUDIO_PARAM_FLAC {
    OMX_U32 nSize;          /**< size of the structure in bytes */
    OMX_VERSIONTYPE nVersion; /**< OMX specification version information */
    OMX_U32 nPortIndex;     /**< port that this structure applies to */
    OMX_U32 nChannels;       /**< Number of channels */
    OMX_U32 nBitRate;        /**< Bit rate of the input data. Use 0 for variable
                             rate or unknown bit rates */
    OMX_U32 nSampleRate;     /**< Sampling rate of the source data. Use 0 for
                             variable or unknown sampling rate. */
    OMX_AUDIO_CHANNELMODETYPE eChannelMode; /**< Channel mode enumeration */
} OMX_AUDIO_PARAM_FLAC;
```

Then modify the [third_party_predefine.h](#) as below.

```
#define CODEC_NAME          flacdec
#define CODEC_TYPE          FLAC
```

3.2 Port_Setting

To operate the OpenMAX IL component, you have to set the component's port.

Open the [third_party_audio_dec_api.c](#) in the `\hardware\telechips\omx\omx_tp_audiodec_component\lib` directory. Find the Port_Setting function and change mime-type part as below.

You only have to modify the red colored part.

```
void Port_Setting(component_PrivateType* pPrivate)
{
    omx_base_audio_PortType *inPort,*outPort;

    /** parameters related to input port */
    inPort = (omx_base_audio_PortType *) pPrivate->ports[OMX_BASE_FILTER_INPUTPORT_INDEX];

    inPort->sPortParam.nBufferSize = DEFAULT_IN_BUFFER_SIZE*2;

    //TODO: modify mime-type name below to fit your decoder
    strcpy(inPort->sPortParam.format.audio.cMIMETYPE, "audio/XXX");
    ///////////////////////////////////////////////////////////////////

    inPort->sPortParam.format.audio.eEncoding = audio_coding_type;
    inPort->sAudioParam.eEncoding = audio_coding_type;

    /** parameters related to output port */
    outPort = (omx_base_audio_PortType *) pPrivate->ports[OMX_BASE_FILTER_OUTPUTPORT_INDEX];
    outPort->sPortParam.format.audio.eEncoding = OMX_AUDIO_CodingPCM;
    outPort->sPortParam.nBufferSize = AUDIO_DEC_OUT_BUFFER_SIZE;
    outPort->sAudioParam.eEncoding = OMX_AUDIO_CodingPCM;

    setHeader(&pPrivate->pAudio, sizeof(audio_param_type));
    pPrivate->pAudio.nPortIndex = 0;
    pPrivate->pAudio.nChannels = 2;
    pPrivate->pAudio.nBitRate = 0;
    pPrivate->pAudio.nSamplingRate = 44100;
    pPrivate->pAudio.eChannelMode = OMX_AUDIO_ChannelModeStereo;

    /** settings of output port audio format - pcm */
    setHeader(&pPrivate->pAudioPcmMode, sizeof(OMX_AUDIO_PARAM_PCMMODETYPE));
    pPrivate->pAudioPcmMode.nPortIndex = 1;
    pPrivate->pAudioPcmMode.nChannels = 2;
    pPrivate->pAudioPcmMode.eNumData = OMX_NumericalDataSigned;
    pPrivate->pAudioPcmMode.eEndian = OMX_EndianLittle;
    pPrivate->pAudioPcmMode.blInterleaved = OMX_TRUE;
    pPrivate->pAudioPcmMode.nBitPerSample = 16;
    pPrivate->pAudioPcmMode.nSamplingRate = 44100;
    pPrivate->pAudioPcmMode.ePCMMode = OMX_AUDIO_PCMModeLinear;
    pPrivate->pAudioPcmMode.eChannelMapping[0] = OMX_AUDIO_ChannelLF;
    pPrivate->pAudioPcmMode.eChannelMapping[1] = OMX_AUDIO_ChannelRF;
}
```

The strings used for MIMETYPE such as "audio/XXX" in the above are defined in the [MediaDefs.cpp](#), which is located in the `frameworks/base/media/libstagefright` directory.

The MIME Type on the **Port_Setting** function and the file must exactly same.

For example, in case of FLAC, the MIME Type for FLAC is defined in this file as below:

```
const char *MEDIA_MIMETYPE_AUDIO_FLAC = "audio/flac";
```

and you must change the cMIMETYPE as below:

```
strcpy(inPort->sPortParam.format.audio.cMIMETYPE, "audio/flac");
```

3.3 Declaration decoder structure

Open the [third_party_audio_dec_api.c](#) file.

Add any variables required for your decoder implementation within the green colored part of the [ThirdPartyAudio_t](#).

This structure is an instance (object) of your decoder and this is used in the API functions in the section 3.4.

```
typedef struct ThirdPartyAudio\_t
{
    // TODO: add member variables needed

    //////////////////////////////////////
} ThirdPartyAudio\_t;
```

The variables and structures, pointers, arrays and etc, everything you need to connect your decoder.
See section 3.4.1 [ThirdPartyAudio_OpenDecoder](#)

Example:

```
typedef struct ThirdPartyAudio\_t
{
    // TODO: add member variables needed
    FLAC__StreamDecoder *decoder
    int *flac_ch_data[8];
    short output_pcm[2][4608];

    int samplerate;
    int number_of_channel;
    int usedbytes;
    //////////////////////////////////////
} ThirdPartyAudio\_t;
```


3.4 Modify the API functions

In the [third_party_audio_dec_api.c](#), The functions that should be modified are as follows:

Function Name	Description
ThirdPartyAudio_OpenDecoder	To open 3 rd -party audio decoder
ThirdPartyAudio_InitDecoder	To init 3 rd -party audio decoder
ThirdPartyAudio_DecodeFrame	To decode audio stream using 3 rd -party audio decoder
ThirdPartyAudio_FlushDecoder	To flush internal buffer of the 3 rd -party audio decoder
ThirdPartyAudio_CloseDecoder	To close 3 rd -party audio decoder

The next section will explain how to modify these functions.

3.4.1 ThirdPartyAudio_OpenDecoder

```
void* ThirdPartyAudio_OpenDecoder(
    component_PrivateType* pPrivate
)
```

This function is used to create an instance of 3rd-party decoder.

Input

pPrivate is component handle. If you do not need this, just ignore.

Return Value

Decoder handle created.

If there is a problem must return NULL.

Normally, you can write codes for the memory allocation and default setting needed to operate your own decoder.

The basic code needed has already been implemented.

You only have to add additional code to open your decoder into the green colored part If necessary.

```
void* ThirdPartyAudio_OpenDecoder(
    component_PrivateType* pPrivate
)
{
    ThirdPartyAudio_t *pDecoder; // see section 3.2 Declaration decoder structure

    pDecoder = (ThirdPartyAudio_t *)malloc(sizeof(ThirdPartyAudio_t));
    if(pDecoder == NULL)
        return NULL;

    memset(pDecoder, 0, sizeof(ThirdPartyAudio_t));

    LOGD( "ThirdParty Decoder Open\n");

    // TODO: add additional opening code here

    ///////////////////////////////////////////////////////////////////

    return (void *)pDecoder;
}
```

3.4.2 ThirdPartyAudio_InitDecoder

```
OMX_S32 ThirdPartyAudio_InitDecoder(
    component_PrivateType* pPrivate,
    OMX_BUFFERHEADERTYPE* inputbuffer,
    OMX_BUFFERHEADERTYPE* outputbuffer,
    OMX_S32 *piDecoderSampleRate,
    OMX_S32 *piDecoderChannels
)
```

This function is used to initialize the 3rd-party audio decoder.

If additional information from the user or application layer is required to initialize the decoder, this function can be used.

If this information does not needed, this function need not be modified.

Input

pPrivate

is component handle.

You can get the decoder instance opened by the [ThirdPartyAudio_OpenDecoder](#) function like below:

```
ThirdPartyAudio_t *pDecoder = (ThirdPartyAudio_t *)pPrivate->pDecoder;
```

You can get the information required to initialize the decoder from the below structure:

```
pPrivate->cdmx_info.m_sAudioInfo
```

Examples:

```
pPrivate->cdmx_info.m_sAudioInfo.m_iSamplePerSec; // samplerate
pPrivate->cdmx_info.m_sAudioInfo.m_iChannels; // number of channel
pPrivate->cdmx_info.m_sAudioInfo.m_iBitsPerSample; // bits/sample
```

If you need the codec specific configuration data such as AudioSpecificConfig data for AAC, you can get these data as below:

```
pPrivate->cdmx_info.m_sAudioInfo.m_pExtraData; // buffer pointer of codec specific configuration data;
pPrivate->cdmx_info.m_sAudioInfo.m_iExtraDataLen; // length of codec specific configuration data;
```

For more information about the [m_sAudioInfo](#) structure, confirm the [cdmx_info_t](#) structure in the [cdmx.h](#) file, this file is located in the below directory.
/hardware/telechips/common/CDK/container/cdmx.h

If you do not need these information, just ignore.

inputbuffer

is a pointer to input buffer header. [In the normal case, not used!](#)

outputbuffer

is a pointer to output buffer header pointer. [In the normal case, not used!](#)

Output

piDecoderSampleRate is a pointer to store output sample-rate of 3rd-party decoder. (See [**1](#) below, normally, not used!)

piDecoderChannels is a pointer to store the number of output channels. (See [**1](#) below, normally, not used!)

Return Value

This function should return one of the following values:

0 : Success.

Non zero : Failure.

The basic code has already been implemented.

You only have to add additional code to initialize your decoder into the green colored part If necessary.

```
OMX_S32 ThirdPartyAudio_InitDecoder(
    component_PrivateType* pPrivate,
    OMX_BUFFERHEADERTYPE* inputbuffer,
    OMX_BUFFERHEADERTYPE* outputbuffer,
    OMX_S32 *piDecoderSampleRate,
    OMX_S32 *piDecoderChannels
)
{
    OMX_S32 ret = 0;
    ThirdPartyAudio_t *pDecoder = (ThirdPartyAudio_t *)pPrivate->pDecoder; // get decoder handle

    // TODO: add additional initialization code here
    Example)
```

```
pDecoder->set_samplerate = pPrivate->cdmx_info.m_sAudioInfo.m_iSamplePerSec;  
init_xxx_decoder();  
////////////////////////////////////  
return ret;  
}
```

** 1

In the case of some decoder, the information about the output samplerate and number of channels received from application can be changed because of the following reasons:

- A. Do not support
- B. Output is fixed
- C. Any other reason

If your decoder is so, you should notify this information. **or just ignore**

3.4.3 ThirdPartyAudio_DecodeFrame

```
OMX_S32 ThirdPartyAudio_DecodeFrame(
    component_PrivateType* pPrivate,
    OMX_U8 *puclInput,
    OMX_S32 ilInputLength,
    OMX_PTR *pOutBuff,
    OMX_S32 *piUsedBytes,
    OMX_S32 *piDecodedSamples,
    OMX_S32 *piOutSampleRate,
    OMX_S32 *piOutChannels
)
```

This function decodes the frame data.

Input

pPrivate is component handle.
puclInput is a pointer to input stream buffer.
 Read the encoded bit-stream from here.
ilInputLength is the amount of input bit-stream in bytes.
pOutBuff is a pointer to output pcm buffer.
 Save the decoded PCM samples here in interleaved format.

Output

piUsedBytes (you must inform the corresponding information through the following parameters.)
 is a pointer to store how many bytes the decoder has read (used bytes of input buffer)
 In the component layer outside, this parameter is used to manage the input stream buffer.
 And the data used will be discarded in the component layer.
 Therefore, you must tell the amount of bytes of the input data used using this parameter.
 *piUsedBytes = usedbytes;
piDecodedSamples is a pointer to store how many samples were decoded (number of total samples, not bytes)
 This parameter is used to update the PCM related buffer and calculate the timestamp outside.
 You must inform the number of total PCM samples generated using this parameter.
 *piDecodedSamples = total_samples (= number_of_samples_per_channel * number_of_channel);
piOutSampleRate is a pointer to store the samplerate of decoded PCM.
 *piOutSampleRate = samplerate_of_decoded_pcm;
piOutChannels is a pointer to store the number of channels of decoded PCM.
 * piOutChannels = number_of_channels_of_decoded_pcm;

Return Value

This function should return one of the following values:

Return value	Meaning
0	Decoding success
1	Need more bitstream data, if there is insufficient data
2	This is recoverable, just ignore the current frame
Otherwise	Unrecoverable error, the remaining data in the input bit-stream buffer will be discarded

You only have to add the code to decode a frame into the green colored part.

```
OMX_S32 ThirdPartyAudio_DecodeFrame(
    component_PrivateType* pPrivate, // (i) component handle
    OMX_U8 *puclInput, // (i) input stream buffer pointer
    OMX_S32 ilInputLength, // (i) input stream length (in bytes)
    OMX_PTR *pOutBuff, // (i) output pcm buffer pointer
    OMX_S32 *piUsedBytes, // (o) used bytes (number of bytes used to decode)
    OMX_S32 *piDecodedSamples, // (o) number of total samples decoded
    OMX_S32 *piOutSampleRate, // (o) samplerate of the current frame decoded
    OMX_S32 *piOutChannels // (o) number of channel
)
{
    OMX_S32 ret = 0;
    ThirdPartyAudio_t *pDecoder = (ThirdPartyAudio_t *)pPrivate->pDecoder;

    //TODO: add frame decoding code here
```

```
Example 1)
xxx_decode_header(pDecoder, &pucInput, &iInputLength);
xxx_decode_frame(pDecoder, pOutBuff);
xxx_get_decoded_info(pDecoder, &num_ch, &num_sample_per_ch, &samplerate);
*piUsedBytes = xxx_calc_used_byte(pDecoder);
*piDecodedSamples = num_ch * num_sample_per_ch;
*piOutSampleRate = samplerate;
*piOutChannels = num_ch;

Example 2)
ret = XXX_DecodeFrame(pDecoder, pucInput, iInputLength, pOutBuff,
                    piUsedBytes, piDecodedSamples, piOutSampleRate, piOutChannels);
////////////////////////////////////
return ret;
}
```

Matters that require attention:

piUsedBytes: number of bytes of input data used

Is not the amount remaining.

The input data will be discarded as much as the value of this variable after this function, therefore, if you have the necessary data for the next time decoding in the current input buffer, as much as it should be excluded from the used bytes.

piDecodedSamples: number of PCM samples decoded

Is not the size of PCM.

This is the total number of PCM samples decoded.

Total_samples = samples per channel * total number of channel

3.4.4 ThirdPartyAudio_FlushDecoder

```
OMX_S32 ThirdPartyAudio_FlushDecoder(component_PrivateType* pPrivate)
```

If some action is required after seeking operation, this function can be used.

Input

pPrivate is a component handle

When a seek operation is performed, this function is called just before calling the [ThirdPartyAudio_DecodeFrame](#) function.

Especially, this function is used for below case:

If the internally buffered input or other data in your decoder need to be flushed after the seek operation.

You only have to flush the internal data of your decoder.

If you do not need these operations, this function need not be modified.

Add the code for internal buffer flush into the green colored part.

```
OMX_S32 ThirdPartyAudio_FlushDecoder(component_PrivateType* pPrivate)
{
    ThirdPartyAudio_t *pDecoder = (ThirdPartyAudio_t *)pPrivate->pDecoder;

    //TODO: add flushing code here

    //////////////////////////////////////

    return 0;
}
```

3.4.5 ThirdPartyAudio_CloseDecoder

```
OMX_S32 ThirdPartyAudio_CloseDecoder(component_PrivateType* pPrivate)
```

This function is used to close your decoder instance opened by the [ThirdPartyAudio_OpenDecoder](#) function.

Input

pPrivate is a component handle

The basic code needed has already been implemented.

Add the code for additional close operation into the green colored part if necessary.

```
OMX_S32 ThirdPartyAudio_CloseDecoder(component_PrivateType* pPrivate)
{
    ThirdPartyAudio_t *pDecoder = (ThirdPartyAudio_t *)pPrivate->pDecoder;

    LOGD( "Close ThirdParty Decoder\n");

    if(pDecoder != NULL)
    {
        //TODO: add additional closing code here

        //////////////////////////////////////

        free(pDecoder);
    }

    return 0;
}
```

4 How to modify the makefile

This chapter describes how to modify the makefiles, Android.mk, to build your decoder and the component.

4.1 Using source code

4.1.1 Modify the makefile for your decoder library

If you want to build source code of your decoder under the android make system.

Copy the source code of your decoder to the following folder

`\hardware\telechips\omx\omx_tp_audiodec_component\lib\src`

Rename Android.mk_ to Android.mk in `\hardware\telechips\omx\omx_tp_audiodec_component\lib`

Open the Android.mk file and find the string labeled "TODO:" and modify there like below table.

The library that is made in this step will be used to make your decoder component.

```

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_PRELINK_MODULE := false

# TODO: add your source code files
LOCAL_SRC_FILES := \
    third_party_audio_dec_api.c \
    src/xxx_decoder.c \
    src/xxx_decheader.c \
    src/xxx_decframe.c \
    ←= write source code file name here

# TODO: specify the library name
LOCAL_MODULE := lib3rd_partydec ←= write your decoder library's name

LOCAL_MODULE_TAGS := optional

LOCAL_C_INCLUDES := \
    $(LOCAL_PATH) \
    $(LOCAL_PATH)/../include \
    $(LOCAL_PATH)/src \
    $(OMX_TOP)/omx_audio_interface/include \
    $(CDK_DIR)/cdk \
    $(CDK_DIR)/audio_codec \
    $(CDK_DIR)/container \
    $(OMX_TOP)/omx_include \
    $(OMX_TOP)/omx_base/include \

LOCAL_STATIC_LIBRARIES := \

LOCAL_SHARED_LIBRARIES := \
    libc \
    libutils \
    libcutils \
    liblog \

LOCAL_LDLIBS += \
    -lpthread \
    -ldl

LOCAL_ARM_MODE := arm

# TODO: specify the compile options
LOCAL_CFLAGS := \
    ←= write compile option

LOCAL_LDFLAGS := \

include $(BUILD_STATIC_LIBRARY)

```


**LOCAL_SRC_FILES := **

Write your decoder's source code file name.

LOCAL_MODULE := lib3rd_partydec

Write your decoder library's name

The library's name must have prefix lib, and the extension, .a, must be omitted.

**LOCAL_CFLAGS := **

Write compile options to build your decoder library

4.1.2 Modify the makefile for component

Rename Android_src.mk_ to Android.mk in [\hardware\telechips\omx\omx_tp_audiodec_component](#)
Open the Android.mk file and find the string labeled "TODO:" and modify there like below table.

```
LOCAL_PATH := $(call my-dir)
MY_PATH := $(LOCAL_PATH)
include $(CLEAR_VARS)

include $(LOCAL_PATH)/lib/Android.mk

include $(CLEAR_VARS)

LOCAL_PATH := $(MY_PATH)

LOCAL_PRELINK_MODULE := false

LOCAL_SRC_FILES := \
    src/omx_tp_audiodec_component.c

LOCAL_C_INCLUDES := \
    $(LOCAL_PATH)/include \
    $(OMX_TOP)/omx_audio_interface/include \
    $(CDK_DIR)/cdk \
    $(CDK_DIR)/audio_codec \
    $(CDK_DIR)/container \
    $(OMX_TOP)/omx_include \
    $(OMX_TOP)/omx_base/include \

LOCAL_SHARED_LIBRARIES := \
    libc \
    libutils \
    libcutils \
    liblog \
    libOMX.TCC.base \
    libOMX.TCC.audio

# TODO: write your decoder module name
LOCAL_WHOLE_STATIC_LIBRARIES := \
    lib3rd_partydec

LOCAL_LDLIBS += \
    -lpthread \
    -ldl

LOCAL_ARM_MODE := arm

LOCAL_CFLAGS := \
    $(TCC_OMX_FLAGS) \
    -DSYS_LINUX

# TODO: modify component name
LOCAL_MODULE := libOMX.TCC.xxxdec          <== write local module name
LOCAL_MODULE_TAGS := optional

include $(BUILD_SHARED_LIBRARY)
```

**LOCAL_WHOLE_STATIC_LIBRARIES := **

Write your decoder library's name as you wrote in the 4.1.1 **LOCAL_MODULE** section of above table .

LOCAL_MODULE

Write component's name, which should take the following form.

libOMX.TCC.codecname

Please refer to the `tcc_omxcore.c` file for the list of available **codec name**.

The `tcc_omxcore.c` file is located in the `\hardware\telechips\common\stagefright\omx_core` directory.

In the `tcc_omxcore.c` file, you can find the table named `tComponentName` as below. The name you specified must equal to the name in this table.

```
static char *tComponentName[MAXCOMP][2] = {

    /*video Decoder components */
    { "OMX.TCC.mpeg4dec", "video_decoder.mpeg4" },
    /*FLV1 decoder component*/
    { "OMX.TCC.Google.vpxdec", "google_decoder.vpx" },

    /*video Encoder components */
    { "OMX.TCC.ENC.mpeg4", "video_encoder.mpeg4" },

    /* Audio Decoder components */
    { "OMX.TCC.flacdec", "audio_decoder.flac" },
    { "OMX.TCC.apedec", "audio_decoder.ape" },
    { "OMX.TCC.vorbisdec", "audio_decoder.vorbis" },
    /* terminate the table */
    {NULL, NULL},
};
```

For example, in case of FLAC

LOCAL_MODULE field must be written as below:

```
LOCAL_MODULE := libOMX.TCC.flacdec
```

4.2 Using Pre-built library

You can use the your decoder library pre-made.

Copy pre-made library to the following folder

[\hardware\telechips\omx\omx_tp_audiodec_component\lib](#)

Rename `Android_lib.mk` to `Android.mk` in [\hardware\telechips\omx\omx_tp_audiodec_component](#)

And open the `Android.mk` file find the string labeled "TODO:" and modify there like below table.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_PRELINK_MODULE := false

LOCAL_SRC_FILES := \
    src/omx_tp_audiodec_component.c \
    lib/third_party_audio_dec_api.c \

LOCAL_C_INCLUDES := \
    $(LOCAL_PATH)/include \
    $(OMX_TOP)/omx_audio_interface/include \
    $(CDK_DIR)/cdk \
    $(CDK_DIR)/audio_codec \
    $(CDK_DIR)/container \
    $(OMX_TOP)/omx_include \
    $(OMX_TOP)/omx_base/include \

LOCAL_SHARED_LIBRARIES := \
    libc \
    libutils \
    libcutils \
    liblog \
    libOMX.TCC.base \
    libOMX.TCC.audio

LOCAL_LDLIBS += \
    -lpthread \
    -ldl

LOCAL_ARM_MODE := arm

LOCAL_CFLAGS := \
    $(TCC_OMX_FLAGS) \
    -DSYS_LINUX

# TODO: write the pre-made 3rd-party library name
LOCAL_LDFLAGS := \
    -L$(LOCAL_PATH)/lib \
    -lThirdPartyXXXDEC_Lib      <== write pre-built library name

# TODO: modify component name
LOCAL_MODULE := libOMX.TCC.xxxdec      <== write local module name
LOCAL_MODULE_TAGS := optional

include $(BUILD_SHARED_LIBRARY)
```

```
LOCAL_LDFLAGS := \
    --whole-archive \
    -L$(LOCAL_PATH)/lib \
    -lThirdPartyXXXDEC_Lib
```

Write the pre-made library's name of your decoder.
The library's name must have prefix lib

Example:

libThirdPartyFLACDec.a

when you write the name of the library, the prefix, lib, and the extension, .a, must be omitted like below.

-lThirdPartyFLACDec

LOCAL_MODULE

Write component's name, which should take the following form.

libOMX.TCC.codecname

Please refer to the tcc_omxcore.c file for the list of available **codec name**.

The tcc_omxcore.c file is located in the `hardware\telechips\common\stagefright\omx_core` directory.

In the tcc_omxcore.c file, you can find the table named **tComponentName** as below.

The name you specified must equal to the name in this table.

```
static char *tComponentName[MAXCOMP][2] = {

    /*video Decoder components */
    { "OMX.TCC.mpeg4dec", "video_decoder.mpeg4" },
    /*FLV1 decoder component*/
    { "OMX.TCC.Google.vpxdec", "google_decoder.vpx" },

    /*video Encoder components */
    { "OMX.TCC.ENC.mpeg4", "video_encoder.mpeg4" },

    /* Audio Decoder components */
    { "OMX.TCC.flacdec", "audio_decoder.flac" },
    { "OMX.TCC.apedec", "audio_decoder.ape" },
    { "OMX.TCC.vorbisdec", "audio_decoder.vorbis" },
    /* terminate the table */
    {NULL, NULL},
};
```

For example, in case of FLAC

LOCAL_MODULE field must be written as below:

LOCAL_MODULE := libOMX.TCC.flacdec