|  |  |
| --- | --- |
| Document Number | IA-MPEG-H-3D-Audio-Enc-Mux-Demux-GSG |
| Version | 1.0 |
| Date | June 20, 2022 |
| Ittiam Systems Confidential | |

MPEG-H 3D Audio LC Profile Encoder Multiplexer/Demultiplexer

Getting Started Guide

Ittiam Systems (P) Ltd,

The Consulate, 1 Richmond Road,

Bangalore 560 025, India



Notice

Ittiam Systems reserves the right to make changes to its products or discontinue any of its products or offerings without notice.

Ittiam warrants the performance of its products to the specifications applicable at the time of sale in accordance with Ittiam’s standard warranty.

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | | Date | | Changes |
| 1.0 | June 20, 2022 | | Original version | |

Copyright © 2022, Ittiam Systems (P) Ltd

Contents

[1. Introduction 1](#_Toc109060504)

[1.1 Motivation 1](#_Toc109060505)

[1.2 Scope 1](#_Toc109060506)

[2. Running the Sample Application - Multiplexer 2](#_Toc109060507)

[2.1 Introduction 2](#_Toc109060508)

[2.2 Build instructions 2](#_Toc109060509)

[2.2.1 Using Makefiles 2](#_Toc109060510)

[2.2.2 Using MSVS project files 3](#_Toc109060511)

[2.3 Executable usage: 4](#_Toc109060512)

[2.3.1 Example to convert the MHAS to MHA1 type MP4: 4](#_Toc109060513)

[2.3.2 Example to convert the MHAS to MHM1 type MP4: 4](#_Toc109060514)

[2.4 Parameter file 5](#_Toc109060515)

[3. Running the Sample Application - Demultiplexer 6](#_Toc109060516)

[3.1 Introduction 6](#_Toc109060517)

[3.2 Build instructions 6](#_Toc109060518)

[3.2.1 Using Makefiles 6](#_Toc109060519)

[3.2.2 Using MSVS project files 7](#_Toc109060520)

[3.3 Executable usage: 7](#_Toc109060521)

[3.3.1 Example to convert MP4 to MHAS: 7](#_Toc109060522)

[3.4 Parameter file 8](#_Toc109060523)

# Introduction

## Motivation

The advent of object-based audio and scene-based audio has revolutionized the 3D sound experience. Traditional channel-based coding expects the capture and playback geometric locations to be exactly the same for best reconstruction of the audio scene. On the other hand, object-based audio coding represents discrete sound sources in the audio scene. The positional information of each of the objects in the audio scene is conveyed in the form of metadata. Scene-based audio coding attempts to reconstruct the sound field at the playback end using the coded higher order ambisonics (HOA) information. In case of both object-based and scene-based audio, the rendering algorithm can adapt based on the configuration of the playback end loudspeaker arrangement, thus making the sound experience agnostic to the configuration of speakers at the playback end.

MPEG-H 3D Audio is a coding standard that supports traditional channel-based, object-based and scene-based audio coding. The immersive 3D sound experience created by MPEG-H 3D Audio is significantly enhanced in comparison with the traditional channel-based multichannel programs and playback systems.

The MPEG-H 3D Audio Low Complexity Profile supports all specified formats, including channel-based audio, object-based audio and scene-based audio via higher order ambisonics (HOA). State-of-the-art MPEG-H 3D Audio Low Complexity Profile according to [ISO/IEC 23008-3:2019/AMD 2:2020](https://www.iso.org/standard/79147.html) also support the MPEG-H 3D Audio Baseline Profile, which is a subset of the MPEG-H 3D Audio Low Complexity Profile. The MPEG-H 3D Audio Baseline Profile does not support scene-based audio/HOA.

## Scope

This document discusses the following:

* Running the Multiplexer (**Chapter 2**)
* This chapter gives a complete overview of the procedure to run the sample application provided.
* Running the Demultiplexer (**Chapter 3**)
* This chapter gives a complete overview of the procedure to run the sample application provided.

# Running the Sample Application - Multiplexer

## Introduction

MP4 multiplexer can be used for multiplexing MHAS file as MP4 file of type MHA1 or MHM1.

## Build instructions

### Using Makefiles

#### For x86 build:

Go to folder *libmpeghe/build/makefile\_mux\_demux* and run:

make -f Makefile\_mux clean all ARCH=x86

The executable gets generated as *libmpeghe/build/bin/ia\_mp4\_mhas\_mux\_x86.out*

#### For x86\_64 build:

Go to folder *libmpeghe/build/makefile\_mux\_demux* and run:

make -f Makefile\_mux clean all ARCH=x86\_64

The executable gets generated as *libmpeghe/build/bin/ia\_mp4\_mhas\_mux\_* *x86\_64.out*

#### For armv7 build:

Go to folder *libmpeghe/build/makefile\_mux\_demux* and run:

make -f Makefile\_mux clean all ARCH=armv7

The executable gets generated as *libmpeghe/build/bin/ia\_mp4\_mhas\_mux\_* *armv7.out*

#### For armv8 build:

Go to folder *libmpeghe/build/makefile\_mux\_demux* and run:

make -f Makefile\_mux clean all ARCH=armv8

The executable gets generated as *libmpeghe/build/bin/ia\_mp4\_mhas\_mux\_* *armv8.out*

### Using MSVS project files

Go to folder *libmpeghe/build/msvs* and open the project *ia\_mp4\_mhas\_mux.sln* using Microsoft Visual Studio tool.

Build the project using the Microsoft Visual Studio IDE

The executable gets generated as *ia\_mp4\_mhas\_mux.exe*

## Executable usage:

Executable can be run with command line or with a parameter file.

The ia\_mp4\_mhas\_mux multiplexer can be run for multiple test files through different command lines.The command line usage is as follows:

<executable> -ifile:<inputfile> -ofile:<outputfile> [options]

[options] can be,

[-op\_fmt:<output\_format>]

where

<input\_file> is the input MHAS file name

<outputfile> is the output MP4 file name

<output\_format> is the output format flag which can take value of 2 or 3. Default is 2. 2 - Encodes mhas file to mha1 type mp4 file.

3 - Encodes mhas file to mhm1 type mp4 file.

### Example to convert the MHAS to MHA1 type MP4:

<executable> -ifile:test.mhas -ofile:test.mp4 -op\_fmt:2

### Example to convert the MHAS to MHM1 type MP4:

<executable> -ifile:test.mhas -ofile:test.mp4 -op\_fmt:3

## Parameter file

Parameter file can be used to run for multiple files. The parameter file needs to be named as 'paramfilesimple\_mux.txt'.

The syntax for writing into the parameter file is:

@Start

@Input\_path <path to be appended to all input files>

@Output\_path <path to be appended to all output files>

<command line 1>

<command line 2>

....

@Stop

# Running the Sample Application - Demultiplexer

## Introduction

MP4 de-multiplexer can be used to demultiplex MP4 file of type MHA1 or MHM1 to MHAS.

## Build instructions

### Using Makefiles

#### For x86 build:

Go to folder *libmpeghe/build/makefile\_mux\_demux* and run:

make -f Makefile\_demux clean all ARCH=x86

The executable gets generated as *libmpeghe/build/bin/ia\_mp4\_mhas\_demux\_x86.out*

#### For x86\_64 build:

Go to folder *libmpeghe/build/makefile\_mux\_demux* and run:

make -f Makefile\_demux clean all ARCH=x86\_64

The executable gets generated as *libmpeghe/build/bin/ia\_mp4\_mhas\_demux\_* *x86\_64.out*

#### For armv7 build:

Go to folder *libmpeghe/build/makefile\_mux\_demux* and run:

make -f Makefile\_demux clean all ARCH=armv7

The executable gets generated as *libmpeghe/build/bin/ia\_mp4\_mhas\_demux\_* *armv7.out*

#### For armv8 build:

Go to folder *libmpeghe/build/makefile\_mux\_demux* and run:

make -f Makefile\_demux clean all ARCH=armv8

The executable gets generated as *libmpeghe/build/bin/ia\_mp4\_mhas\_demux\_* *armv8.out*

### Using MSVS project files

Go to folder *libmpeghe/build/msvs* and open the project *ia\_mp4\_mhas\_demux.sln* using Microsoft Visual Studio tool.

Build the project using the Microsoft Visual Studio IDE

The executable gets generated as *ia\_mp4\_mhas\_demux.exe*

## Executable usage:

Executable can be run with command line or with a parameter file.

The ia\_mp4\_mhas\_demux de-multiplexer can be run for multiple test files through different command lines.The command line usage is as follows:

<executable> -ifile:<inputfile> -ofile:<outputfile>

where

<input\_file> is the input MP4 file name

<outputfile> is the output MHAS file name

### Example to convert MP4 to MHAS:

<executable> -ifile:test.mp4 -ofile:test.mhas

## Parameter file

Parameter file can be used to run for multiple files. The parameter file needs to be named as 'paramfilesimple\_demux.txt'.

The syntax for writing into the parameter file is:

@Start

@Input\_path <path to be appended to all input files>

@Output\_path <path to be appended to all output files>

<command line 1>

<command line 2>

....

@Stop