# MHI_dev

Paul Qureshi

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>MHI_dev | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Paul Qureshi | July 31, 2024 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# Chapter 1

# MHI_dev

## 1.1 MHI Developer Guide

* MHI Developers Guide *

2001 Paul Qureshi

MHI is a system for accessing hardware to decode MPEG audio data (including
the popular MP3 format). It provides an abstraction layer so that software
may use a common API to access any kind of hardware a user has.

Introduction Introduction to MHI and MPEG audio

MPEG overview Facts about MPEG audio

Using MHI How to use MHI in your program

Writing decoders How to create your own driver

Authors Contact us about anything

BSD licence This software is free!



## 1.2 Introduction to MHI and MPEG audio

Overview of MHI

---

MHI is a new standard for MPEG audio decoders, both hardware and CPU based.
It was born from a need to find a standard way of using MPEG decoder hardware
in AmigaAMP, which supported internal and external decoders, mpega.library
(68K and PPC), and the Prelude MPEGiT hardware.

At the time, the only common system was the mpeg.device, developed by
Commodore for the $CD^{32}$ FMV device. This system was, however, flawed in many
ways. For a start, the programmer had little control over the audio and it
was hard to tell what the capabilities of a decoder were. There is also the
difficulty involved in writing devices, where as libraries are relatively easy

to create.
Thus, MHI was created. Using MHI an application can use any MPEG audio decoder in a simple, efficient and consistent manner. Support for tone control (bass, treble, mid and prefactor), volume, panning and stereo mixing are also available. Full developer material is also available for those wishing to use MHI or develop MHI applications. AmigaAMP is the first program to support MHI.
MHI was developed by Paul Qureshi, Thomas Whenzel and Dirk Conrad. If you want to have input on this project or ask any questions,
contact us.

## 1.3 MPEG overview

Overview of MPEG audio
---
Note: a full understanding of MPEG audio is not needed to program with MHI. Free free to skip this section if you like.
MPEG audio was designed to compress audio to a fraction of the original storage size while maintaining audio quality as much as possible. It works by using a psycho-acoustic model (i.e. one based on the way the human ear works and the way the brain decodes sounds) to guess at which parts of the audio are most important. Using this model it can discard 90% of the audio information (which is unimportant to the way human beings perceive sound) and compress the rest as much as possible. The results are surprisingly good, with quality well above the requirements of most listeners.
MPEG is actually a standard for video and audio. MPEG comes in the form of streams. A stream can be in the form of a file on disk, a stream of data from a TCP/IP (internet) connection, or any other data storage method. The three most common MPEG streams are MPEG movies, MP3 audio files and streaming MP3 from the internet.
There are currently three MPEG standards in common use: MPEG 1, MPEG 2 and MPEG 4[1]. MPEG 4 is not very common yet. From a simple applications point of view there isn't much difference between these standards when used for audio streams. MHI is capable of dealing with them all (although note that not all MHI decoders will be capable of decoding them all).
MPEG streams are made up of layers. A layer is a collection of one type of data (say, audio data) in a standard format. The most commonly mentioned layer is Layer III, the audio layer used in MP3 (MPEG Layer 3). For MPEG 1 and 2 there are three possible audio layers, 1, 2 and

3. Each uses a different kind of compression, with layer 3 being the most compressed. The layer used in an MPEG stream is only important if you are interested in the parameters of the audio stream, otherwise you can simply ignore it and pass it to MHI for decoding.
MPEG audio streams have certain characteristics. The two main ones are the bitrate and the sampling frequency. The bitrate refers to the number of bits the encoded stream uses per second of decoded data. A common value is 128kb, which gives near CD quality when used with layer 3. The sampling frequency is the number of samples per second of the decoded data. Commonly this is 44100, the same as used for CD audio.
An extension to the standard MPEG stream that is only of use when the stream is stored as a file is the ID header. This header is the last 128 bytes of a stream file and contains information on the track name, artist, album, comment and the genera the audio fits into. It is referred to as an ID3 tag.
[1] Yes, that's right, there is no MPEG 3 :)

## 1.4 Getting Started

Getting started with MHI
---
All the example code in this manual is in C, since that is my favorite language. Some times I leave out error checking, but in real life you should always check the values MHI returns to you.
MHI drivers are actually just standard Amiga shared libraries. They are located in Libs:mhi/. You can use a file requester or similar to pick one. Each driver uses the same API.
To use MHI, you need to buffer your MPEG data. I recommend you use at least three buffers (I use 8 in my example code) and make them at least around 8K each. You should not really go over about 128k per buffer either, although there is no limit built into MHI. Experiment to see what gives you the best results.
MHI communicates with you via signals. Essentially, they are used to tell you when a buffer had been decoded and is now ready to be re-filled.
Using MHI is really quite simple. The overall structure of your code will look something like this:
Open driver library
|

Query available features

|

Allocate a handle

|

Buffer MPEG data

|

Start playing

|

Feed buffers until end of file

|

Wait for stream end

|

Free handle

Lets walk through it. To start with, you open your selected driver library.
It's as easy as:

```
OpenLibrary("libs:mhi/mhiXXX.library")
```

Once that is done, you query the driver for what features it supports. You should always check if the driver supports a feature before you use it. Having said that, you can usually safely ignore things like the MPEG types and layers supported (see here) since you will just get silence if you try to play non supported streams. You can also query things like the driver name, version and author here, for display in your program :)

The next thing to do is allocate a handle. It is important to be aware of when you should do this. Allocating a handle effectively allocates the decoder and all required hardware (such as audio channels, ports etc). It may also start things like interrupts which use CPU time. Therefore, you should only allocate a handle when you are about to play audio, and free it as soon as you are done. Don't just allocate it when your program loads up!

Allocating a handle is easy:

```
mytask = FindTask(0);
mysignal = AllocSignal(-1);
sigmask = 1L << mysignal;
handle = MHIAllocDecoder(mytask, sigmask);
```

Notice that you have to pass a pointer to your task and a signal mask. This is how MHI sends you feedback on the decoding process. Once you have allocated a handle, the decoder is in the default state. Volume is full, there is no tone control, panning is centered etc. The player is 'stopped' (MHIF_STOPPED)

At this point, you should buffer up some MPEG data by passing MHI the

buffers, like this:

MHIQueueBuffer(handle, bufmem, size);

By buffering up data before you start playing, you prevent skipping. Once memory (called a 'buffer') has been added to the queue, you must not touch it. Do not deallocate it, don't write into it, don't even read from it.

Once you are done, you can start playing the audio with MHIPlay(handle);. Now the audio data is being decoded, and the decoder will continue to decoder any buffers you add to the queue as long as it is in 'play' (MHIF_PLAYING) mode.

The main loop of your program should look like this:

```
while (not end of file)
{
tempsigs = Wait(sigmask | SIGBREAKF_CTRL_C);
while (usedbuf = MHIGetEmpty(handle))
{
find empty buffer
load more data into it
add it to the queue again
}
}
```

In other words, you keep looping until the end of your file. When you get a signal from MHI (make sure it is the signal you gave to MHI, and not a CTRL-C or something!) you know that at least one buffer is now 'empty' (i.e. has been completely decoded) and is ready to be re-used. It is possible for more than one buffer to be empty, so you must loop round and keep checking until MHIGetEmpty returns 0. If there is an empty buffer MHIGetEmpty will return the address of that buffer. It is up to you to work out which buffer that address relates to, although if all your buffers are the same size you can simply load more data into that address safely. Once you have loaded more of the MPEG stream into the buffer, you put it back in the queue with MHIQueueBuffer().

Of course, while you are playing the MPEG audio you can use other MHI commands to pause, stop or alter the decoding parameters (such as volume, tone control etc).

Once you come to the end of the file, you must wait for MHI to finish decoding all the data still in the queue. To do this, simply loop around waiting for signals from MHI, and when they arrive check if the player is now out of data, like this:

```
while((MHIGetStatus(handle) == MHIF_PLAYING))
```

{
wait for another signal
}
When the decoder runs out of buffers to decoder, MHIGetStatus() will return MHIF_OUT_OF_DATA. You can then safely free your handle, or call MHIStop() and load your next file. Freeing your handle can be done at any time, and it will automatically remove all buffers from the queue. MHIStop() will also clear the queue, and return all buffers to you. You can then safely FreeMem() the buffer memory, or write into it as you like.

## 1.5 Writing a decoder

If you wish to write an MHI decoder, please contact us. You are free to use the MAS Player Pro decoder as a base. Writing a decoder is not too hard, so don't worry!

## 1.6 Authors and contacts

The Authors
---
MHI was developed jointly by:
Paul Qureshi (paul.qureshi@btinternet.com)
Thomas Whenzel
Dirk Conrad
Feel free to contact us about anything regarding MHI, especially if you want to write an MHI driver.
Our thanks go to Eternity for sending Thomas a MAS Player Pro unit so he could work on MHI support in AmigaAMP. Thanks also to the ASA crew who encouraged Paul to work on MHI, thanks guys! Find them at www.amigasupport.co.uk.

## 1.7 Legal stuff