MusicDNS Developers Guide

Version 1.0

May 11, 2006

Table of Contents

Overview

MusicDNS is a service that provides accurate acoustic song identification and metadata. Using MusicDNS requires using a small piece of code to create an acoustic fingerprint (a 758 character string). This fingerprint can be sent to the MusicDNS server to retrieve a canonical identifier for the song (a Portable Unique IDentifier, or PUID), which will represent the song. If available, you can also retrieve the associated artist name and track title.

A sample PUID looks like this:

```
6afaf934-f2ee-d434-a477-57c804c9a6be
```

The complete system for creating and resolving fingerprints is also called the Open Fingerprint Architecture. This document was created with version 0.9.3 of the SDK.

Getting Started

The source code for generating fingerprints is available at http://www.musicdns.org. The code is available under several licenses – you can choose the one which is most appropriate for your situation. The options are:

(1) Adaptive Public License
(2) GNU Public License (GPL)
(3) A commercial license

Details on the first two licenses are available in the distribution file "`COPYING`" – details on the commercial license are available at http://www.musicip.com.

After downloading and extracting the source tarball/zip, the `/lib` folder will contain the necessary source code (note that for some platforms you may also be able to download binary distributions of the library which you can simply link against). The code has currently been tested on Windows, Mac OS X, and Linux.

There are three components you should understand before beginning the coding process.

1.	Codecs	In order to create a fingerprint, you will need to load the audio data from a music file. The `/examples` Folder includes Public Domain source code to load Wave files, and an example using the open source LAME codec (http://lame.sourceforge.net).
2.	Fingerprint	You will need the first 135 seconds of audio data in order to generate a fingerprint. The code to generate fingerprints is stored inside the `/lib` Directory.
3.	Protocol Layer	Once you have a fingerprint generated locally, you can send it to the MusicDNS server to retrieve available information about that fingerprint. The source code implementing the protocol is Public Domain, and is stored in the `/examples` Folder.

The code below shows the entire process:

```
// First, load the first 135 seconds of audio data.
AudioData *data = loadWave(file);
if (!data) return;

// Generate the fingerprint.  This should only fail if the audio data
// is less than 10 seconds long.
if (data->createPrint()) {
    // Look up information from the MusicDNS server.  This will require
    // a client id.  You can get a free client id at http://www.musicdns.org
    TrackInformation *info = data->getMetadata("..insert client id here…", "1.0",
true);
    if (info) {
        string artist = info->getArtist();  // Artist name, if available
        string track = info->getTrack(); // Name of track, if available
        string puid = info->getPUID(); // PUID, if available
    }
}
// Make sure to delete the AudioData object so memory is not leaked
delete data;
```

Using the Example Programs

The `/examples` Folder includes Public Domain implementations of the codec and protocol layers. In addition, it includes a simple main function to create a standalone test program. If you have a binary distribution, it will include the sample programs which can be executed directly.

If you want to compile the examples yourself, instructions are in the `README` files. Note that you will need to have other packages installed, as detailed in the `README` file, and listed in the specific following sections.

When you run the example program, it will display the returned information, like this:

```
[c:\example]ofaprint “FrodoCPU – Its Time.wav”
Decoding file FrodoCPU – Its Time.wav
 Title: It’s Time
Artist: FrodoCPU
  PUID: 6afaf934-f2ee-d434-a477-57c804c9a6be
```

If you need help getting the example programs to work, visit the community forums at http://www.musicdns.org.

Open Source Community Support

The MusicBrainz [2] community music metadata database project has embraced the Open Fingerprint Architecture for its future music fingerprinting technology. If you are interested in taking a look at how MusicBrainz has integrated libofa into its own tools, take a look at libtunepimp [1] which provides the internals for libofa-based music tagging tools.
For end-users MusicBrainz provides Picard, a libofa-based music identification and tagging tool that lets you clean up the metadata in your music collection with minimal hassle.

A Comprehensive Look at the Different Layers

The rest of this document will explore specific details on the three main components, and will show how you can integrate these components into your projects at a more customized level. If you are comfortable with the interface exposed in the example program, you are not required to read the rest of this document, however, there may be some useful tips.

The Codec Layer

The codec layer is responsible for ensuring the audio data is in a format that can be used by the fingerprint layer. The core task of the codec layer is to instantiate the AudioData class, and pass in the first 135 seconds of acoustic data (you can pass more if you have it though it will be truncated at 135 seconds, but you cannot pass less, unless the song is less than 135 seconds – in this case, you must pass the entire song).

```
AudioData *data = new AudioData();
data->setData(samples, OFA_LITTLE_ENDIAN, bytes/2,  srate, channels == 2 ? 1 : 0,
ms, “wav”);
```

The arguments match those passed to ofa_create_print, which are explained in the next section. The parameters are as follows:

Samples	A buffer of the actual audio samples (2 bytes or 16 bits per sample)
Byte order	The byte order of the bytes in each sample. This may be OFA_LITTLE_ENDIAN or OFA_BIG_ENDIAN
Number of samples	Number of samples being passed
Sample rate	The number of samples per second of audio (given in samples/second)
Stereo	= 1 if audio is in stereo, or 0 otherwise
Duration	The length of the track in milliseconds, regardless of how many samples are passed
Format	Source format of the data, generally the file extension of the original audio (i.e. “mp3”, “wav”, “wma”, etc).

One open source alternative is to use the libtunepimp [1] library available from MusicBrainz [2], which supports MP3, Ogg/Vorbis, FLAC, WAV, MP4, WMA and MPC codecs.
You can test your codecs against the sample file formats available at http://music.predixis.com/downloads/ofasamples/index.html - you should not distribute your application until the files are returning correct results.

The Fingerprint Layer

The fingerprint layer is the core mathematical analysis that summarizes the key acoustic properties for the track being analyzed. This algorithm is covered by U.S. Patent #7,013,301. If you are using the AudioData class, you can generate a fingerprint using the createPrint method, like this:

```
bool success = data->createPrint();
```

If you want to see what the fingerprint looks like, you can get it like this:

```
string print = data->getPrint();
```

This section shows the underlying methods used by createPrint. If you are using the AudioData class, you can skip to the next section (although the description of the parameters can be useful if you are implementing your own codec).
There are only two methods in the core analysis:

ofa_get_version Retrieve the version number of the library.

```
void ofa_get_version(int *major, int *minor, int *rev);
```

ofa_create_print Create an acoustic fingerprint from regular PCM audio data.

```
const char *ofa_create_print(unsigned char* samples, int
byteOrder,
                                                long size, int
sRate, int stereo);
```

Prototypes for these methods are in the file ofa.h, which is available in the Include Folder of the source distribution.

The API has been designed this way in order to accommodate all potential decoders – the fingerprinting algorithm has been designed and tested against a large number of lossy codecs, so you can get a highly accurate identification across different file formats and different bit-rates.

The ofa_create_print function accepts the following parameters:

samples	A buffer of 16-bit samples in interleaved format (if stereo), i.e. L,R,L,R, etc. *This buffer may be modified during processing!* This buffer must contain at least the first 135 seconds of the song (unless the song is less than 135 seconds in which case is must contain the entire song). Songs less than 10 seconds cannot be fingerprinted (the method returns null).
byteOrder	Byte order of the 16-bit samples passed in the first parameter. You may use either of the constants defined in ofa.h: OFA_LITTLE_ENDIAN, or OFA_BIG_ENDIAN. Failure to correctly identify the byte order of the samples will cause an incorrect fingerprint to be generated.
size	The size of the buffer, with regards to the number of **samples**. (In other words, half the number of bytes). A stereo signal will have twice as many samples as a mono signal.
sRate	The sample rate of the acoustic data. This can be an arbitrary rate, as long as it can be expressed as an integer (in samples per second). If this is different from 44100, rate conversion will be performed during preprocessing, which will add significantly to the overhead.
stereo	1 if there are left and right channels stored, 0 if the data is mono

On success, a valid text representation of the fingerprint is returned. The returned buffer will remain valid until the next call to ofa_create_print, which makes this library **not** reentrant. This shortcoming will be fixed in a future release of this library. On failure, null is returned (0).

Dependencies

The fingerprint layer requires FFT libraries to perform some of the signal analysis. The following libraries can be used for this:

Mac OS X	Use the VDSP macro to use the native acceleration framework. This is accomplished by passing "–framework Accelerate" to your compiler/linker.
Windows/Linux	You have two options. **FFTW** You can download FFTW at http://www.fftw.org/ Define the FFTW3 macro to use this library. - or - **Intel's Math Kernel Library** You can download MKL at http://www.intel.com:80/cd/software/products/asmo-na/eng/perflib/mkl/index.htm Define the MKL macro to use this library.

The default build meets licensing requirements for these libraries, as it's available via open source. However, if you build OFA into your own applications, it is your responsibility to make sure you meet the licensing requirements of whichever of the above libraries you choose to use.

The Protocol Layer

The protocol layer takes the generated fingerprint and passes it to the MusicDNS server to retrieve relevant metadata. This is done by sending an HTTP POST with the necessary information, retrieving an XML response, and putting the results into a TrackInformation object. The basic call to the protocol layer looks like this:

```
TrackInformation *info = data->getMetadata("..insert client id here…", "1.0",
true);
```

The first parameter is the client id. You will need to get a client id from http://www.musicdns.org. A client id should be created for each different product or service which accesses the MusicDNS server. The use of the client id is bound by the Terms and Conditions of the MusicDNS service. The second parameter is the version of your product or service. You can use any string here to indicate a version number which is meaningful. Some samples might include “1.0”, “0.0.1 alpha”, “1.2 (build 14”), or “2.0 beta 1”.

The third parameter indicates whether you want to get metadata in addition to the PUID. If this parameter is false, only PUID’s will be returned. If it’s true, the artist and track name will also be returned, if available.

If the query succeeds, you will get a pointer to a TrackInformation object. If the query fails, it will return 0 (null). The TrackInformation is stored inside the AudioData object, so you should NOT delete this pointer, or hold on to it after the associated AudioData object has been destroyed. You can get the relevant information from the TrackInformation object like this:

```
string artist = info->getArtist();  // Artist name, if available
string track = info->getTrack(); // Name of track, if available
string puid = info->getPUID(); // PUID, if available
```

If a field is unavailable, the length of the returned string will be zero.
The complete protocol for interacting with the MusicDNS server is documented in the Open Fingerprint Architecture Protocol Specification, available separately. This may be useful if you want to access the server using a different language than C/C++.

Important Note: The terms of service for using MusicDNS require that you send any existing metadata associated with the file as part of the initial request. This information is used to help verify the accuracy of the internal metadata and helps improve the system for other users. This can be done with the existing framework by setting fields in the TrackInformation object *before* calling getMetadata. Ideally, you may want to do this when you create the AudioData object initially.

Dependencies

This implementation of the protocol uses the open source libraries libcurl (http://curl.haxx.se/) and expat (http://expat.sourceforge.net/). They are used respectively to implement HTTP requests, and to parse the resulting XML responses. Versions of these libraries are also included for convenience in the `/protocol` directory within the Windows distribution, but it is highly recommended that for implementation purposes you obtain current versions of these libraries.

References

[1] http://musicbrainz.org/products/tunepimp/index.html
[2] http://musicbrainz.org