Forensics Crucible

ManTech

Contents

1 History

In the spring of 2009, a ManTech forensics geek was out jogging when he came across an iPod lying in the street. It had been run over by a few cars and was in bad shape. Spotting a challenge, this geek brought the shattered iPod into the lab and threw it down on the table as a lab challenge. One of the big parts of the challenge was getting past Apple's anti-piracy measures. To deter the illegal copying of MP3s, Apple renames all files to random gibberish. When viewing the file system it's impossible to determine what song a given file contains, which makes it harder to casually share music.

We needed some way to quickly and efficiently catalog the music. The solution would (of course) be in Python, but how could we do it?

2 Metadata

By leveraging the metadata embedded in each MP3, we were able to quickly sort and categorize these files. This is what you're going to be doing in the crucible.

3 Your task

You have been given a Windows machine, a copy of Python, the ID3 format specification and an awful lot of MP3s that have totally messed up names.

You will write a function, copy all files, which takes two parameters, src root and dst root. This function will walk through the entire directory tree starting at src root. Whenever it finds a filename ending in .MP3 it will check for ID3 version 2.3 or 2.4 tags. If these tags are present, it will use these tags to construct a proper filename in dst root. For instance, if dst root is C:\music_copy, and you discover your file is a copy of Shriekback's Gunning for the Buddha from the album Big Night Music, the file will be copied to C:\music_copy\Shriekback\Big Night Music \Gunning for the Buddha.mp3.

If album art is present in an MP3, you will copy it to an appropriately named file. E.g., it might be named Gunning for the Buddha.jpg and put in the same directory as the MP3.

4 Hints

- **You've been spammed.** There is far more data in this document and in the ID3 spec than you need to finish the job. Figure out what's necessary and what's not.
- **The files are all correct.** All these files conform to the specification. No one is messing with you by giving you broken files.
- **Album art is almost always a JPEG.** You can recognize the beginning by looking for the magic value 0xFF 0xD8.
- **Break up the job.** Don't write this as all one giant blob of Python. You really need to write functions. Let each function do a small, distinct part of the problem.

- Pay attention to character encodings. ID3 tags may use either of ISO 8859-1 or utf-16. Your file system is probably Windows-1252. When you read in text data, turn it into a unicode object immediately: it will make it much easier to munge it into a variety of different encodings later. (If Unicode has not been covered in class, we'll set aside 15 minutes for it. It's not hard. It's kind of like a butcher knife: yes, you can cut yourself pretty badly with it, but it doesn't take a whole lot of education to realize the pointy end shouldn't go into your body and you should keep your fingers away from the sharp edge. Unicode is the same way.)
- Some files don’t have an album or artist. Use “unknown” for those.

A The ID3 file format

MP3s contain metadata in the form of ID3 tags. There are several versions in use: the most common is version 2.3. A good 2.3 parser will also handle most of the 2.4 spec.

You have been provided with a copy of the ID3 2.3 specification. That document is canonical: the version given here is just a really quick overview.

A.1 The first ten bytes

The first ten bytes of an ID3 2-tagged file will contain “ID3", a byte representing the major version of the release (e.g., 3 for ID3 2.3, or 4 for ID3 2.4), a byte representing the minor version of the release, one byte containing flags which we don't care about, and four bytes containing a 28-bit unsigned integer representing the size of the ID3 header. [*Only seven bits are used in each of the four bytes, making 28 total bits. The most significant bit is always set to zero, to prevent introducing false synchronization signals.*]

The header size does not include the first ten bytes: it represents the size of the remaining header, not the entire header.

A.2 Tags

Once you've read in the rest of the ID3 header, it's time to begin parsing out the tags. Each tag will begin with four bytes that are uppercase alphanumeric, followed by four bytes containing a 32-bit integer representing the size of the data packet, and two bytes containing flags we don't care about. The data packet begins immediately after.

A.2.1 Text tags

Any tag beginning with `T' is a text tag. The first byte represents the character set that's used in the rest of the data. If the encoding byte is null, ISO 8859-1 is used. If the encoding byte is 1, utf-16 with a byte order mark (bom) is used. [*Version 2.4 extends this to support a few more character sets, but those are so niche you won't need them for this crucible*.]

A.2.2 Comment tags

The tag COMM represents a comment. The first byte of a COMM data packet represents the encoding [*…which is almost always ISO 8859-1*]. It's followed by three bytes containing an ISO language code (e.g., eng for English), and two strings immediately following. A set of two null bytes is used to separate the two strings.

A.2.3 Album art

APIC tags represent \attached pictures." They begin with a byte to represent the text encoding3, a string representing a mime type, a null byte, a one-byte field representing the kind of picture it is, a string describing the album art, a null byte, and then binary data out to the end of the data packet.

A.2.4 Other tags

The ID3 specification is surprisingly large. Feel free to read it and implement support for as many tags as you want!