beets Documentation

Release 1.0b14

Adrian Sampson

May 13, 2012

CONTENTS

Welcome to the documentation for beets, the media library management system for obsessive-compulsive music geeks.

If you're new to beets, begin with the *Getting Started* guide. That guide walks you through installing beets, setting it up how you like it, and starting to build your music library.

Then you can get a more detailed look at beets' features in the *Command-Line Interface* and *.beetsconfig* references. You might also be interested in exploring the *Plugins Included With Beets*.

If you still need help, your can drop by the `#beets` IRC channel on Freenode, email the author, or file a bug in the issue tracker. Please let me know where you think this documentation can be improved.

CHAPTER ONE

CONTENTS

1.1 Guides

This section contains a couple of walkthroughs that will help you get familiar with beets. If you're new to beets, you'll want to begin with the *Getting Started* guide.

1.1.1 Getting Started

Welcome to beets! This guide will help you begin using it to make your music collection better.

Installing

You will need Python. (Beets is written for Python 2.7, but it works with 2.6 as well. Python 3.x is not yet supported.)

- **Mac OS X** v10.7 (Lion) includes Python 2.7 out of the box; Snow Leopard ships with Python 2.6.
- On **Debian or Ubuntu**, depending on the version, beets is available as an official package (Debian details, Ubuntu details), so try typing: `apt-get install beets`. To build from source, you can get everything you need by running: `apt-get install python-dev python-setuptools python-pip`
- For **Arch Linux**, try getting beets from AUR. (There's also a bleeding-edge dev package, which will probably set your computer on fire.)
- For **Gentoo Linux**, there are a couple of overlays that include beets. One by vh4x0r includes a development ebuild and one by syranez includes the released version. Eventually, beets may eventually get added to the official Portage tree.

If you have pip, just say `pip install beets` (you might need `sudo` in front of that). On Arch, you'll need to use `pip2` instead of `pip`.

To install without pip, head over to the Downloads area, download the most recent source distribution, and run `python setup.py install` in the directory therein.

The best way to upgrade beets to a new version is by running `pip install -U beets`. You may want to follow @b33ts on Twitter to hear about progress on new versions.

Installing on Windows

Installing beets on Windows can be tricky. Following these steps might help you get it right:

1. If you don't have it, install Python (you want Python 2.7).

2. Install Setuptools from PyPI. To do this, scroll to the bottom of that page and download the Windows installer (`.exe`, not `.egg`) for your Python version (for example: `setuptools-0.6c11.win32-py2.7.exe`).
3. If you haven't done so already, set your `PATH` environment variable to include Python and its scripts. To do so, you have to get the "Properties" window for "My Computer", then choose the "Advanced" tab, then hit the "Environment Variables" button, and then look for the `PATH` variable in the table. Add the following to the end of the variable's value: `;C:\Python27;C:\Python27\Scripts`.
4. Open a command prompt and install pip by running: `easy_install pip`
5. Now install beets by running: `pip install beets`
6. You're all set! Type `beet` at the command prompt to make sure everything's in order.

Because I don't use Windows myself, I may have missed something. If you have trouble or you have more detail to contribute here, please let me know.

Configuring

You'll want to set a few basic options before you start using beets. The configuration is stored in a text file: on Unix-like OSes, the config file is at `~/.beetsconfig`; on Windows, it's at `%APPDATA%\beetsconfig.ini`. Create and edit the appropriate file with your favorite text editor. This file will start out empty, but here's good place to start:

```
[beets]
directory: ~/music
library: ~/data/musiclibrary.blb
```

Change that first path to a directory where you'd like to keep your music. Then, for `library`, choose a good place to keep a database file that keeps an index of your music.

The default configuration assumes you want to start a new organized music folder (that `directory` above) and that you'll *copy* cleaned-up music into that empty folder using beets' `import` command (see below). But you can configure beets to behave many other ways:

- Start with a new empty directory, but *move* new music in instead of copying it (saving disk space). Put `import_move: yes` in your config file.
- Keep your current directory structure; importing should never move or copy files but instead just correct the tags on music. Put the line `import_copy: no` in your config file to disable any copying or renaming. Make sure to point `directory` at the place where your music is currently stored.
- Keep your current directory structure and *do not* correct files' tags: leave files completely unmodified on your disk. (Corrected tags will still be stored in beets' database, and you can use them to do renaming or tag changes later.) Add both `import_copy: no` and `import_write: no` to your config file to disable renaming and tag-writing.

There are approximately six million other configuration options you can set here, including the directory and file naming scheme. See *.beetsconfig* for a full reference.

Importing Your Library

There are two good ways to bring your existing library into beets. You can either: (a) quickly bring all your files with all their current metadata into beets' database, or (b) use beets' highly-refined autotagger to find canonical metadata for every album you import. Option (a) is really fast, but option (b) makes sure all your songs' tags are exactly right from the get-go. The point about speed bears repeating: using the autotagger on a large library can take a very long time, and it's an interactive process. So set aside a good chunk of time if you're going to go that route. (I'm working on improving the autotagger's performance and automation.) For more information on the interactive tagging process, see *Using the Auto-Tagger*.

If you've got time and want to tag all your music right once and for all, do this:

```
$ beet import /path/to/my/music
```

(Note that by default, this command will *copy music into the directory you specified above*. If you want to use your current directory structure, set the `import_copy` config option.) To take the fast, un-autotagged path, just say:

```
$ beet import -A /my/huge/mp3/library
```

Note that you just need to add `-A` for "don't autotag".

Adding More Music

If you've ripped or... otherwise obtained some new music, you can add it with the `beet import` command, the same way you imported your library. Like so:

```
$ beet import ~/some_great_album
```

This will attempt to autotag the new album (interactively) and add it to your library. There are, of course, more options for this command—just type `beet help import` to see what's available.

By default, the `import` command will try to find and download album art for every album it finds. It will store the art in a file called `cover.jpg` alongside the songs. If you don't like that, you can disable it with the `-R` switch or by setting a value in the *configuration file*.

Seeing Your Music

If you want to query your music library, the `beet list` (shortened to `beet ls`) command is for you. You give it a *query string*, which is formatted something like a Google search, and it gives you a list of songs. Thus:

```
$ beet ls the magnetic fields
The Magnetic Fields - Distortion - Three-Way
The Magnetic Fields - Distortion - California Girls
The Magnetic Fields - Distortion - Old Fools
$ beet ls hissing gronlandic
of Montreal - Hissing Fauna, Are You the Destroyer? - Gronlandic Edit
$ beet ls bird
The Knife - The Knife - Bird
The Mae Shi - Terrorbird - Revelation Six
$ beet ls album:bird
The Mae Shi - Terrorbird - Revelation Six
```

As you can see, search terms by default search all attributes of songs. (They're also implicitly joined by ANDs: a track must match *all* criteria in order to match the query.) To narrow a search term to a particular metadata field, just put the field before the term, separated by a : character. So `album:bird` only looks for `bird` in the "album" field of your songs. (Need to know more? *Queries* will answer all your questions.)

The `beet list` command has another useful option worth mentioning, `-a`, which searches for albums instead of songs:

```
$ beet ls -a forever
Bon Iver - For Emma, Forever Ago
Freezepop - Freezepop Forever
```

So handy!

Beets also has a `stats` command, just in case you want to see how much music you have:

```
$ beet stats
Tracks: 13019
Total time: 4.9 weeks
Total size: 71.1 GB
Artists: 548
Albums: 1094
```

Playing Music

Beets is primarily intended as a music organizer, not a player. It's designed to be used in conjunction with other players (consider Decibel or cmus; there's even *a cmus plugin for beets*). However, it does include a simple music player—it doesn't have a ton of features, but it gets the job done.

The player, called BPD, is a clone of an excellent music player called MPD. Like MPD, it runs as a daemon (i.e., without a user interface). Another program, called an MPD client, controls the player and provides the user with an interface. You'll need to enable the BPD plugin before you can use it. Check out *BPD Plugin*.

You can, of course, use the bona fide MPD server with your beets library. MPD is a great player and has more features than BPD. BPD just provides a convenient, built-in player that integrates tightly with your beets database.

Keep Playing

The *Command-Line Interface* page has more detailed description of all of beets' functionality. (Like deleting music! That's important.) Start exploring!

Also, check out *Plugins Included With Beets* as well as *Other Plugins*. The real power of beets is in its extensibility—with plugins, beets can do almost anything for your music collection.

You can always get help using the `beet help` command. The plain `beet help` command lists all the available commands; then, for example, `beet help import` gives more specific help about the `import` command.

Please let me know what you think of beets via email or Twitter.

1.1.2 Using the Auto-Tagger

Beets' automatic metadata correcter is sophisticated but complicated and cryptic. This is a guide to help you through its myriad inputs and options.

An Apology and a Brief Interlude

I would like to sincerely apologize that the autotagger in beets is so fussy. It asks you a *lot* of complicated questions, insecurely asking that you verify nearly every assumption it makes. This means importing and correcting the tags for a large library can be an endless, tedious process. I'm sorry for this.

Maybe it will help to think of it as a tradeoff. By carefully examining every album you own, you get to become more familiar with your library, its extent, its variation, and its quirks. People used to spend hours lovingly sorting and resorting their shelves of LPs. In the iTunes age, many of us toss our music into a heap and forget about it. This is great for some people. But there's value in intimate, complete familiarity with your collection. So instead of a chore, try thinking of correcting tags as quality time with your music collection. That's what I do.

One practical piece of advice: because beets' importer runs in multiple threads, it queues up work in the background while it's waiting for you to respond. So if you find yourself waiting for beets for a few seconds between every question it asks you, try walking away from the computer for a while, making some tea, and coming back. Beets will have a chance to catch up with you and will ask you questions much more quickly.

Back to the guide.

Overview

Beets' tagger is invoked using the `beet import` command. Point it at a directory and it imports the files into your library, tagging them as it goes (unless you pass `--noautotag`, of course). There are several assumptions beets currently makes about the music you import. In time, we'd like to remove all of these limitations.

- Your music should be organized by album into directories. That is, the tagger assumes that each album is in a single directory. These directories can be arbitrarily deep (like `music/2010/hiphop/seattle/freshespresso/glamour`), but any directory with music files in it is interpreted as a separate album. This means that your flat directory of six thousand uncategorized MP3s won't currently be autotaggable. (This will change eventually.)

  There is one exception to this rule: directories that look like separate parts of a *multi-disc album* are tagged together as a single release. This situation is detected by looking at the names of directories. If one directory has sub-directories with, for example, "disc 1" and "disc 2" in their names, they get lumped together as a single album. The marker words for this feature are "part", "volume", "vol.", "disc", and "CD".
- The music may have bad tags, but it's not completely untagged. (This is actually not a hard-and-fast rule: using the *E* option described below, it's entirely possible to search for a release to tag a given album.) This is because beets by default infers tags based on existing metadata. The *Acoustid plugin* extends the autotagger to use acoustic fingerprinting to find information for arbitrary audio. Install that plugin if you're willing to spend a little more CPU power to get tags for unidentified albums.
- Currently MP3, AAC, FLAC, Ogg Vorbis, Monkey's Audio, WavPack, and Musepack files are supported. (Do you use some other format? Let me know!

Now that that's out of the way, let's tag some music.

Options

To import music, just say `beet import MUSICDIR`. There are, of course, a few command-line options you should know:

- `beet import -A`: don't try to autotag anything; just import files (this goes much faster than with autotagging enabled)
- `beet import -W`: when autotagging, don't write new tags to the files themselves (just keep the new metadata in beets' database)
- `beet import -C`: don't copy imported files to your music directory; leave them where they are
- `beet import -R`: don't fetch album art.
- `beet import -l LOGFILE`: write a message to `LOGFILE` every time you skip an album or choose to take its tags "as-is" (see below) or the album is skipped as a duplicate; this lets you come back later and reexamine albums that weren't tagged successfully
- `beet import -q`: quiet mode. Never prompt for input and, instead, conservatively skip any albums that need your opinion. The `-ql` combination is recommended.
- `beet import -t`: timid mode, which is sort of the opposite of "quiet." The importer will ask your permission for everything it does, confirming even very good matches with a prompt.
- `beet import -p`: automatically resume an interrupted import. The importer keeps track of imports that don't finish completely (either due to a crash or because you stop them halfway through) and, by default, prompts you to decide whether to resume them. The `-p` flag automatically says "yes" to this question. Relatedly, `-P` flag automatically says "no."

- `beet import -s`: run in *singleton* mode, tagging individual tracks instead of whole albums at a time. See the "as Tracks" choice below. This means you can use `beet import -AC` to quickly add a bunch of files to your library without doing anything to them.

Similarity

So you import an album into your beets library. It goes like this:

```
$ beet imp witchinghour
Tagging: Ladytron - Witching Hour
(Similarity: 98.4%)
* Last One Standing -> The Last One Standing
* Beauty -> Beauty*2
* White Light Generation -> Whitelightgenerator
* All the Way -> All the Way...
```

Here, beets gives you a preview of the album match it has found. It shows you which track titles will be changed if the match is applied. In this case, beets has found a match and thinks it's a good enough match to proceed without asking your permission. It has reported the *similarity* for the match it's found. Similarity is a measure of how well-matched beets thinks a tagging option is. 100% similarity means a perfect match 0% indicates a truly horrible match.

In this case, beets has proceeded automatically because it found an option with very high similarity (98.4%). But, as you'll notice, if the similarity isn't quite so high, beets will ask you to confirm changes. This is because beets can't be very confident about more dissimilar matches, and you (as a human) are better at making the call than a computer. So it occasionally asks for help.

Choices

When beets needs your input about a match, it says something like this:

```
Tagging: Beirut - Lon Gisland
(Similarity: 94.4%)
* Scenic World (Second Version) -> Scenic World
[A]pply, More candidates, Skip, Use as-is, as Tracks, Enter search, or aBort?
```

When beets asks you this question, it wants you to enter one of the capital letters: A, M, S, U, T, E, or B. That is, you can choose one of the following:

- *A*: Apply the suggested changes shown and move on.
- *M*: Show more options. (See the Candidates section, below.)
- *S*: Skip this album entirely and move on to the next one.
- *U*: Import the album without changing any tags. This is a good option for albums that aren't in the MusicBrainz database, like your friend's operatic faux-goth solo record that's only on two CD-Rs in the universe.
- *T*: Import the directory as *singleton* tracks, not as an album. Choose this if the tracks don't form a real release—you just have one or more loner tracks that aren't a full album. This will temporarily flip the tagger into *singleton* mode, which attempts to match each track individually.
- *E*: Enter an artist and album to use as a search in the database. Use this option if beets hasn't found any good options because the album is mistagged or untagged.
- *B*: Cancel this import task altogether. No further albums will be tagged; beets shuts down immediately. The next time you attempt to import the same directory, though, beets will ask you if you want to resume tagging where you left off.

Note that the option with `[B]rackets` is the default—so if you want to apply the changes, you can just hit return without entering anything.

Candidates

If you choose the M option, or if beets isn't very confident about any of the choices it found, it will present you with a list of choices (called candidates), like so:

```
Finding tags for "Panther - Panther".
Candidates:
1. Panther - Yourself (66.8%)
2. Tav Falco's Panther Burns - Return of the Blue Panther (30.4%)
# selection (default 1), Skip, Use as-is, or Enter search, or aBort?
```

Here, you have many of the same options as before, but you can also enter a number to choose one of the options that beets has found. Don't worry about guessing—beets will show you the proposed changes and ask you to confirm them, just like the earlier example. As the prompt suggests, you can just hit return to select the first candidate.

Duplicates

If beets finds an album or item in your library that seems to be the same as the one you're importing, you may see a prompt like this:

```
This album is already in the library!
[S]kip new, Keep both, Remove old?
```

Beets wants to keep you safe from duplicates, which can be a real pain, so you have three choices in this situation. You can skip importing the new music, choosing to keep the stuff you already have in your library; you can keep both the old and the new music; or you can remove the existing music and choose the new stuff. If you choose that last "trump" option, any duplicates will be removed from your library database—and, if the corresponding files are located inside of your beets library directory, the files themselves will be deleted as well.

If you choose to keep two identically-named albums, beets can avoid storing both in the same directory. See *Album Disambiguation* for details.

Fingerprinting

You may have noticed by now that beets' autotagger works pretty well for most files, but can get confused when files don't have any metadata (or have wildly incorrect metadata). In this case, you need *acoustic fingerprinting*, a technology that identifies songs from the audio itself. With fingerprinting, beets can autotag files that have very bad or missing tags. The *"chroma" plugin*, distributed with beets, uses the Chromaprint open-source fingerprinting technology, but it's disabled by default. That's because it's sort of tricky to install. See the *Chromaprint/Acoustid Plugin* page for a guide to getting it set up.

Album Art

By default, beets will search for cover art for every album you import, placing the cover art in a file named something like `cover.jpg` in the album's folder. (If you want to customize the name or disable album art fetching altogether, see *.beetsconfig* for the `art_filename` and `import_art` settings.) Currently, beets looks for art on Amazon.com and on your local filesystem: if you have an image file called "cover," "front," "art," "album," for "folder" alongside your music, beets will treat it as album art and skip searching any online databases.

Beets will not, by default, embed album art into files' tags. To do that, take a look at the *EmbedArt Plugin*.

Missing Albums?

If you're having trouble tagging a particular album with beets, you might want to check the following possibilities:

- Is the album present in the MusicBrainz database? You can search on their site to make sure it's cataloged there. If not, anyone can edit MusicBrainz—so consider adding the data yourself.
- Beets won't show you possibilities from MusicBrainz that have *fewer* tracks than the current album. In other words, if you have extra tracks that aren't included on the release, that candidate won't be displayed. (The tagger should, on the other hand, show you candidates that have *more* tracks than you do in the case that you're missing some of the album's songs. Beets will warn you when any candidate is a partial match.)

If neither of these situations apply and you're still having trouble tagging something, please file a bug report.

I Hope That Makes Sense

I haven't made the process clear, please drop me an email and I'll try to improve this guide.

1.2 Reference

This section contains reference materials for various parts of beets. To get started with beets as a new user, though, you may want to read the *Getting Started* guide first.

1.2.1 Command-Line Interface

Commands

import

```
beet import [-CWAPRqst] [-l LOGPATH] DIR...
beet import [options] -L QUERY
```

Add music to your library, attempting to get correct tags for it from MusicBrainz.

Point the command at a directory full of music. The directory can be a single album or a directory whose leaf subdirectories are albums (the latter case is true of typical Artist/Album organizations and many people's "downloads" folders). The music will be copied to a configurable directory structure (see below) and added to a library database (see below). The command is interactive and will try to get you to verify MusicBrainz tags that it thinks are suspect. (This means that importing a large amount of music is therefore very tedious right now; this is something we need to work on. Read the *autotagging guide* if you need help.)

- By default, the command copies files your the library directory and updates the ID3 tags on your music. If you'd like to leave your music files untouched, try the `-C` (don't copy) and `-W` (don't write tags) options. You can also disable this behavior by default in the configuration file (below).
- Also, you can disable the autotagging behavior entirely using `-A` (don't autotag) – then your music will be imported with its existing metadata.
- During a long tagging import, it can be useful to keep track of albums that weren't tagged successfully – either because they're not in the MusicBrainz database or because something's wrong with the files. Use the `-l` option to specify a filename to log every time you skip and album or import it "as-is" or an album gets skipped as a duplicate.

- Relatedly, the `-q` (quiet) option can help with large imports by autotagging without ever bothering to ask for user input. Whenever the normal autotagger mode would ask for confirmation, the quiet mode pessimistically skips the album. The quiet mode also disables the tagger's ability to resume interrupted imports.
- Speaking of resuming interrupted imports, the tagger will prompt you if it seems like the last import of the directory was interrupted (by you or by a crash). If you want to skip this prompt, you can say "yes" automatically by providing `-p` or "no" using `-P`. The resuming feature can be disabled by default using a configuration option (see below).
- If you want to import only the *new* stuff from a directory, use the `-i` option to run an *incremental* import. With this flag, beets will keep track of every directory it ever imports and avoid importing them again. This is useful if you have an "incoming" directory that you periodically add things to. (The `-I` flag disables incremental imports.)
- By default, beets will proceed without asking if it finds a very close metadata match. To disable this and have the importer as you every time, use the `-t` (for *timid*) option.
- The importer automatically tries to download album art for each album it finds. To disable or enable this, use the `-r` or `-R` options.
- The importer typically works in a whole-album-at-a-time mode. If you instead want to import individual, non-album tracks, use the *singleton* mode by supplying the `-s` option.

list

```
beet list [-ap] QUERY
```

Queries the database for music.

Want to search for "Gronlandic Edit" by of Montreal? Try `beet list gronlandic`. Maybe you want to see everything released in 2009 with "vegetables" in the title? Try `beet list year:2009 title:vegetables`. (Read more in *Queries*.) You can use the `-a` switch to search for albums instead of individual items.

The `-p` option makes beets print out filenames of matched items, which might be useful for piping into other Unix commands (such as xargs). Similarly, the `-f` option lets you specify a specific format with which to print every album or track. This uses the same template syntax as beets' *path formats*. For example, the command `beet ls -af '$album: $tracktotal' beatles` prints out the number of tracks on each Beatles album. In Unix shells, remember to enclose the template argument in single quotes to avoid environment variable expansion.

remove

```
beet remove [-ad] QUERY
```

Remove music from your library.

This command uses the same *query* syntax as the `list` command. You'll be shown a list of the files that will be removed and asked to confirm. By default, this just removes entries from the library database; it doesn't touch the files on disk. To actually delete the files, use `beet remove -d`.

modify

```
beet modify [-MWay] QUERY FIELD=VALUE...
```

Change the metadata for items or albums in the database.

Supply a *query* matching the things you want to change and a series of `field=value` pairs. For example, `beet modify genius of love artist="Tom Tom Club"` will change the artist for the track "Genius of Love." The `-a` switch operates on albums instead of individual tracks. Items will automatically be moved around when necessary if they're in your library directory, but you can disable that with `-M`. Tags will be written to the files according to the settings you have for imports, but these can be overridden with `-w` (write tags, the default) and `-W` (don't write tags). Finally, this command politely asks for your permission before making any changes, but you can skip that prompt with the `-y` switch.

move

```
beet move [-ca] [-d DIR] QUERY
```

Move or copy items in your library.

This command, by default, acts as a library consolidator: items matching the query are renamed into your library directory structure. By specifying a destination directory with `-d` manually, you can move items matching a query anywhere in your filesystem. The `-c` option copies files instead of moving them. As with other commands, the `-a` option matches albums instead of items.

update

```
beet update [-aM] QUERY
```

Update the library (and, optionally, move files) to reflect out-of-band metadata changes and file deletions.

This will scan all the matched files and read their tags, populating the database with the new values. By default, files will be renamed according to their new metadata; disable this with `-M`.

To perform a "dry run" an update, just use the `-p` (for "pretend") flag. This will show you all the proposed changes but won't actually change anything on disk.

stats

```
beet stats [QUERY]
```

Show some statistics on your entire library (if you don't provide a *query* or the matched items (if you do).

fields

```
beet fields
```

Show the item and album metadata fields available for use in *Queries* and *Path Formats*.

Global Flags

Beets has a few "global" flags that affect all commands. These must appear between the executable name (`beet`) and the command: for example, `beet -v import ...`.

- `-l LIBPATH`: specify the library database file to use.
- `-d DIRECTORY`: specify the library root directory.
- `-v`: verbose mode; prints out a deluge of debugging information. Please use this flag when reporting bugs.

1.2.2 .beetsconfig

The `beet` command reads configuration information from `~/.beetsconfig` on Unix-like OSes (inluding Mac OS X) and `%APPDATA%\beetsconfig.ini` on Windows. The file is in INI format.

Options

These options are available, all of which must appear under the `[beets]` section header:

`library` Path to the beets library file. Defaults to `~/.beetsmusic.blb` on Unix and `%APPDATA\beetsmusic.blb` on Windows.

`directory` The directory to which files will be copied/moved when adding them to the library. Defaults to `~/Music`.

`import_write` Either `yes` or `no`, controlling whether metadata (e.g., ID3) tags are written to files when using `beet import`. Defaults to `yes`. The `-w` and `-W` command-line options override this setting.

`import_copy` Either `yes` or `no`, indicating whether to **copy** files into the library directory when using `beet import`. Defaults to `yes`. Can be overridden with the `-c` and `-C` command-line options.

The option is ignored if `import_move` is enabled (i.e., beets can move or copy files but it doesn't make sense to do both).

`import_move` Either `yes` or `no`, indicating whether to **move** files into the library directory when using `beet import`. Defaults to `no`.

The effect is similar to the `import_copy` option but you end up with only one copy of the imported file. ("Moving" works even across filesystems; if necessary, beets will copy and then delete when a simple rename is impossible.) Moving files can be risky—it's a good idea to keep a backup in case beets doesn't do what you expect with your files.

This option *overrides* `import_copy`, so enabling it will always move (and not copy) files. The `-c` switch to the `beet import` command, however, still takes precedence.

`import_resume` Either `yes`, `no`, or `ask`. Controls whether interrupted imports should be resumed. "Yes" means that imports are always resumed when possible; "no" means resuming is disabled entirely; "ask" (the default) means that the user should be prompted when resuming is possible. The `-p` and `-P` flags correspond to the "yes" and "no" settings and override this option.

`import_incremental` Either `yes` or `no`, controlling whether imported directories are recorded and whether these recorded directories are skipped. This corresponds to the `-i` flag to `beet import`.

`import_art` Either `yes` or `no`, indicating whether the autotagger should attempt to find and download album cover art for the files it imports. Defaults to `yes`. The `-r` and `-R` command-line options override this setting.

`import_quiet_fallback` Either `skip` (default) or `asis`, specifying what should happen in quiet mode (see the `-q` flag to `import`, above) when there is no strong recommendation.

`import_timid` Either `yes` or `no`, controlling whether the importer runs in *timid* mode, in which it asks for confirmation on every autotagging match, even the ones that seem very close. Defaults to `no`. The `-t` command-line flag controls the same setting.

`import_log` Specifies a filename where the importer's log should be kept. By default, no log is written. This can be overridden with the `-l` flag to `import`.

`ignore` A space-separated list of glob patterns specifying file and directory names to be ignored when importing. Defaults to `.* *~` (i.e., ignore Unix-style hidden files and backup files).

`replace` A set of regular expression/replacement pairs to be applied to all filenames created by beets. Typically, these replacements are used to avoid confusing problems or errors with the filesystem (for example, leading `.`

characters are replaced on Unix and trailing whitespace is removed on Windows). To override these substitutions, specify a sequence of whitespace-separated terms; the first term is a regular expression and the second is a string that should replace anything matching that regex. For example, `replace = [xy] z` will make beets replace all instances of the characters `x` or `y` with the character `z`.

If you do change this value, be certain that you include at least enough substitutions to avoid causing errors on your operating system. Here are the default substitutions used by beets, which are sufficient to avoid unexpected behavior on all popular platforms:

```
replace = [\\/] _
          ^\. _
          [\x00-\x1f] _
          [<>:"\?\*\|] _
          \.$ _
          \s+$ <strip>
```

These substitutions remove forward and back slashes, leading dots, and control characters—all of which is a good idea on any OS. The fourth line removes the Windows "reserved characters" (useful even on Unix for for compatibility with Windows-influenced network filesystems like Samba). Trailing dots and trailing whitespace, which can cause problems on Windows clients, are also removed.

To replace space characters, use the `\s` (whitespace) entity:

```
replace = \s _
          ...
```

This will avoid using a literal space and thus confusing beets. (`\s` also matches tabs and newlines, but that is probably fine.)

To remove characters entirely, use `<strip>` as the replacement. For example, to remove all vowels from your filenames:

```
replace = [aeiou] <strip>
          ...
```

art_filename When importing album art, the name of the file (without extension) where the cover art image should be placed. Defaults to `cover` (i.e., images will be named `cover.jpg` or `cover.png` and placed in the album's directory).

plugins A space-separated list of plugin module names to load. For instance, beets includes the BPD plugin for playing music.

pluginpath A colon-separated list of directories to search for plugins. These paths are just added to `sys.path` before the plugins are loaded. The plugins still have to be contained in a `beetsplug` namespace package.

threaded Either `yes` or `no`, indicating whether the autotagger should use multiple threads. This makes things faster but may behave strangely. Defaults to `yes`.

color Either `yes` or `no`; whether to use color in console output (currently only in the `import` command). Turn this off if your terminal doesn't support ANSI colors.

timeout The amount of time that the SQLite library should wait before raising an exception when the database lock is contended. This should almost never need to be changed except on very slow systems. Defaults to 5.0 (5 seconds).

import_delete Either `yes` or `no`. When enabled in conjunction with `import_copy`, deletes original files after they are copied into your library. Has no effect if the importer is in `import_move` mode or "leave files in place" mode. Defaults to `no`.

This option is historical and deprecated: it's almost always more appropriate to use `import_move` instead.

Path Format Configuration

You can also configure the directory hierarchy beets uses to store music. These settings appear under the `[paths]` section (rather than the main `[beets]` section we used above). Each string is a template string that can refer to metadata fields like `$artist` or `$title`. The filename extension is added automatically. At the moment, you can specify three special paths: `default` for most releases, `comp` for "various artist" releases with no dominant artist, and `singleton` for non-album tracks. The defaults look like this:

```
[paths]
default: $albumartist/$album%aunique{}/$track $title
singleton: Non-Album/$artist/$title
comp: Compilations/$album%aunique{}/$track $title
```

Note the use of `$albumartist` instead of `$artist`; this ensure that albums will be well-organized. For more about these format strings, see *Path Formats*. The `aunique{}` function ensures that identically-named albums are placed in different directories; see *Album Disambiguation* for details.

In addition to `default`, `comp`, and `singleton`, you can condition path queries based on beets queries (see *Queries*). There's one catch: because the `:` character is reserved for separating the query from the template string, the _ character is substituted for `:` in these queries. This means that a config file like this:

```
[paths]
albumtype_soundtrack: Soundtracks/$album/$track $title
```

will place soundtrack albums in a separate directory. The queries are tested in the order they appear in the configuration file, meaning that if an item matches multiple queries, beets will use the path format for the *first* matching query.

Note that the special `singleton` and `comp` path format conditions are, in fact, just shorthand for the explicit queries `singleton_true` and `comp_true`. In contrast, `default` is special and has no query equivalent: the `default` format is only used if no queries match.

Example

Here's an example file:

```
[beets]
library: /var/music.blb
directory: /var/mp3
path_format: $genre/$artist/$album/$track $title
import_copy: yes
import_write: yes
import_resume: ask
import_art: yes
import_quiet_fallback: skip
import_timid: no
import_log: beetslog.txt
ignore: .AppleDouble ._* *~ .DS_Store
art_filename: albumart
plugins: bpd
pluginpath: ~/beets/myplugins
threaded: yes
color: yes

[paths]
default: $genre/$albumartist/$album/$track $title
singleton: Singletons/$artist - $title
comp: $genre/$album/$track $title
albumtype_soundtrack: Soundtracks/$album/$track $title
```

```
[bpd]
host: 127.0.0.1
port: 6600
password: seekrit
```

(That `[bpd]` section configures the optional *BPD* plugin.)

Location

The configuration file is typically located at `$HOME/.beetsconfig`. If you want to store your `.beetsconfig` file somewhere else for whatever reason, you can specify its path by setting the `BEETSCONFIG` environment variable.

1.2.3 Path Formats

The `[paths]` section of the config file (see *.beetsconfig*) lets you specify the directory and file naming scheme for your music library. Templates substitute symbols like `$title` (any field value prefixed by `$`) with the appropriate value from the track's metadata. Beets adds the filename extension automatically.

For example, consider this path format string: `$albumartist/$album/$track $title`

Here are some paths this format will generate:

- `Yeah Yeah Yeahs/It's Blitz!/01 Zero.mp3`
- `Spank Rock/YoYoYoYoYo/11 Competition.mp3`
- `The Magnetic Fields/Realism/01 You Must Be Out of Your Mind.mp3`

Because `$` is used to delineate a field reference, you can use `$$` to emit a dollars sign. As with Python template strings, `${title}` is equivalent to `$title`; you can use this if you need to separate a field name from the text that follows it.

A Note About Artists

Note that in path formats, you almost certainly want to use `$albumartist` and not `$artist`. The latter refers to the "track artist" when it is present, which means that albums that have tracks from different artists on them (like Stop Making Sense, for example) will be placed into different folders! Continuing with the Stop Making Sense example, you'll end up with most of the tracks in a "Talking Heads" directory and one in a "Tom Tom Club" directory. You probably don't want that! So use `$albumartist`.

As a convenience, however, beets allows `$albumartist` to fall back to the value for `$artist` and vice-versa if one tag is present but the other is not.

Functions

Beets path formats also support *function calls*, which can be used to transform text and perform logical manipulations. The syntax for function calls is like this: `%func{arg,arg}`. For example, the `upper` function makes its argument upper-case, so `%upper{beets rocks}` will be replaced with `BEETS ROCKS`. You can, of course, nest function calls and place variable references in function arguments, so `%upper{$artist}` becomes the upper-case version of the track's artists.

These functions are built in to beets:

- `%lower{text}`: Convert `text` to lowercase.
- `%upper{text}`: Convert `text` to UPPERCASE.

- `%title{text}`: Convert `text` to Title Case.
- `%left{text,n}`: Return the first `n` characters of `text`.
- `%right{text,n}`: Return the last `n` characters of `text`.
- `%if{condition,text}` or `%if{condition,truetext,falsetext}`: If `condition` is nonempty (or nonzero, if it's a number), then returns the second argument. Otherwise, returns the third argument if specified (or nothing if `falsetext` is left off).
- `%asciify{text}`: Convert non-ASCII characters to their ASCII equivalents. For example, "café" becomes "cafe". Uses the mapping provided by the unidecode module.
- `%aunique{identifiers,disambiguators}`: Provides a unique string to disambiguate similar albums in the database. See *Album Disambiguation*, below.

Plugins can extend beets with more template functions (see *Writing Plugins*).

Album Disambiguation

Occasionally, bands release two albums with the same name (c.f. Crystal Castles, Weezer, and any situation where a single has the same name as an album or EP). Beets ships with special support, in the form of the `%aunique{}` template function, to avoid placing two identically-named albums in the same directory on disk.

The `aunique` function detects situations where two albums have some identical fields and emits text from additional fields to disambiguate the albums. For example, if you have both Crystal Castles albums in your library, `%aunique{}` will expand to "[2008]" for one album and "[2010]" for the other. The function detects that you have two albums with the same artist and title but that they have different release years.

For full flexibility, the `%auniqe` function takes two arguments, each of which are whitespace-separated lists of album field names: a set of *identifiers* and a set of *disambiguators*. Any group of albums with identical values for all the identifiers will be considered "duplicates". Then, the function tries each disambiguator field, looking for one that distinguishes each of the duplicate albums from each other. The first such field is used as the result for `%aunique`. If no field suffices, an arbitrary number is used to distinguish the two albums.

The default identifiers are `albumartist album` and the default disambiguators are `albumtype year label catalognum albumdisambig`. So you can get reasonable disambiguation behavior if you just use `%aunique{}` with no parameters in your path forms (as in the default path formats), but you can customize the disambiguation if, for example, you include the year by default in path formats.

One caveat: When you import an album that is named identically to one already in your library, the *first* album—the one already in your library— will not consider itself a duplicate at import time. This means that `%aunique{}` will expand to nothing for this album and no disambiguation string will be used at its import time. Only the second album will receive a disambiguation string. If you want to add the disambiguation string to both albums, just run `beet move` (possibly restricted by a query) to update the paths for the albums.

Syntax Details

The characters `$`, `%`, `{`, `}`, and `,` are "special" in the path template syntax. This means that, for example, if you want a `%` character to appear in your paths, you'll need to be careful that you don't accidentally write a function call. To escape any of these characters (except `{`), prefix it with a `$`. For example, `$$` becomes `$`; `$%` becomes `%`, etc. The only exception is `${`, which is ambiguous with the variable reference syntax (like `${title}`). To insert a `{` alone, it's always sufficient to just type `{`.

If a value or function is undefined, the syntax is simply left unreplaced. For example, if you write `$foo` in a path template, this will yield `$foo` in the resulting paths because "foo" is not a valid field name. The same is true of syntax errors like unclosed `{}` pairs; if you ever see template syntax constructs leaking into your paths, check your template for errors.

If an error occurs in the Python code that implements a function, the function call will be expanded to a string that describes the exception so you can debug your template. For example, the second parameter to `%left` must be an integer; if you write `%left{foo,bar}`, this will be expanded to something like `<ValueError: invalid literal for int()>`.

Available Values

Here's a list of the different values available to path formats. The current list can be found definitively by running the command `beet fields`. Note that plugins can add new (or replace existing) template values (see *Writing Plugins*).

Ordinary metadata:

- title
- artist
- artist_sort
- album
- albumartist
- albumartist_sort
- genre
- composer
- grouping
- year
- month
- day
- track
- tracktotal
- disc
- disctotal
- lyrics
- comments
- bpm
- comp
- albumtype (the MusicBrainz album type; the MusicBrainz wiki has a list of type names)
- label
- asin
- catalognum
- script
- language
- country
- albumstatus

- media
- albumdisambig
- disctitle
- encoder

Audio information:

- length (in seconds)
- bitrate (in kilobits per second, with units: e.g., “192kbps”)
- format (e.g., “MP3” or “FLAC”)
- channels
- bitdepth (only available for some formats)
- samplerate (in kilohertz, with units: e.g., “48kHz”)

MusicBrainz and fingerprint information:

- mb_trackid
- mb_albumid
- mb_artistid
- mb_albumartistid
- mb_releasegroupid
- acoustid_fingerprint
- acoustid_id

1.2.4 Queries

Many of beets’ *commands* are built around **query strings:** searches that select tracks and albums from your library. This page explains the query string syntax, which is meant to vaguely resemble the syntax used by Web search engines.

Keyword

This command:

```
$ beet list love
```

will show all tracks matching the query string `love`. Any unadorned word like this matches *anywhere* in a track’s metadata, so you’ll see all the tracks with “love” in their title, in their album name, in the artist, and so on.

For example, this is what I might see when I run the command above:

```
Against Me! - Reinventing Axl Rose - I Still Love You Julie
Air - Love 2 - Do the Joy
Bag Raiders - Turbo Love - Shooting Stars
Bat for Lashes - Two Suns - Good Love
...
```

Combining Keywords

Multiple keywords are implicitly joined with a Boolean "and." That is, if a query has two keywords, it only matches tracks that contain *both* keywords. For example, this command:

```
$ beet ls magnetic tomorrow
```

matches songs from the album "The House of Tomorrow" by The Magnetic Fields in my library. It *doesn't* match other songs by the Magnetic Fields, nor does it match "Tomorrowland" by Walter Meego—those songs only have *one* of the two keywords I specified.

Specific Fields

Sometimes, a broad keyword match isn't enough. Beets supports a syntax that lets you query a specific field—only the artist, only the track title, and so on. Just say `field:value`, where `field` is the name of the thing you're trying to match (such as `artist`, `album`, or `title`) and `value` is the keyword you're searching for.

For example, while this query:

```
$ beet list dream
```

matches a lot of songs in my library, this more-specific query:

```
$ beet list artist:dream
```

only matches songs by the artist The-Dream. One query I especially appreciate is one that matches albums by year:

```
$ beet list -a year:2012
```

Recall that `-a` makes the `list` command show albums instead of individual tracks, so this command shows me all the releases I have from this year.

Phrases

You can query for strings with spaces in them by quoting or escaping them using your shell's argument syntax. For example, this command:

```
$ beet list the rebel
```

shows several tracks in my library, but these (equivalent) commands:

```
$ beet list "the rebel"
$ beet list the\ rebel
```

only match the track "The Rebel" by Buck 65. Note that the quotes and backslashes are not part of beets' syntax; I'm just using the escaping functionality of my shell (bash or zsh, for instance) to pass `the rebel` as a single argument instead of two.

Regular Expressions

While ordinary keywords perform simple substring matches, beets also supports regular expression matching for more advanced queries. To run a regex query, use an additional `:` between the field name and the expression:

```
$ beet list 'artist::Ann(a|ie)'
```

That query finds songs by Anna Calvi and Annie but not Annuals. Similarly, this query prints the path to any file in my library that's missing a track title:

```
$ beet list -p title::^$
```

To search *all* fields using a regular expression, just prefix the expression with a single `:`, like so:

```
$ beet list :Ho[pm]eless
```

Regular expressions are case-sensitive and build on Python's built-in implementation. See Python's documentation for specifics on regex syntax.

Path Queries

Sometimes it's useful to find all the items in your library that are (recursively) inside a certain directory. Use the `path:` field to do this:

```
$ beet list path:/my/music/directory
```

In fact, beets automatically recognizes any query term containing a path separator (`/` on POSIX systems) as a path query, so this command is equivalent:

```
$ beet list /my/music/directory
```

Note that this only matches items that are *already in your library*, so a path query won't necessarily find *all* the audio files in a directory—just the ones you've already added to your beets library.

1.3 Plugins

Plugins can extend beets' core functionality. Plugins can add new commands to the command-line interface, respond to events in beets, augment the autotagger, or provide new path template functions.

1.3.1 Using Plugins

To use a plugin, you have two options:

- Make sure it's in the Python path (known as *sys.path* to developers). This just means the plugin has to be installed on your system (e.g., with a *setup.py* script or a command like *pip* or *easy_install*).
- Set the *pythonpath* config variable to point to the directory containing the plugin. (See *Command-Line Interface*.)

Then, set the *plugins* option in your *~/.beetsconfig* file, like so:

```
[beets]
plugins = mygreatplugin someotherplugin
```

The value for *plugins* should be a space-separated list of plugin module names.

1.3.2 Plugins Included With Beets

There are a few plugins that are included with the beets distribution. They're disabled by default, but you can turn them on as described above.

Chromaprint/Acoustid Plugin

Acoustic fingerprinting is a technique for identifying songs from the way they "sound" rather from their existing metadata. That means that beets' autotagger can theoretically use fingerprinting to tag files that don't have any ID3 information at all (or have completely incorrect data). This plugin uses an open-source fingerprinting technology called Chromaprint and its associated Web service, called Acoustid.

Turning on fingerprinting can increase the accuracy of the autotagger—especially on files with very poor metadata—but it comes at a cost. First, it can be trickier to set up than beets itself (you need to set up the native fingerprinting library, whereas all of the beets core is written in pure Python). Also, fingerprinting takes significantly more CPU and memory than ordinary tagging—which means that imports will go substantially slower.

If you're willing to pay the performance cost for fingerprinting, read on!

Installing Dependencies

To get fingerprinting working, you'll need to install three things: the Chromaprint library or command-line tool, an audio decoder, and the pyacoustid Python library (version 0.6 or later).

First, you will need to install Chromaprint, either as a dynamic library or in the form of a command-line tool (`fpcalc`). The Chromaprint site has links to packages for major Linux distributions. On Mac OS X and Windows, download the appropriate binary package and place the `fpcalc` (or `fpcalc.exe`) on your shell search path (e.g., in `/usr/local/bin` on Mac OS X or `C:\\Program Files` on Windows).

Next, you will need a mechanism for decoding audio files supported by the audioread library. Mac OS X has a number of decoders already built into Core Audio; on Linux, you can install GStreamer for Python, FFmpeg, or MAD and pymad. (Let me know if you have a good source for installing a decoder on Windows.) How you install these will depend on your distribution. For example:

- On Ubuntu, run `apt-get install python-gst0.10-dev`.
- On Arch Linux, you want `pacman -S gstreamer0.10-python`.

To decode audio formats (MP3, FLAC, etc.) with GStreamer, you'll need the standard set of Gstreamer plugins. For example, on Ubuntu, install the packages `gstreamer0.10-plugins-good`, `gstreamer0.10-plugins-bad`, and `gstreamer0.10-plugins-ugly`.

Then, install pyacoustid itself. You can do this using pip, like so:

```
$ pip install pyacoustid
```

Using

Once you have all the dependencies sorted out, you can enable fingerprinting by editing your *.beetsconfig*. Put `chroma` on your `plugins:` line. Your config file should contain something like this:

```
[beets]
plugins: chroma
```

With that, beets will use fingerprinting the next time you run `beet import`.

Submitting Fingerprints You can help expand the Acoustid database by submitting fingerprints for the music in your collection. To do this, first get an API key from the Acoustid service. Just use an OpenID or MusicBrainz account to log in and you'll get a short token string. Then, add the key to your *.beetsconfig* as the value `apikey` in a section called `acoustid` like so:

```
[acoustid]
apikey=AbCd1234
```

Then, run `beet submit`. (You can also provide a query to submit a subset of your library.) The command will use stored fingerprints if they're available; otherwise it will fingerprint each file before submitting it.

Lyrics Plugin

The `lyrics` plugin fetches and stores song lyrics from databases on the Web. Namely, the current version of the plugin uses Lyric Wiki and Lyrics.com.

Fetch Lyrics During Import

To automatically fetch lyrics for songs you import, just enable the plugin by putting `lyrics` on your config file's `plugins` line (see *Plugins*). When importing new files, beets will now fetch lyrics for files that don't already have them. The lyrics will be stored in the beets database. If the `import_write` config option is on, then the lyrics will also be written to the files' tags.

This behavior can be disabled with the `autofetch` config option (see below).

Fetching Lyrics Manually

The `lyrics` command provided by this plugin fetches lyrics for items that match a query (see *Queries*). For example, `beet lyrics magnetic fields absolutely cuckoo` will get the lyrics for the appropriate Magnetic Fields song, `beet lyrics magnetic fields` will get lyrics for all my tracks by that band, and `beet lyrics` will get lyrics for my entire library. The lyrics will be added to the beets database and, if `import_write` is on, embedded into files' metadata.

The `-p` option to the `lyrics` command makes it print lyrics out to the console so you can view the fetched (or previously-stored) lyrics.

Configuring

The plugin has one configuration option, `autofetch`, which lets you disable automatic lyrics fetching during import. To do so, add this to your `~/.beetsconfig`:

```
[lyrics]
autofetch: no
```

BPD Plugin

BPD is a music player using music from a beets library. It runs as a daemon and implements the MPD protocol, so it's compatible with all the great MPD clients out there. I'm using Theremin, gmpc, Sonata, and Ario successfully.

Dependencies

Before you can use BPD, you'll need the media library called GStreamer (along with its Python bindings) on your system.

- On Mac OS X, you can use MacPorts or Homebrew. For MacPorts, just run `port install py27-gst-python`. For Homebrew, use my auxiliary repository to install: `brew tap sampsyo/py ; brew install gst-python`. (Note that you'll need the Mac OS X Developer Tools in either case.)
- On Linux, it's likely that you already have gst-python. (If not, your distribution almost certainly has a package for it.)
- On Windows, you may want to try GStreamer WinBuilds (cavet emptor: I haven't tried this).

Using and Configuring

BPD is a plugin for beets. It comes with beets, but it's disabled by default. To enable it, you'll need to edit your `.beetsconfig` file and add the line `plugins: bpd`. Like so:

```
[beets]
plugins: bpd
```

Then, you can run BPD by invoking:

```
$ beet bpd
```

Fire up your favorite MPD client to start playing music. The MPD site has a long list of available clients. Here are my favorites:

- Linux: gmpc, Sonata
- Mac: Theremin
- Windows: I don't know. Get in touch if you have a recommendation.
- iPhone/iPod touch: MPoD

One nice thing about MPD's (and thus BPD's) client-server architecture is that the client can just as easily on a different computer from the server as it can be run locally. Control your music from your laptop (or phone!) while it plays on your headless server box. Rad!

To configure the BPD server, add a `[bpd]` section to your `.beetsconfig` file. The configuration values, which are pretty self-explanatory, are `host`, `port`, and `password`. Here's an example:

```
[bpd]
host: 127.0.0.1
port: 6600
password: seekrit
```

Implementation Notes

In the real MPD, the user can browse a music directory as it appears on disk. In beets, we like to abstract away from the directory structure. Therefore, BPD creates a "virtual" directory structure (artist/album/track) to present to clients. This is static for now and cannot be reconfigured like the real on-disk directory structure can. (Note that an obvious solution to this is just string matching on items' destination, but this requires examining the entire library Python-side for every query.)

We don't currently support versioned playlists. Many clients, however, use plchanges instead of playlistinfo to get the current playlist, so plchanges contains a dummy implementation that just calls playlistinfo.

The `stats` command always send zero for `playtime`, which is supposed to indicate the amount of time the server has spent playing music. BPD doesn't currently keep track of this. Also, because database updates aren't yet supported, `db_update` is just the time the server was started.

Unimplemented Commands

These are the commands from the MPD protocol that have not yet been implemented in BPD.

Database:

- update

Saved playlists:

- playlistclear
- playlistdelete
- playlistmove
- playlistadd
- playlistsearch
- listplaylist
- listplaylistinfo
- playlistfind
- rm
- save
- load
- rename

Deprecated:

- playlist
- volume

MPDUpdate Plugin

`mpdupdate` is a very simple plugin for beets that lets you automatically update MPD's index whenever you change your beets library.

To use it, enable it in your `.beetsconfig` by putting `mpdupdate` on your `plugins` line. Your `.beetsconfig` should look like this:

```
[beets]
plugins: mpdupdate
```

Then, you'll probably want to configure the specifics of your MPD server. You can do that using an `[mpdupdate]` section in your `.beetsconfig`, which looks like this:

```
[mpdupdate]
host = localhost
port = 6600
password = seekrit
```

With that all in place, you'll see beets send the "update" command to your MPD server every time you change your beets library.

EmbedArt Plugin

Typically, beets stores album art in a “file on the side”: along with each album, there is a file (named “cover.jpg” by default) that stores the album art. You might want to embed the album art directly into each file’s metadata. While this will take more space than the external-file approach, it is necessary for displaying album art in some media players (iPods, for example).

This plugin was added in beets 1.0b8.

Embedding Art Automatically

To automatically embed discovered album art into imported files, just *enable the plugin*. Art will be embedded after each album is added to the library.

This behavior can be disabled with the `autoembed` config option (see below).

Manually Embedding and Extracting Art

The `embedart` plugin provides a couple of commands for manually managing embedded album art:

- `beet embedart IMAGE QUERY`: given an image file and a query matching an album, embed the image into the metadata of every track on the album.
- `beet extractart [-o FILE] QUERY`: extracts the image from an item matching the query and stores it in a file. You can specify the destination file using the `-o` option, but leave off the extension: it will be chosen automatically. The destination filename defaults to `cover` if it’s not specified.
- `beet clearart QUERY`: removes all embedded images from all items matching the query. (Use with caution!)

Configuring

The plugin has one configuration option, `autoembed`, which lets you disable automatic album art embedding. To do so, add this to your `~/.beetsconfig`:

```
[embedart]
autoembed: no
```

Web Plugin

The `web` plugin is a very basic alternative interface to beets that supplements the CLI. It can’t do much right now, and the interface is a little clunky, but you can use it to query and browse your music and—in browsers that support HTML5 Audio—you can even play music.

While it’s not meant to replace the CLI, a graphical interface has a number of advantages in certain situations. For example, when editing a tag, a natural CLI makes you retype the whole thing—common GUI conventions can be used to just edit the part of the tag you want to change. A graphical interface could also drastically increase the number of people who can use beets.

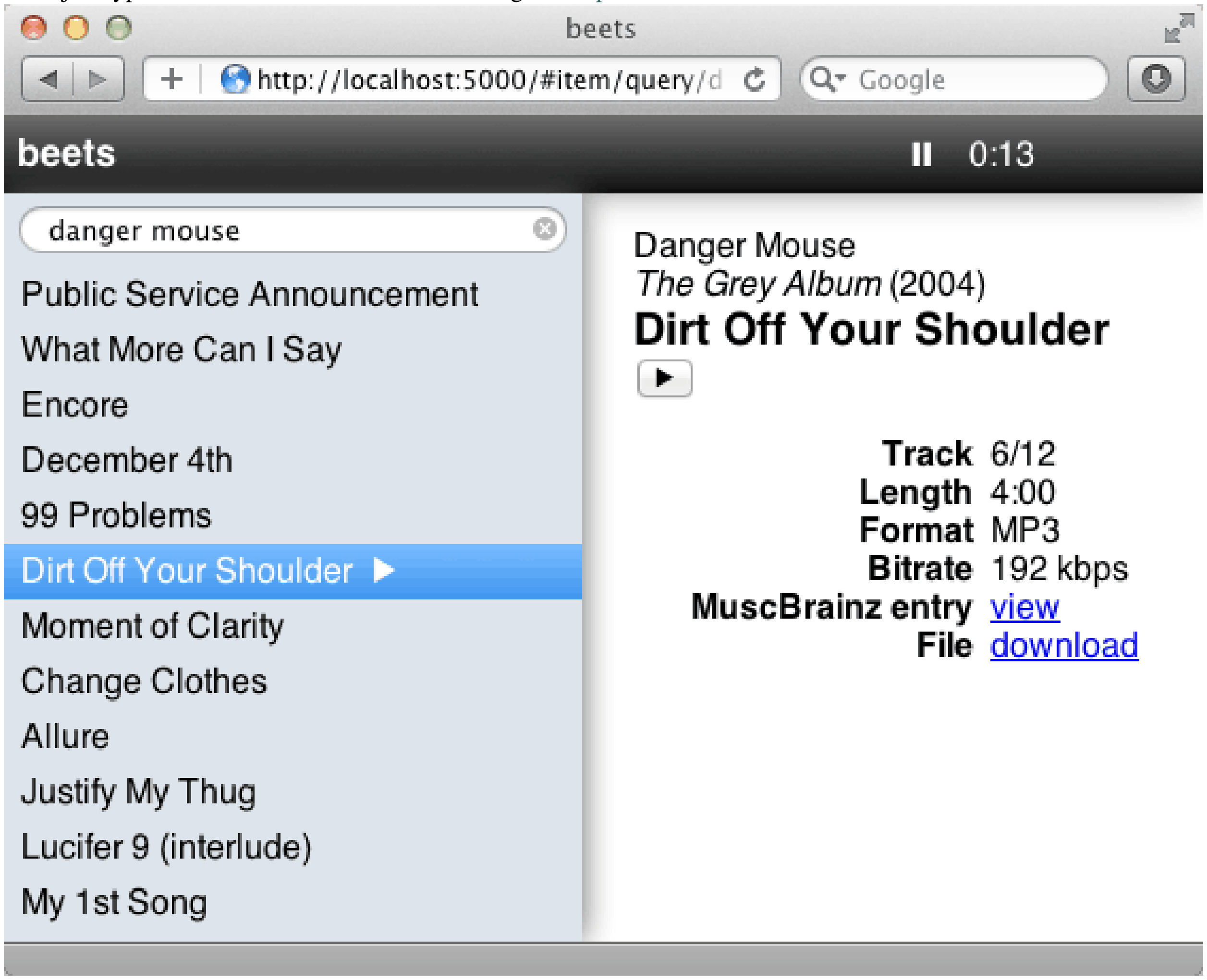
Install

The Web interface depends on Flask. To get it, just run `pip install flask`.

Put `plugins=web` in your `.beetsconfig` to enable the plugin.

Run the Server

Then just type `beet web` to start the server and go to http://localhost:8337/. This is what it looks like:



You can also specify the hostname and port number used by the Web server. These can be specified on the command line or in the `[web]` section of your [Usage#Configuration .beetsconfig].

On the command line, use `beet web [HOSTNAME] [PORT]`. In the config file, use something like this:

```
[web]
host=127.0.0.1
port=8888
```

Usage

Type queries into the little search box. Double-click a track to play it with HTML5 Audio.

Implementation

The Web backend is built using a simple REST+JSON API with the excellent Flask library. The frontend is a single-page application written with Backbone.js. This allows future non-Web clients to use the same backend API.

Eventually, to make the Web player really viable, we should use a Flash fallback for unsupported formats/browsers. There are a number of options for this:

- audio.js
- html5media
- MediaElement.js

LastGenre Plugin

The MusicBrainz database does not contain genre information. Therefore, when importing and autotagging music, beets does not assign a genre. The `lastgenre` plugin fetches *tags* from Last.fm and assigns them as genres to your albums and items. The plugin is included with beets as of version 1.0b11.

The plugin requires pylast, which you can install using pip by typing:

```
pip install pylast
```

After you have pylast installed, enable the plugin by putting `lastgenre` on your `plugins` line in *.beetsconfig*, like so:

```
[beets]
plugins: lastgenre
```

The plugin chooses genres based on a *whitelist*, meaning that only certain tags can be considered genres. This way, tags like “my favorite music” or “seen live” won’t be considered genres. The plugin ships with a fairly extensive internal whitelist, but you can set your own in the config file using the `whitelist` configuration value:

```
[lastgenre]
whitelist: /path/to/genres.txt
```

The genre list file should contain one genre per line. Blank lines are ignored. For the curious, the default genre list is generated by a script that scrapes Wikipedia.

If no genre is found, you have the opportunity to specify a fixed string instead (declare *fallback_str* with no value to blank the genre field):

```
[lastgenre]
fallback_str:
```

Canonicalization

The plugin can also *canonicalize* genres, meaning that more obscure genres can be turned into coarser-grained ones that are present in the whitelist. This works using a tree of nested genre names, represented using YAML, where the leaves of the tree represent the most specific genres.

To enable canonicalization, first install the pyyaml module (`pip install pyyaml`). Then set the `canonical` configuration value:

```
[lastgenre]
canonical:
```

Leaving this value blank will use a built-in canonicalization tree. You can also set it to a path, just like the `whitelist` config value, to use your own tree.

ReplayGain Plugin

This plugin adds support for ReplayGain, a technique for normalizing audio playback levels.

> **Warning:** Some users have reported problems with the Gstreamer ReplayGain calculation plugin. If you experience segmentation faults or random hangs with this plugin enabled, consider disabling it. (Please file a bug if you can get a gdb traceback for such a segfault or hang.)

Installation

This plugin requires GStreamer with the rganalysis plugin (part of gst-plugins-good), gst-python, and the rgain Python module.

First, install GStreamer, its "good" plugins, and the Python bindings if your system doesn't have them already. (The *BPD Plugin* and *Chromaprint/Acoustid Plugin* pages have hints on getting GStreamer stuff installed.) Then install rgain using pip:

```
$ pip install rgain
```

Finally, add `replaygain` to your `plugins` line in your *.beetsconfig*, like so:

```
[beets]
plugins = replaygain
```

Usage & Configuration

The plugin will automatically analyze albums and individual tracks as you import them. It writes tags to each file according to the ReplayGain specification; if your player supports these tags, it can use them to do level adjustment.

By default, files that already have ReplayGain tags will not be re-analyzed. If you want to analyze *every* file on import, you can set the `overwrite` option for the plugin in your *.beetsconfig*, like so:

```
[replaygain]
overwrite: yes
```

Inline Plugin

The `inline` plugin lets you use Python expressions to customize your path formats. Using it, you can define template fields in your beets configuration file and refer to them from your template strings in the `[paths]` section (see *.beetsconfig*).

To use inline field definitions, first enable the plugin by putting `inline` on your `plugins` line, like so:

```
[beets]
plugins: inline
```

Then, make a `[pathfields]` section in your config file. In this section, every line defines a new template field; the key is the name of the field (you'll use the name to refer to the field in your templates) and the value is a Python expression. The expression has all of a track's fields in scope, so you can refer to any normal attributes (such as `artist` or `title`) as Python variables. Here are a couple of examples:

```
[pathfields]
artist_initial: artist[0].upper() + u'.'
```

```
disc_and_track: u'%02i.%02i' % (disc, track) if
                disctotal > 1 else u'%02i' % (track)
```

(Note that the config file's syntax allows newlines in values if the subsequent lines are indented.) These examples define `$artist_initial` and `$disc_and_track` fields that can be referenced in path templates like so:

```
[paths]
default: $artist_initial/$artist/$album/$disc_and_track $title
```

Scrub Plugin

The `scrub` plugin lets you remove extraneous metadata from files' tags. If you'd prefer never to see crufty tags that come from other tools, the plugin can automatically remove all non-beets-tracked tags whenever a file's metadata is written to disk by removing the tag entirely before writing new data. The plugin also provides a command that lets you manually remove files' tags.

Automatic Scrubbing

To automatically remove files' tags before writing new ones, just enable the plugin (see *Plugins*). When importing new files (with `import_write` turned on) or modifying files' tags with the `beet modify` command, beets will first strip the tags entirely and then write the database-tracked metadata to the file.

This behavior can be disabled with the `autoscrub` config option (see below).

Manual Scrubbing

The `scrub` command provided by this plugin removes tags from files and then rewrites their database-tracked metadata. To run it, just type `beet scrub QUERY` where `QUERY` matches the tracks to be scrubbed. Use this command with caution, however, because any information in the tags that is out of sync with the database will be lost.

The `-W` (or `--nowrite`) option causes the command to just remove tags but not restore any information. This will leave the files with no metadata whatsoever.

Configuring

The plugin has one configuration option, `autoscrub`, which lets you disable automatic metadata stripping. To do so, add this to your `~/.beetsconfig`:

```
[scrub]
autoscrub: no
```

Rewrite Plugin

The `rewrite` plugin lets you easily substitute values in your path formats. Specifically, it is intended to let you *canonicalize* names such as artists: for example, perhaps you want albums from The Jimi Hendrix Experience to be sorted into the same folder as solo Hendrix albums.

To use field rewriting, first enable the plugin by putting `rewrite` on your `plugins` line:

```
[beets]
plugins: rewrite
```

Then, make a `[rewrite]` section in your config file to contain your rewrite rules. Each rule consists of a field name, a regular expression pattern, and a replacement value. Rules are written `fieldname regex:  replacement`. For example, this line implements the Jimi Hendrix example above:

```
[rewrite]
artist The Jimi Hendrix Experience: Jimi Hendrix
```

This will make `$artist` in your path formats expand to "Jimi Henrix" where it would otherwise be "The Jimi Hendrix Experience".

The pattern is a case-insensitive regular expression. This means you can use ordinary regular expression syntax to match multiple artists. For example, you might use:

```
[rewrite]
artist .*jimi hendrix.*: Jimi Hendrix
```

As a convenience, the plugin applies patterns for the `artist` field to the `albumartist` field as well. (Otherwise, you would probably want to duplicate every rule for `artist` and `albumartist`.)

Note that this plugin only applies to path templating; it does not modify files' metadata tags or the values tracked by beets' library database.

Random Plugin

The `rdm` plugin provides a command that randomly selects tracks or albums from your library. This can be helpful if you need some help deciding what to listen to.

First, enable the plugin named `rdm` (see *Plugins*). You'll then be able to use the `beet random` command:

```
$ beet random
Aesop Rock - None Shall Pass - The Harbor Is Yours
```

The command has several options that resemble those for the `beet list` command (see *Command-Line Interface*). To choose an album instead of a single track, use `-a`; to print paths to items instead of metadata, use `-p`; and to use a custom format for printing, use `-f FORMAT`.

The `-n NUMBER` option controls the number of objects that are selected and printed (default 1). To select 5 tracks from your library, type `beet random -n5`.

MusicBrainz Collection Plugin

The `mbcollection` plugin lets you submit your catalog to MusicBrainz to maintain your music collection list there.

To begin, just enable the `mbcollection` plugin (see *Plugins*). Then, add your MusicBrainz username and password to your *.beetsconfig* in a `musicbrainz` section:

```
[musicbrainz]
user: USERNAME
pass: PASSWORD
```

Then, use the `beet mbupdate` command to send your albums to MusicBrainz. The command automatically adds all of your albums to the first collection it finds. If you don't have a MusicBrainz collection yet, you may need to add one to your profile first.

ImportFeeds Plugin

The `importfeeds` plugin helps you keep track of newly imported music in your library.

To use the plugin, just put `importfeeds` on the `plugins` line in your *.beetsconfig*:

```
[beets]
plugins: importfeeds
```

The `feeds_dir` parameter can be set to specify another folder than the default library directory. Three different types of outputs coexist, specify the ones you want to use by setting the `feeds_formats` parameter:

- `m3u`: catalog the imports in a centralized playlist. By default, the playlist is named `imported.m3u`. To use a different file, just set the `m3u_name` parameter inside the `importfeeds` config section.
- `m3u_multi`: create a new playlist for each import (uniquely named by appending the date and track/album name).
- `link`: create a symlink for each imported item. This is the recommended setting to propagate beets imports to your iTunes library: just drag and drop the `feeds_dir` folder on the iTunes dock icon.

An example of `importfeeds` configuration:

```
[importfeeds]
feeds_formats: m3u link
feeds_dir: ~/imports/
m3u_name: newfiles.m3u
```

Autotagger Extensions

- *Chromaprint/Acoustid Plugin*: Use acoustic fingerprinting to identify audio files with missing or incorrect metadata.

Metadata

- *Lyrics Plugin*: Automatically fetch song lyrics.
- *LastGenre Plugin*: Fetch genres based on Last.fm tags.
- *EmbedArt Plugin*: Embed album art images into files' metadata. (By default, beets uses image files "on the side" instead of embedding images.)
- *ReplayGain Plugin*: Calculate volume normalization for players that support it.
- *Scrub Plugin*: Clean extraneous metadata from music files.

Path Formats

- *Inline Plugin*: Use Python snippets to customize path format strings.
- *Rewrite Plugin*: Substitute values in path formats.

Interoperability

- *MPDUpdate Plugin*: Automatically notifies MPD whenever the beets library changes.
- *ImportFeeds Plugin*: Keep track of imported files via `.m3u` playlist file(s) or symlinks.

Miscellaneous

- *Web Plugin*: An experimental Web-based GUI for beets.
- *Random Plugin*: Randomly choose albums and tracks from your library.
- *MusicBrainz Collection Plugin*: Maintain your MusicBrainz collection list.
- *BPD Plugin*: A music player for your beets library that emulates MPD and is compatible with MPD clients.

1.3.3 Other Plugins

Here are a few of the plugins written by the beets community:

- beetFs is a FUSE filesystem for browsing the music in your beets library. (Might be out of date.)
- A cmus plugin integrates with the cmus console music player.
- featInTitle moves featured artists from the artist tag to the title tag.

1.3.4 Writing Plugins

If you know a little Python, you can write your own plugin to do almost anything you can imagine with your music collection. See the *guide to writing beets plugins*.

Writing Plugins

A beets plugin is just a Python module inside the `beetsplug` namespace package. (Check out this Stack Overflow question about namespace packages if you haven't heard of them.) So, to make one, create a directory called `beetsplug` and put two files in it: one called `__init__.py` and one called `myawesomeplugin.py` (but don't actually call it that). Your directory structure should look like this:

```
beetsplug/
    __init__.py
    myawesomeplugin.py
```

Then, you'll need to put this stuff in `__init__.py` to make `beetsplug` a namespace package:

```
from pkgutil import extend_path
__path__ = extend_path(__path__, __name__)
```

That's all for `__init__.py`; you can can leave it alone. The meat of your plugin goes in `myawesomeplugin.py`. There, you'll have to import the `beets.plugins` module and define a subclass of the `BeetsPlugin` class found therein. Here's a skeleton of a plugin file:

```
from beets.plugins import BeetsPlugin

class MyPlugin(BeetsPlugin):
    pass
```

Once you have your `BeetsPlugin` subclass, there's a variety of things your plugin can do. (Read on!)

To use your new plugin, make sure your `beetsplug` directory is in the Python path (using `PYTHONPATH` or by installing in a virtualenv, for example). Then, as described above, edit your `.beetsconfig` to include `plugins=myawesomeplugin` (substituting the name of the Python module containing your plugin).

Add Commands to the CLI

Plugins can add new subcommands to the beet command-line interface. Define the plugin class' commands() method to return a list of Subcommand objects. (The Subcommand class is defined in the beets.ui module.) Here's an example plugin that adds a simple command:

```
from beets.plugins import BeetsPlugin
from beets.ui import Subcommand

my_super_command = Subcommand('super', help='do something super')
def say_hi(lib, config, opts, args):
    print "Hello everybody! I'm a plugin!"
my_super_command.func = say_hi

class SuperPlug(BeetsPlugin):
    def commands(self):
        return [my_super_command]
```

To make a subcommand, invoke the constructor like so: Subcommand(name, parser, help, aliases). The name parameter is the only required one and should just be the name of your command. parser can be an OptionParser instance, but it defaults to an empty parser (you can extend it later). help is a description of your command, and aliases is a list of shorthand versions of your command name.

You'll need to add a function to your command by saying mycommand.func = myfunction. This function should take the following parameters: lib (a beets Library object), config (a ConfigParser object containing the configuration values), and opts and args (command-line options and arguments as returned by OptionParser.parse_args).

The function should use any of the utility functions defined in beets.ui. Try running pydoc beets.ui to see what's available.

You can add command-line options to your new command using the parser member of the Subcommand class, which is an OptionParser instance. Just use it like you would a normal OptionParser in an independent script.

Listen for Events

Event handlers allow plugins to run code whenever something happens in beets' operation. For instance, a plugin could write a log message every time an album is successfully autotagged or update MPD's index whenever the database is changed.

You can "listen" for events using the BeetsPlugin.listen decorator. Here's an example:

```
from beets.plugins import BeetsPlugin

class SomePlugin(BeetsPlugin):
    pass

@SomePlugin.listen('pluginload')
def loaded():
    print 'Plugin loaded!'
```

Pass the name of the event in question to the listen decorator. The events currently available are:

- *pluginload*: called after all the plugins have been loaded after the beet command starts
- *import*: called after a beet import command fishes (the lib keyword argument is a Library object; paths is a list of paths (strings) that were imported)

- *album_imported*: called with an `Album` object every time the `import` command finishes adding an album to the library. Parameters: `lib`, `album`, `config`
- *item_imported*: called with an `Item` object every time the importer adds a singleton to the library (not called for full-album imports). Parameters: `lib`, `item`, `config`
- *write*: called with an `Item` and a `MediaFile` object just before a file's metadata is written to disk.
- *import_task_start*: called when before an import task begins processing. Parameters: `task` and `config`.
- *import_task_apply*: called after metadata changes have been applied in an import task. Parameters: `task` and `config`.

The included `mpdupdate` plugin provides an example use case for event listeners.

Extend the Autotagger

Plugins in can also enhance the functionality of the autotagger. For a comprehensive example, try looking at the `chroma` plugin, which is included with beets.

A plugin can extend three parts of the autotagger's process: the track distance function, the album distance function, and the initial MusicBrainz search. The distance functions determine how "good" a match is at the track and album levels; the initial search controls which candidates are presented to the matching algorithm. Plugins implement these extensions by implementing three methods on the plugin class:

- `track_distance(self, item, info)`: adds a component to the distance function (i.e., the similarity metric) for individual tracks. `item` is the track to be matched (and Item object) and `info` is the MusicBrainz track entry that is proposed as a match. Should return a `(dist, dist_max)` pair of floats indicating the distance.
- `album_distance(self, items, info)`: like the above, but compares a list of items (representing an album) to an album-level MusicBrainz entry. Should only consider album-level metadata (e.g., the artist name and album title) and should not duplicate the factors considered by `track_distance`.
- `candidates(self, items)`: given a list of items comprised by an album to be matched, return a list of `AlbumInfo` objects for candidate albums to be compared and matched.
- `item_candidates(self, item)`: given a *singleton* item, return a list of `TrackInfo` objects for candidate tracks to be compared and matched.

When implementing these functions, it will probably be very necessary to use the functions from the `beets.autotag` and `beets.autotag.mb` modules, both of which have somewhat helpful docstrings.

Read Configuration Options

Plugins can configure themselves using the `.beetsconfig` file. Define a `configure` method on your plugin that takes an `OptionParser` object as an argument. Then use the `beets.ui.config_val` convenience function to access values from the config file. Like so:

```
class MyPlugin(BeetsPlugin):
    def configure(self, config):
        number_of_goats = beets.ui.config_val(config, 'myplug', 'goats', '42')
```

Try looking at the `mpdupdate` plugin (included with beets) for an example of real-world use of this API.

Add Path Format Functions and Fields

Beets supports *function calls* in its path format syntax (see *Path Formats*). Beets includes a few built-in functions, but plugins can add new functions using the `template_func` decorator. To use it, decorate a function with `MyPlugin.template_func("name")` where `name` is the name of the function as it should appear in template strings.

Here's an example:

```
class MyPlugin(BeetsPlugin):
    pass
@MyPlugin.template_func('initial')
def _tmpl_initial(text):
    if text:
        return text[0].upper()
    else:
        return u''
```

This plugin provides a function `%initial` to path templates where `%initial{$artist}` expands to the artist's initial (its capitalized first character).

Plugins can also add template *fields*, which are computed values referenced as `$name` in templates. To add a new field, decorate a function taking a single parameter, `item`, with `MyPlugin.template_field("name")`. Here's an example that adds a `$disc_and_track` field:

```
@MyPlugin.template_field('disc_and_track')
def _tmpl_disc_and_track(item):
    """Expand to the disc number and track number if this is a
    multi-disc release. Otherwise, just exapnds to the track
    number.
    """
    if item.disctotal > 1:
        return u'%02i.%02i' % (item.disc, item.track)
    else:
        return u'%02i' % (item.track)
```

With this plugin enabled, templates can reference `$disc_and_track` as they can any standard metadata field.

Extend MediaFile

MediaFile is the file tag abstraction layer that beets uses to make cross-format metadata manipulation simple. Plugins can add fields to MediaFile to extend the kinds of metadata that they can easily manage.

The `item_fields` method on plugins should be overridden to return a dictionary whose keys are field names and whose values are descriptor objects that provide the field in question. The descriptors should probably be `MediaField` instances (defined in `beets.mediafile`). Here's an example plugin that provides a meaningless new field "foo":

```
from beets import mediafile, plugins, ui
class FooPlugin(plugins.BeetsPlugin):
    def item_fields(self):
        return {
            'foo': mediafile.MediaField(
                mp3 = mediafile.StorageStyle(
                    'TXXX', id3_desc=u'Foo Field'),
                mp4 = mediafile.StorageStyle(
                    '----:com.apple.iTunes:Foo Field'),
                etc = mediafile.StorageStyle('FOO FIELD')
            ),
        }
```

Later, the plugin can manipulate this new field by saying something like `mf.foo = 'bar'` where `mf` is a `MediaFile` instance.

Note that, currently, these additional fields are *only* applied to `MediaFile` itself. The beets library database schema and the `Item` class are not extended, so the fields are second-class citizens. This may change eventually.

1.4 Changelog

The centerpiece of this beets release is the graceful handling of similarly-named albums. It's now possible to import two albums with the same artist and title and to keep them from conflicting in the filesystem. Many other awesome new features were contributed by the beets community, including regular expression queries, artist sort names, moving files on import. There are three new plugins: random song/album selection; MusicBrainz "collection" integration; and a plugin for interoperability with other music library systems.

A million thanks to the (growing) beets community for making this a huge release.

1.4.1 1.0b14 (May 12, 2012)

- The importer now gives you **choices when duplicates are detected**. Previously, when beets found an existing album or item in your library matching the metadata on a newly-imported one, it would just skip the new music to avoid introducing duplicates into your library. Now, you have three choices: skip the new music (the previous behavior), keep both, or remove the old music. See the *Duplicates* section in the autotagging guide for details.
- Beets can now avoid storing identically-named albums in the same directory. The new `%aunique{}` template function, which is included in the default path formats, ensures that Crystal Castles' albums will be placed into different directories. See *Album Disambiguation* for details.
- Beets queries can now use **regular expressions**. Use an additional `:` in your query to enable regex matching. See *Regular Expressions* for the full details. Thanks to Matteo Mecucci.
- Artist **sort names** are now fetched from MusicBrainz. There are two new data fields, `artist_sort` and `albumartist_sort`, that contain sortable artist names like "Beatles, The". These fields are also used to sort albums and items when using the `list` command. Thanks to Paul Provost.
- Many other **new metadata fields** were added, including ASIN, label catalog number, disc title, encoder, and MusicBrainz release group ID. For a full list of fields, see *Available Values*.
- *Chromaprint/Acoustid Plugin*: A new command, `beet submit`, will **submit fingerprints** to the Acoustid database. Submitting your library helps increase the coverage and accuracy of Acoustid fingerprinting. The Chromaprint fingerprint and Acoustid ID are also now stored for all fingerprinted tracks. This version of beets *requires* at least version 0.6 of pyacoustid for fingerprinting to work.
- The importer can now **move files**. Previously, beets could only copy files and delete the originals, which is inefficient if the source and destination are on the same filesystem. Use the `import_move` configuration option and see *.beetsconfig* for more details. Thanks to Domen Kožar.
- New *Random Plugin*: Randomly select albums and tracks from your library. Thanks to Philippe Mongeau.
- The *MusicBrainz Collection Plugin* by Jeffrey Aylesworth was added to the core beets distribution.
- New *ImportFeeds Plugin*: Catalog imported files in `m3u` playlist files or as symlinks for easy importing to other systems. Thanks to Fabrice Laporte.

- The `-f` (output format) option to the `beet list` command can now contain template functions as well as field references. Thanks to Steve Dougherty.
- A new command `beet fields` displays the available metadata fields (thanks to Matteo Mecucci).
- The `import` command now has a `--noincremental` or `-I` flag to disable incremental imports (thanks to Matteo Mecucci).
- When the autotagger fails to find a match, it now displays the number of tracks on the album (to help you guess what might be going wrong) and a link to the FAQ.
- The default filename character substitutions were changed to be more conservative. The Windows "reserved characters" are substituted by default even on Unix platforms (this causes less surprise when using Samba shares to store music). To customize your character substitutions, see *the replace config option*.
- *LastGenre Plugin*: Added a "fallback" option when no suitable genre can be found (thanks to Fabrice Laporte).
- *Rewrite Plugin*: Unicode rewriting rules are now allowed (thanks to Nicolas Dietrich).
- Filename collisions are now avoided when moving album art.
- *BPD Plugin*: Print messages to show when directory tree is being constructed.
- *BPD Plugin*: Use Gstreamer's `playbin2` element instead of the deprecated `playbin`.
- *BPD Plugin*: Random and repeat modes are now supported (thanks to Matteo Mecucci).
- *BPD Plugin*: Listings are now sorted (thanks once again to Matteo Mecucci).
- Filenames are normalized with Unicode Normal Form D (NFD) on Mac OS X and NFC on all other platforms.
- Significant internal restructuring to avoid SQLite locking errors. As part of these changes, the not-very-useful "save" plugin event has been removed.

1.4.2 1.0b13 (March 16, 2012)

Beets 1.0b13 consists of a plethora of small but important fixes and refinements. A lyrics plugin is now included with beets; new audio properties are catalogged; the `list` command has been made more powerful; the autotagger is more tolerant of different tagging styles; and importing with original file deletion now cleans up after itself more thoroughly. Many, many bugs—including several crashers—were fixed. This release lays the foundation for more features to come in the next couple of releases.

- The *Lyrics Plugin*, originally by Peter Brunner, is revamped and included with beets, making it easy to fetch **song lyrics**.
- Items now expose their audio **sample rate**, number of **channels**, and **bits per sample** (bitdepth). See *Path Formats* for a list of all available audio properties. Thanks to Andrew Dunn.
- The `beet list` command now accepts a "format" argument that lets you **show specific information about each album or track**. For example, run `beet ls -af '$album: $tracktotal' beatles` to see how long each Beatles album is. Thanks to Philippe Mongeau.
- The autotagger now tolerates tracks on multi-disc albums that are numbered per-disc. For example, if track 24 on a release is the first track on the second disc, then it is not penalized for having its track number set to 1 instead of 24.
- The autotagger sets the disc number and disc total fields on autotagged albums.
- The autotagger now also tolerates tracks whose track artists tags are set to "Various Artists".
- Terminal colors are now supported on Windows via Colorama (thanks to Karl).
- When previewing metadata differences, the importer now shows discrepancies in track length.

- Importing with `import_delete` enabled now cleans up empty directories that contained deleting imported music files.
- Similarly, `import_delete` now causes original album art imported from the disk to be deleted.
- Plugin-supplied template values, such as those created by `rewrite`, are now properly sanitized (for example, `AC/DC` properly becomes `AC_DC`).
- Filename extensions are now always lower-cased when copying and moving files.
- The `inline` plugin now prints a more comprehensible error when exceptions occur in Python snippets.
- The `replace` configuration option can now remove characters entirely (in addition to replacing them) if the special string `<strip>` is specified as the replacement.
- New plugin API: plugins can now add fields to the MediaFile tag abstraction layer. See *Writing Plugins*.
- A reasonable error message is now shown when the import log file cannot be opened.
- The import log file is now flushed and closed properly so that it can be used to monitor import progress, even when the import crashes.
- Duplicate track matches are no longer shown when autotagging singletons.
- The `chroma` plugin now logs errors when fingerprinting fails.
- The `lastgenre` plugin suppresses more errors when dealing with the Last.fm API.
- Fix a bug in the `rewrite` plugin that broke the use of multiple rules for a single field.
- Fix a crash with non-ASCII characters in bytestring metadata fields (e.g., MusicBrainz IDs).
- Fix another crash with non-ASCII characters in the configuration paths.
- Fix a divide-by-zero crash on zero-length audio files.
- Fix a crash in the `chroma` plugin when the Acoustid database had no recording associated with a fingerprint.
- Fix a crash when an autotagging with an artist or album containing “AND” or “OR” (upper case).
- Fix an error in the `rewrite` and `inline` plugins when the corresponding config sections did not exist.
- Fix bitrate estimation for AAC files whose headers are missing the relevant data.
- Fix the `list` command in BPD (thanks to Simon Chopin).

1.4.3 1.0b12 (January 16, 2012)

This release focuses on making beets’ path formatting vastly more powerful. It adds a function syntax for transforming text. Via a new plugin, arbitrary Python code can also be used to define new path format fields. Each path format template can now be activated conditionally based on a query. Character set substitutions are also now configurable.

In addition, beets avoids problematic filename conflicts by appending numbers to filenames that would otherwise conflict. Three new plugins (`inline`, `scrub`, and `rewrite`) are included in this release.

- **Functions in path formats** provide a simple way to write complex file naming rules: for example, `%upper{%left{$artist,1}}` will insert the capitalized first letter of the track’s artist. For more details, see *Path Formats*. If you’re interested in adding your own template functions via a plugin, see *Writing Plugins*.
- Plugins can also now define new path *fields* in addition to functions.
- The new *Inline Plugin* lets you **use Python expressions to customize path formats** by defining new fields in the config file.

- The configuration can **condition path formats based on queries**. That is, you can write a path format that is only used if an item matches a given query. (This supersedes the earlier functionality that only allowed conditioning on album type; if you used this feature in a previous version, you will need to replace, for example, `soundtrack:` with `albumtype_soundtrack:`.) See *Path Format Configuration*.
- **Filename substitutions are now configurable** via the `replace` config value. You can choose which characters you think should be allowed in your directory and music file names. See *.beetsconfig*.
- Beets now ensures that files have **unique filenames** by appending a number to any filename that would otherwise conflict with an existing file.
- The new *Scrub Plugin* can remove extraneous metadata either manually or automatically.
- The new *Rewrite Plugin* can canonicalize names for path formats.
- The autotagging heuristics have been tweaked in situations where the MusicBrainz database did not contain track lengths. Previously, beets penalized matches where this was the case, leading to situations where seemingly good matches would have poor similarity. This penalty has been removed.
- Fix an incompatibility in BPD with libmpc (the library that powers mpc and ncmpc).
- Fix a crash when importing a partial match whose first track was missing.
- The `lastgenre` plugin now correctly writes discovered genres to imported files (when tag-writing is enabled).
- Add a message when skipping directories during an incremental import.
- The default ignore settings now ignore all files beginning with a dot.
- Date values in path formats (`$year`, `$month`, and `$day`) are now appropriately zero-padded.
- Removed the `--path-format` global flag for `beet`.
- Removed the `lastid` plugin, which was deprecated in the previous version.

1.4.4 1.0b11 (December 12, 2011)

This version of beets focuses on transitioning the autotagger to the new version of the MusicBrainz database (called NGS). This transition brings with it a number of long-overdue improvements: most notably, predictable behavior when tagging multi-disc albums and integration with the new Acoustid acoustic fingerprinting technology.

The importer can also now tag *incomplete* albums when you're missing a few tracks from a given release. Two other new plugins are also included with this release: one for assigning genres and another for ReplayGain analysis.

- Beets now communicates with MusicBrainz via the new Next Generation Schema (NGS) service via python-musicbrainz-ngs. The bindings are included with this version of beets, but a future version will make them an external dependency.
- The importer now detects **multi-disc albums** and tags them together. Using a heuristic based on the names of directories, certain structures are classified as multi-disc albums: for example, if a directory contains subdirectories labeled "disc 1" and "disc 2", these subdirectories will be coalesced into a single album for tagging.
- The new *Chromaprint/Acoustid Plugin* uses the Acoustid **open-source acoustic fingerprinting** service. This replaces the old `lastid` plugin, which used Last.fm fingerprinting and is now deprecated. Fingerprinting with this library should be faster and more reliable.
- The importer can now perform **partial matches**. This means that, if you're missing a few tracks from an album, beets can still tag the remaining tracks as a single album. (Thanks to Simon Chopin.)
- The new *LastGenre Plugin* automatically **assigns genres to imported albums** and items based on Last.fm tags and an internal whitelist. (Thanks to KraYmer.)

- The *ReplayGain Plugin*, written by Peter Brunner, has been merged into the core beets distribution. Use it to analyze audio and **adjust playback levels** in ReplayGain-aware music players.
- Albums are now tagged with their *original* release date rather than the date of any reissue, remaster, "special edition", or the like.
- The config file and library databases are now given better names and locations on Windows. Namely, both files now reside in `%APPDATA%`; the config file is named `beetsconfig.ini` and the database is called `beetslibrary.blb` (neither has a leading dot as on Unix). For backwards compatibility, beets will check the old locations first.
- When entering an ID manually during tagging, beets now searches for anything that looks like an MBID in the entered string. This means that full MusicBrainz URLs now work as IDs at the prompt. (Thanks to derwin.)
- The importer now ignores certain "clutter" files like `.AppleDouble` directories and `._*` files. The list of ignored patterns is configurable via the `ignore` setting; see *.beetsconfig*.
- The database now keeps track of files' modification times so that, during an `update`, unmodified files can be skipped. (Thanks to Jos van der Til.)
- The album art fetcher now uses albumart.org as a fallback when the Amazon art downloader fails.
- A new `timeout` config value avoids database locking errors on slow systems.
- Fix a crash after using the "as Tracks" option during import.
- Fix a Unicode error when tagging items with missing titles.
- Fix a crash when the state file (`~/.beetsstate`) became emptied or corrupted.

1.4.5 1.0b10 (September 22, 2011)

This version of beets focuses on making it easier to manage your metadata *after* you've imported it. A bumper crop of new commands has been added: a manual tag editor (`modify`), a tool to pick up out-of-band deletions and modifications (`update`), and functionality for moving and copying files around (`move`). Furthermore, the concept of "re-importing" is new: you can choose to re-run beets' advanced autotagger on any files you already have in your library if you change your mind after you finish the initial import.

As a couple of added bonuses, imports can now automatically skip previously-imported directories (with the `-i` flag) and there's an *experimental Web interface* to beets in a new standard plugin.

- A new `beet modify` command enables **manual, command-line-based modification** of music metadata. Pass it a query along with `field=value` pairs that specify the changes you want to make.
- A new `beet update` command updates the database to reflect **changes in the on-disk metadata**. You can now use an external program to edit tags on files, remove files and directories, etc., and then run `beet update` to make sure your beets library is in sync. This will also rename files to reflect their new metadata.
- A new `beet move` command can **copy or move files** into your library directory or to another specified directory.
- When importing files that are already in the library database, the items are no longer duplicated—instead, the library is updated to reflect the new metadata. This way, the import command can be transparently used as a **re-import**.
- Relatedly, the `-L` flag to the "import" command makes it take a query as its argument instead of a list of directories. The matched albums (or items, depending on the `-s` flag) are then re-imported.
- A new flag `-i` to the import command runs **incremental imports**, keeping track of and skipping previously-imported directories. This has the effect of making repeated import commands pick up only newly-added directories. The `import_incremental` config option makes this the default.

- When pruning directories, "clutter" files such as `.DS_Store` and `Thumbs.db` are ignored (and removed with otherwise-empty directories).
- The *Web Plugin* encapsulates a simple **Web-based GUI for beets**. The current iteration can browse the library and play music in browsers that support HTML5 Audio.
- When moving items that are part of an album, the album art implicitly moves too.
- Files are no longer silently overwritten when moving and copying files.
- Handle exceptions thrown when running Mutagen.
- Fix a missing `__future__` import in `embed art` on Python 2.5.
- Fix ID3 and MPEG-4 tag names for the album-artist field.
- Fix Unicode encoding of album artist, album type, and label.
- Fix crash when "copying" an art file that's already in place.

1.4.6 1.0b9 (July 9, 2011)

This release focuses on a large number of small fixes and improvements that turn beets into a well-oiled, music-devouring machine. See the full release notes, below, for a plethora of new features.

- **Queries can now contain whitespace.** Spaces passed as shell arguments are now preserved, so you can use your shell's escaping syntax (quotes or backslashes, for instance) to include spaces in queries. For example, typing‘‘beet ls "the knife"‘‘ or `beet ls the\ knife`. Read more in *Queries*.
- Queries can **match items from the library by directory**. A `path:` prefix is optional; any query containing a path separator (/ on POSIX systems) is assumed to be a path query. Running `beet ls path/to/music` will show all the music in your library under the specified directory. The *Queries* reference again has more details.
- **Local album art** is now automatically discovered and copied from the imported directories when available.
- When choosing the "as-is" import album (or doing a non-autotagged import), **every album either has an "album artist" set or is marked as a compilation (Various Artists)**. The choice is made based on the homogeneity of the tracks' artists. This prevents compilations that are imported as-is from being scattered across many directories after they are imported.
- The release **label** for albums and tracks is now fetched from !MusicBrainz, written to files, and stored in the database.
- The "list" command now accepts a `-p` switch that causes it to **show paths** instead of titles. This makes the output of `beet ls -p` suitable for piping into another command such as xargs.
- Release year and label are now shown in the candidate selection list to help disambiguate different releases of the same album.
- Prompts in the importer interface are now colorized for easy reading. The default option is always highlighted.
- The importer now provides the option to specify a MusicBrainz ID manually if the built-in searching isn't working for a particular album or track.
- `$bitrate` in path formats is now formatted as a human-readable kbps value instead of as a raw integer.
- The import logger has been improved for "always-on" use. First, it is now possible to specify a log file in .beetsconfig. Also, logs are now appended rather than overwritten and contain timestamps.
- Album art fetching and plugin events are each now run in separate pipeline stages during imports. This should bring additional performance when using album art plugins like embedart or beets-lyrics.

- Accents and other Unicode decorators on characters are now treated more fairly by the autotagger. For example, if you're missing the acute accent on the "e" in "café", that change won't be penalized. This introduces a new dependency on the unidecode Python module.
- When tagging a track with no title set, the track's filename is now shown (instead of nothing at all).
- The bitrate of lossless files is now calculated from their file size (rather than being fixed at 0 or reflecting the uncompressed audio bitrate).
- Fixed a problem where duplicate albums or items imported at the same time would fail to be detected.
- BPD now uses a persistent "virtual filesystem" in order to fake a directory structure. This means that your path format settings are respected in BPD's browsing hierarchy. This may come at a performance cost, however. The virtual filesystem used by BPD is available for reuse by plugins (e.g., the FUSE plugin).
- Singleton imports (`beet import -s`) can now take individual files as arguments as well as directories.
- Fix Unicode queries given on the command line.
- Fix crasher in quiet singleton imports (`import -qs`).
- Fix crash when autotagging files with no metadata.
- Fix a rare deadlock when finishing the import pipeline.
- Fix an issue that was causing mpdupdate to run twice for every album.
- Fix a bug that caused release dates/years not to be fetched.
- Fix a crasher when setting MBIDs on MP3s file metadata.
- Fix a "broken pipe" error when piping beets' standard output.
- A better error message is given when the database file is unopenable.
- Suppress errors due to timeouts and bad responses from MusicBrainz.
- Fix a crash on album queries with item-only field names.

1.4.7 1.0b8 (April 28, 2011)

This release of beets brings two significant new features. First, beets now has first-class support for "singleton" tracks. Previously, it was only really meant to manage whole albums, but many of us have lots of non-album tracks to keep track of alongside our collections of albums. So now beets makes it easy to tag, catalog, and manipulate your individual tracks. Second, beets can now (optionally) embed album art directly into file metadata rather than only storing it in a "file on the side." Check out the *EmbedArt Plugin* for that functionality.

- Better support for **singleton (non-album) tracks**. Whereas beets previously only really supported full albums, now it can also keep track of individual, off-album songs. The "singleton" path format can be used to customize where these tracks are stored. To import singleton tracks, provide the -s switch to the import command or, while doing a normal full-album import, choose the "as Tracks" (T) option to add singletons to your library. To list only singleton or only album tracks, use the new `singleton:` query term: the query `singleton:true` matches only singleton tracks; `singleton:false` matches only album tracks. The `lastid` plugin has been extended to support matching individual items as well.
- The importer/autotagger system has been heavily refactored in this release. If anything breaks as a result, please get in touch or just file a bug.
- Support for **album art embedded in files**. A new *EmbedArt Plugin* implements this functionality. Enable the plugin to automatically embed downloaded album art into your music files' metadata. The plugin also provides the "embedart" and "extractart" commands for moving image files in and out of metadata. See the wiki for more details. (Thanks, daenney!)

- The “distance” number, which quantifies how different an album’s current and proposed metadata are, is now displayed as “similarity” instead. This should be less noisy and confusing; you’ll now see 99.5% instead of 0.00489323.
- A new “timid mode” in the importer asks the user every time, even when it makes a match with very high confidence. The `-t` flag on the command line and the `import_timid` config option control this mode. (Thanks to mdecker on GitHub!)
- The multithreaded importer should now abort (either by selecting aBort or by typing ^C) much more quickly. Previously, it would try to get a lot of work done before quitting; now it gives up as soon as it can.
- Added a new plugin event, `album_imported`, which is called every time an album is added to the library. (Thanks, Lugoues!)
- A new plugin method, `register_listener`, is an imperative alternative to the `@listen` decorator (Thanks again, Lugoues!)
- In path formats, `$albumartist` now falls back to `$artist` (as well as the other way around).
- The importer now prints “(unknown album)” when no tags are present.
- When autotagging, “and” is considered equal to “&”.
- Fix some crashes when deleting files that don’t exist.
- Fix adding individual tracks in BPD.
- Fix crash when `~/.beetsconfig` does not exist.

1.4.8 1.0b7 (April 5, 2011)

Beta 7’s focus is on better support for “various artists” releases. These albums can be treated differently via the new `[paths]` config section and the autotagger is better at handling them. It also includes a number of oft-requested improvements to the `beet` command-line tool, including several new configuration options and the ability to clean up empty directory subtrees.

- **“Various artists” releases** are handled much more gracefully. The autotagger now sets the `comp` flag on albums whenever the album is identified as a “various artists” release by !MusicBrainz. Also, there is now a distinction between the “album artist” and the “track artist”, the latter of which is never “Various Artists” or other such bogus stand-in. *(Thanks to Jonathan for the bulk of the implementation work on this feature!)*
- The directory hierarchy can now be **customized based on release type**. In particular, the `path_format` setting in .beetsconfig has been replaced with a new `[paths]` section, which allows you to specify different path formats for normal and “compilation” (various artists) releases as well as for each album type (see below). The default path formats have been changed to use `$albumartist` instead of `$artist`.
- A **new ‘‘albumtype‘‘ field** reflects the release type as specified by MusicBrainz.
- When deleting files, beets now appropriately “prunes” the directory tree—empty directories are automatically cleaned up. *(Thanks to wlof on GitHub for this!)*
- The tagger’s output now always shows the album directory that is currently being tagged. This should help in situations where files’ current tags are missing or useless.
- The logging option (`-l`) to the `import` command now logs duplicate albums.
- A new `import_resume` configuration option can be used to disable the importer’s resuming feature or force it to resume without asking. This option may be either `yes`, `no`, or `ask`, with the obvious meanings. The `-p` and `-P` command-line flags override this setting and correspond to the “yes” and “no” settings.
- Resuming is automatically disabled when the importer is in quiet (`-q`) mode. Progress is still saved, however, and the `-p` flag (above) can be used to force resuming.

- The `BEETSCONFIG` environment variable can now be used to specify the location of the config file that is at ~/.beetsconfig by default.
- A new `import_quiet_fallback` config option specifies what should happen in quiet mode when there is no strong recommendation. The options are `skip` (the default) and "asis".
- When importing with the "delete" option and importing files that are already at their destination, files could be deleted (leaving zero copies afterward). This is fixed.
- The `version` command now lists all the loaded plugins.
- A new plugin, called `info`, just prints out audio file metadata.
- Fix a bug where some files would be erroneously interpreted as MPEG-4 audio.
- Fix permission bits applied to album art files.
- Fix malformed !MusicBrainz queries caused by null characters.
- Fix a bug with old versions of the Monkey's Audio format.
- Fix a crash on broken symbolic links.
- Retry in more cases when !MusicBrainz servers are slow/overloaded.
- The old "albumify" plugin for upgrading databases was removed.

1.4.9 1.0b6 (January 20, 2011)

This version consists primarily of bug fixes and other small improvements. It's in preparation for a more feature-ful release in beta 7. The most important issue involves correct ordering of autotagged albums.

- **Quiet import:** a new "-q" command line switch for the import command suppresses all prompts for input; it pessimistically skips all albums that the importer is not completely confident about.
- Added support for the **WavPack** and **Musepack** formats. Unfortunately, due to a limitation in the Mutagen library (used by beets for metadata manipulation), Musepack SV8 is not yet supported. Here's the upstream bug in question.
- BPD now uses a pure-Python socket library and no longer requires eventlet/greenlet (the latter of which is a C extension). For the curious, the socket library in question is called Bluelet.
- Non-autotagged imports are now resumable (just like autotagged imports).
- Fix a terrible and long-standing bug where track orderings were never applied. This manifested when the tagger appeared to be applying a reasonable ordering to the tracks but, later, the database reflects a completely wrong association of track names to files. The order applied was always just alphabetical by filename, which is frequently but not always what you want.
- We now use Windows' "long filename" support. This API is fairly tricky, though, so some instability may still be present—please file a bug if you run into pathname weirdness on Windows. Also, filenames on Windows now never end in spaces.
- Fix crash in lastid when the artist name is not available.
- Fixed a spurious crash when `LANG` or a related environment variable is set to an invalid value (such as `'UTF-8'` on some installations of Mac OS X).
- Fixed an error when trying to copy a file that is already at its destination.
- When copying read-only files, the importer now tries to make the copy writable. (Previously, this would just crash the import.)
- Fixed an `UnboundLocalError` when no matches are found during autotag.

- Fixed a Unicode encoding error when entering special characters into the "manual search" prompt.
- Added ‘‘ beet version‘‘ command that just shows the current release version.

1.4.10 1.0b5 (September 28, 2010)

This version of beets focuses on increasing the accuracy of the autotagger. The main addition is an included plugin that uses acoustic fingerprinting to match based on the audio content (rather than existing metadata). Additional heuristics were also added to the metadata-based tagger as well that should make it more reliable. This release also greatly expands the capabilities of beets' *plugin API*. A host of other little features and fixes are also rolled into this release.

- The `lastid` plugin adds Last.fm **acoustic fingerprinting support** to the autotagger. Similar to the PUIDs used by !MusicBrainz Picard, this system allows beets to recognize files that don't have any metadata at all. You'll need to install some dependencies for this plugin to work.
- To support the above, there's also a new system for **extending the autotagger via plugins**. Plugins can currently add components to the track and album distance functions as well as augment the MusicBrainz search. The new API is documented at *Plugins*.
- **String comparisons** in the autotagger have been augmented to act more intuitively. Previously, if your album had the title "Something (EP)" and it was officially called "Something", then beets would think this was a fairly significant change. It now checks for and appropriately reweights certain parts of each string. As another example, the title "The Great Album" is considered equal to "Great Album, The".
- New **event system for plugins** (thanks, Jeff!). Plugins can now get callbacks from beets when certain events occur in the core. Again, the API is documented in *Plugins*.
- The BPD plugin is now disabled by default. This greatly simplifies installation of the beets core, which is now 100% pure Python. To use BPD, though, you'll need to set `plugins: bpd` in your .beetsconfig.
- The `import` command can now remove original files when it copies items into your library. (This might be useful if you're low on disk space.) Set the `import_delete` option in your .beetsconfig to `yes`.
- Importing without autotagging (`beet import -A`) now prints out album names as it imports them to indicate progress.
- The new *MPDUpdate Plugin* will automatically update your MPD server's index whenever your beets library changes.
- Efficiency tweak should reduce the number of !MusicBrainz queries per autotagged album.
- A new `-v` command line switch enables debugging output.
- Fixed bug that completely broke non-autotagged imports (`import -A`).
- Fixed bug that logged the wrong paths when using `import -l`.
- Fixed autotagging for the creatively-named band !!!.
- Fixed normalization of relative paths.
- Fixed escaping of / characters in paths on Windows.

1.4.11 1.0b4 (August 9, 2010)

This thrilling new release of beets focuses on making the tagger more usable in a variety of ways. First and foremost, it should now be much faster: the tagger now uses a multithreaded algorithm by default (although, because the new tagger is experimental, a single-threaded version is still available via a config option). Second, the tagger output now uses a little bit of ANSI terminal coloring to make changes stand out. This way, it should be faster to decide what to do with a proposed match: the more red you see, the worse the match is. Finally, the tagger can be safely interrupted (paused) and restarted later at the same point. Just enter `b` for aBort at any prompt to stop the tagging process and save its progress. (The progress-saving also works in the unthinkable event that beets crashes while tagging.)

Among the under-the-hood changes in 1.0b4 is a major change to the way beets handles paths (filenames). This should make the whole system more tolerant to special characters in filenames, but it may break things (especially databases created with older versions of beets). As always, let me know if you run into weird problems with this release.

Finally, this release's `setup.py` should install a `beet.exe` startup stub for Windows users. This should make running beets much easier: just type `beet` if you have your `PATH` environment variable set up correctly. The *Getting Started* guide has some tips on installing beets on Windows.

Here's the detailed list of changes:

- **Parallel tagger.** The autotagger has been reimplemented to use multiple threads. This means that it can concurrently read files from disk, talk to the user, communicate with MusicBrainz, and write data back to disk. Not only does this make the tagger much faster because independent work may be performed in parallel, but it makes the tagging process much more pleasant for large imports. The user can let albums queue up in the background while making a decision rather than waiting for beets between each question it asks. The parallel tagger is on by default but a sequential (single- threaded) version is still available by setting the `threaded` config value to `no` (because the parallel version is still quite experimental).
- **Colorized tagger output.** The autotagger interface now makes it a little easier to see what's going on at a glance by highlighting changes with terminal colors. This feature is on by default, but you can turn it off by setting `color` to `no` in your `.beetsconfig` (if, for example, your terminal doesn't understand colors and garbles the output).
- **Pause and resume imports.** The `import` command now keeps track of its progress, so if you're interrupted (beets crashes, you abort the process, an alien devours your motherboard, etc.), beets will try to resume from the point where you left off. The next time you run `import` on the same directory, it will ask if you want to resume. It accomplishes this by "fast-forwarding" through the albums in the directory until it encounters the last one it saw. (This means it might fail if that album can't be found.) Also, you can now abort the tagging process by entering `b` (for aBort) at any of the prompts.
- Overhauled methods for handling fileystem paths to allow filenames that have badly encoded special characters. These changes are pretty fragile, so please report any bugs involving `UnicodeError` or SQLite `ProgrammingError` messages in this version.
- The destination paths (the library directory structure) now respect album-level metadata. This means that if you have an album in which two tracks have different album-level attributes (like year, for instance), they will still wind up in the same directory together. (There's currently not a very smart method for picking the "correct" album-level metadata, but we'll fix that later.)
- Fixed a bug where the CLI would fail completely if the `LANG` environment variable was not set.
- Fixed removal of albums (`beet remove -a`): previously, the album record would stay around although the items were deleted.
- The setup script now makes a `beet.exe` startup stub on Windows; Windows users can now just type `beet` at the prompt to run beets.
- Fixed an occasional bug where Mutagen would complain that a tag was already present.
- Fixed a bug with reading invalid integers from ID3 tags.
- The tagger should now be a little more reluctant to reorder tracks that already have indices.

1.4.12 1.0b3 (July 22, 2010)

This release features two major additions to the autotagger's functionality: album art fetching and MusicBrainz ID tags. It also contains some important under-the-hood improvements: a new plugin architecture is introduced and the database schema is extended with explicit support for albums.

This release has one major backwards-incompatibility. Because of the new way beets handles albums in the library, databases created with an old version of beets might have trouble with operations that deal with albums (like the `-a` switch to `beet list` and `beet remove`, as well as the file browser for BPD). To "upgrade" an old database, you can use the included `albumify` plugin (see the fourth bullet point below).

- **Album art.** The tagger now, by default, downloads album art from Amazon that is referenced in the MusicBrainz database. It places the album art alongside the audio files in a file called (for example) `cover.jpg`. The `import_art` config option controls this behavior, as do the `-r` and `-R` options to the import command. You can set the name (minus extension) of the album art file with the `art_filename` config option. (See *.beetsconfig* for more information about how to configure the album art downloader.)
- **Support for MusicBrainz ID tags.** The autotagger now keeps track of the MusicBrainz track, album, and artist IDs it matched for each file. It also looks for album IDs in new files it's importing and uses those to look up data in MusicBrainz. Furthermore, track IDs are used as a component of the tagger's distance metric now. (This obviously lays the groundwork for a utility that can update tags if the MB database changes, but that's for the future.) Tangentially, this change required the database code to support a lightweight form of migrations so that new columns could be added to old databases–this is a delicate feature, so it would be very wise to make a backup of your database before upgrading to this version.
- **Plugin architecture.** Add-on modules can now add new commands to the beets command-line interface. The `bpd` and `dadd` commands were removed from the beets core and turned into plugins; BPD is loaded by default. To load the non-default plugins, use the config options `plugins` (a space-separated list of plugin names) and `pluginpath` (a colon-separated list of directories to search beyond `sys.path`). Plugins are just Python modules under the `beetsplug` namespace package containing subclasses of `beets.plugins.BeetsPlugin`. See the beetsplug directory for examples or *Plugins* for instructions.
- As a consequence of adding album art, the database was significantly refactored to keep track of some information at an album (rather than item) granularity. Databases created with earlier versions of beets should work fine, but they won't have any "albums" in them–they'll just be a bag of items. This means that commands like `beet ls -a` and `beet rm -a` won't match anything. To "upgrade" your database, you can use the included `albumify` plugin. Running `beets albumify` with the plugin activated (set `plugins=albumify` in your config file) will group all your items into albums, making beets behave more or less as it did before.
- Fixed some bugs with encoding paths on Windows. Also, `:` is now replaced with `-` in path names (instead of `_`) for readability.
- `MediaFile``s now have a ``format` attribute, so you can use `$format` in your library path format strings like `$artist - $album ($format)` to get directories with names like `Paul Simon - Graceland (FLAC)`.

Beets also now has its first third-party plugin: beetfs, by Martin Eve! It exposes your music in a FUSE filesystem using a custom directory structure. Even cooler: it lets you keep your files intact on-disk while correcting their tags when accessed through FUSE. Check it out!

1.4.13 1.0b2 (July 7, 2010)

This release focuses on high-priority fixes and conspicuously missing features. Highlights include support for two new audio formats (Monkey's Audio and Ogg Vorbis) and an option to log untaggable albums during import.

- **Support for Ogg Vorbis and Monkey's Audio** files and their tags. (This support should be considered preliminary: I haven't tested it heavily because I don't use either of these formats regularly.)
- An option to the `beet import` command for **logging albums that are untaggable** (i.e., are skipped or taken "as-is"). Use `beet import -l LOGFILE PATHS`. The log format is very simple: it's just a status (either "skip" or "asis") followed by the path to the album in question. The idea is that you can tag a large collection

and automatically keep track of the albums that weren't found in MusicBrainz so you can come back and look at them later.

- Fixed a `UnicodeEncodeError` on terminals that don't (or don't claim to) support UTF-8.
- Importing without autotagging (`beet import -A`) is now faster and doesn't print out a bunch of whitespace. It also lets you specify single files on the command line (rather than just directories).
- Fixed importer crash when attempting to read a corrupt file.
- Reorganized code for CLI in preparation for adding pluggable subcommands. Also removed dependency on the aging `cmdln` module in favor of a hand-rolled solution.

1.4.14 1.0b1 (June 17, 2010)

Initial release.