beets Documentation

Release 1.3.7

Adrian Sampson

August 22, 2014

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 *Configuration* references. You might also be interested in exploring the *plugins*.

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

CHAPTER 1

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) and 10.8 (Mountain Lion) include 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`. But the version in the repositories might lag behind, so make sure you read the right version of these docs. If you want the latest version, you can get everything you need to install with pip as described below by running: `apt-get install python-dev python-pip`
- On **Arch Linux**, beets is in [community], so just run `pacman -S beets`. (There's also a bleeding-edge dev package in the AUR, which will probably set your computer on fire.)
- For **Gentoo Linux**, beets is in Portage as `media-sound/beets`. Just run `emerge beets` to install. There are several USE flags available for optional plugin dependencies.
- On **FreeBSD**, there's a beets port at `audio/beets`.
- On **OpenBSD-current**, beets is available in ports (at `audio/beets`) and as a package, which can be installed with `pkg_add beets`.
- For **Slackware**, there's a SlackBuild available.

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, download beets from its PyPI page 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. 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`.
3. Next, install pip (if you don't have it already) by downloading and running the get-pip.py script.
4. Now install beets by running: `pip install beets`
5. You're all set! Type `beet` at the command prompt to make sure everything's in order.

Windows users may also want to install a context menu item for importing files into beets. Just download and open beets.reg to add the necessary keys to the registry. You can then right-click a directory and choose "Import with beets". If Python is in a nonstandard location on your system, you may have to edit the command path manually.

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 direct it to the mailing list.

Configuring

You'll want to set a few basic options before you start using beets. The *configuration* is stored in a text file. You can show its location by running `beet config -p`, though it may not exist yet. Run `beet config -e` to edit the configuration in your favorite text editor. The file will start out empty, but here's good place to start:

```
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 this in your config file:

  ```
  import:
      move: yes
  ```

- Keep your current directory structure; importing should never move or copy files but instead just correct the tags on music. Put the line `copy: no` under the `import:` heading 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.) Put this in your config file:

```
import:
    copy: no
    write: no
```

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 *Configuration* 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. For more 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.

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
```

Keep Playing

This is only the beginning of your long and prosperous journey with beets. To keep learning, take a look at *Advanced Awesomeness* for a sampling of what else is possible. You'll also want to glance over the *Command-Line Interface* page for a more detailed description of all of beets' functionality. (Like deleting music! That's important.)

Also, check out *beets' 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 the mailing list 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.

  There are, however, a couple of exceptions to this rule:

  First, directories that look like separate parts of a *multi-disc album* are tagged together as a single release. If two adjacent albums have a common prefix, followed by "disc," "disk," or "CD" and then a number, they are tagged together.

  Second, if you have jumbled directories containing more than one album, you can ask beets to split them apart for you based on their metadata. Use either the `--group-albums` command-line flag or the *G* interactive option described below.

- The music may have bad tags, but it's not completely untagged. This is because beets by default infers tags based on existing metadata. But this is not a hard and fast rule—there are a few ways to tag metadata-poor music:
  - You can use the *E* option described below to search in MusicBrainz for a specific album or song.
  - 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. (But be aware that it does slow down the process.)
  - The *FromFilename plugin* adds the ability to guess tags from the filenames. Use this plugin if your tracks have useful names (like "03 Call Me Maybe.mp3") but their tags don't reflect that.
- Currently, MP3, AAC, FLAC, ALAC, Ogg Vorbis, Monkey's Audio, WavPack, Musepack, Windows Media, Opus, and AIFF files are supported. (Do you use some other format? Please file a feature request!)

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 -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.
- `beet import -g`: assume there are multiple albums contained in each directory. The tracks contained a directory are grouped by album artist and album name and you will be asked to import each of these groups separately. See the "Group albums" choice below.

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, G, 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.
- *G*: Group tracks in this directory by *album artist* and *album* and import groups as albums. If the album artist for a track is not set then the artist is used to group that track. For each group importing proceeds as for directories. This is helpful if a directory contains multiple albums.
- *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.

Before you jump into acoustic fingerprinting with both feet, though, give beets a try without it. You may be surprised at how well metadata-based matching works.

Album Art, Lyrics, Genres and Such

Aside from the basic stuff, beets can optionally fetch more specialized metadata. As a rule, plugins are responsible for getting information that doesn't come directly from the MusicBrainz database. This includes *album cover art*, *song lyrics*, and *musical genres*. Check out the *list of plugins* to pick and choose the data you want.

Missing Albums?

If you're having trouble tagging a particular album with beets, check to make sure the album is 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.

If you think beets is ignoring an album that's listed in MusicBrainz, please file a bug report.

I Hope That Makes Sense

If I haven't made the process clear, please send an email to the mailing list and I'll try to improve this guide.

1.1.3 Advanced Awesomeness

So you have beets up and running and you've started *importing your music*. There's a lot more that beets can do now that it has cataloged your collection. Here's a few features to get you started.

Most of these tips involve *plugins* and fiddling with beets' *configuration*. So use your favorite text editor create a config file before you continue.

Fetch album art, genres, and lyrics

Beets can help you fill in more than just the basic taxonomy metadata that comes from MusicBrainz. Plugins can provide *album art*, *lyrics*, and *genres* from databases around the Web.

If you want beets to get any of this data automatically during the import process, just enable any of the three relevant plugins (see *Plugins*). For example, put this line in your *config file* to enable all three:

```
plugins: fetchart lyrics lastgenre
```

Each plugin also has a command you can run to fetch data manually. For example, if you want to get lyrics for all the Beatles tracks in your collection, just type `beet lyrics beatles` after enabling the plugin.

Read more about using each of these plugins:

- *FetchArt Plugin* (and its accompanying *EmbedArt Plugin*)

- *Lyrics Plugin*
- *LastGenre Plugin*

Customize your file and folder names

Beets uses an extremely flexible template system to name the folders and files that organize your music in your filesystem. Take a look at *Path Format Configuration* for the basics: use fields like `$year` and `$title` to build up a naming scheme. But if you need more flexibility, there are two features you need to know about:

- *Template functions* are simple expressions you can use in your path formats to add logic to your names. For example, you can get an artist's first initial using `%upper{%left{$albumartist,1}}`.
- If you need more flexibility, the *Inline Plugin* lets you write snippets of Python code that generate parts of your filenames. The equivalent code for getting an artist initial with the *inline* plugin looks like `initial: albumartist[0].upper()`.

If you already have music in your library and want to update their names according to a new scheme, just run the *move* command to rename everything.

Stream your music to another computer

Sometimes it can be really convenient to store your music on one machine and play it on another. For example, I like to keep my music on a server at home but play it at work (without copying my whole library locally). The *Web Plugin* makes streaming your music easy—it's sort of like having your own personal Spotify.

First, enable the `web` plugin (see *Plugins*). Run the server by typing `beet web` and head to http://localhost:8337 in a browser. You can browse your collection with queries and, if your browser supports it, play music using HTML5 audio.

But for a great listening experience, pair beets with the Tomahawk music player. Tomahawk lets you listen to music from many different sources, including a beets server. Just download Tomahawk and open its settings to connect it to beets. A post on the beets blog has a more detailed guide.

Transcode music files for media players

Do you ever find yourself transcoding high-quality rips to a lower-bitrate, lossy format for your phone or music player? Beets can help with that.

You'll first need to install ffmpeg. Then, enable beets' *Convert Plugin*. Set a destination directory in your *config file* like so:

```
convert:
    dest: ~/converted_music
```

Then, use the command `beet convert QUERY` to transcode everything matching the query and drop the resulting files in that directory, named according to your path formats. For example, `beet convert long winters` will move over everything by the Long Winters for listening on the go.

The plugin has many more dials you can fiddle with to get your conversions how you like them. Check out *its documentation*.

Store any data you like

The beets database keeps track of a long list of *built-in fields*, but you're not limited to just that list. Say, for example, that you like to categorize your music by the setting where it should be played. You can invent a new `context` attribute store this. Set the field using the *modify* command:

```
beet modify context=party artist:'beastie boys'
```

And then *query* your music just as you would with any other field:

```
beet ls context:mope
```

You can even use these fields in your filenames (see *Path Format Configuration*).

And, unlike *built-in fields*, such fields can be removed:

```
beet modify context! artist:'beastie boys'
```

Read more than you ever wanted to know about the *flexible attributes* feature on the beets blog.

Choose a path style manually for some music

Sometimes, you need to categorize some songs differently in your file system. For example, you might want to group together all the music you don't really like but keep around to play for friends and family. This is, of course, impossible to determine automatically using metadata from MusicBrainz.

Instead, use a flexible attribute (see above) to store a flag on the music you want to categorize, like so:

```
beet modify bad=1 christmas
```

Then, you can query on this field in your path formats to sort this music differently. Put something like this in your configuration file:

```
paths:
    bad:1: Bad/$artist/$title
```

Used together, flexible attributes and path format conditions let you sort your music by any criteria you can imagine.

1.1.4 Upgrading from 1.0

Prior to version 1.1, beets used a completely different system for configuration. The config file was in "INI" syntax instead of YAML and the various files used by beets were (messily) stored in `$HOME` instead of a centralized beets directory. If you're upgrading from version 1.0 or earlier, your configuration syntax (and paths) need to be updated to work with the latest version.

Fortunately, this should require very little effort on your part. When you first run beets 1.1, it will look for an old-style `.beetsconfig` to migrate. If it finds one (and there is no new-style `config.yaml` yet), beets will warn you and then transparently convert one to the other. At this point, you'll likely want to:

- Look at your new configuration file (find out where in *Configuration*) to make sure everything was migrated correctly.
- Remove your old configuration file (`~/.beetsconfig` on Unix; `%APPDATA%\beetsconfig.ini` on Windows) to avoid confusion in the future.

You might be interested in the *Changelog* to see which configuration option names have changed.

What's Migrated

Automatic migration is most important for the configuration file, since its syntax is completely different, but two other files are also moved. This is to consolidate everything beets needs in a single directory instead of leaving it messily strewn about in your home directory.

First, the library database file was at `~/.beetsmusic.blb` on Unix and `%APPDATA%\beetsmusic.blb` on Windows. This file will be copied to `library.db` in the same directory as your new configuration file. Finally, the runtime state file, which keeps track of interrupted and incremental imports, was previously known as `~/.beetsstate`; it is copied to a file called `state.pickle`.

Feel free to remove the old files once they've been copied to their new homes.

Manual Migration

If you find you need to re-run the migration process, just type `beet migrate` in your shell. This will migrate the configuration file, the database, and the runtime state file all over again. Unlike automatic migration, no step is suppressed if the file already exists. If you already have a `config.yaml`, for example, it will be renamed to make room for the newly migrated configuration.

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] PATH...
beet import [options] -L QUERY
```

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

Point the command at some music: directories, single files, or compressed archives. The music will be copied to a configurable directory structure and added to a library database. The command is interactive and will try to get you to verify MusicBrainz tags that it thinks are suspect. See the *autotagging guide* for detail on how to use the interactive tag-correction flow.

Directories passed to the import command can contain either a single album or many, in which case the leaf directories will be considered albums (the latter case is true of typical Artist/Album organizations and many people's "downloads" folders). The path can also be a single song or an archive. Beets supports *zip* and *tar* archives out of the box. To extract *rar* files, install the rarfile package and the *unrar* command.

Optional command flags:

- 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 an 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. To get this to work correctly, you'll need to use an incremental import *every time* you run an import on the directory in question—including the first time, when no subdirectories will be skipped. So consider enabling the `incremental` configuration option.
- By default, beets will proceed without asking if it finds a very close metadata match. To disable this and have the importer ask you every time, use the `-t` (for *timid*) option.
- 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.
- If you have an album that's split across several directories under a common top directory, use the `--flat` option. This takes all the music files under the directory (recursively) and treats them as a single large album instead of as one album per directory. This can help with your more stubborn multi-disc albums.
- Similarly, if you have one directory that contains multiple albums, use the `--group-albums` option to split the files based on their metadata before matching them as separate albums.

list

```
beet list [-apf] 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. In this case, the queries you use are restricted to album-level fields: for example, you can search for `year:1969` but query parts for item-level fields like `title:foo` will be ignored. Remember that `artist` is an item-level field; `albumartist` is the corresponding album field.

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...] [FIELD!...]
```

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." To remove fields (which is only possible for flexible attributes), follow a field name with an exclamation point: `field!`.

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" of 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.

When an updated track is part of an album, the album-level fields of *all* tracks from the album are also updated. (Specifically, the command copies album-level data from the first track on the album and applies it to the rest of the tracks.) This means that, if album-level fields aren't identical within an album, some changes shown by the `update` command may be overridden by data from other tracks on the same album. This means that running the `update` command multiple times may show the same changes being applied.

write

```
beet write [-pf] [QUERY]
```

Write metadata from the database into files' tags.

When you make changes to the metadata stored in beets' library database (during import or with the *modify* command, for example), you often have the option of storing changes only in the database, leaving your files untouched. The `write` command lets you later change your mind and write the contents of the database into the files. By default, this writes the changes only if there is a difference between the database and the tags in the file.

You can think of this command as the opposite of *update*.

The `-p` option previews metadata changes without actually applying them.

The `-f` option forces a write to the file, even if the file tags match the database. This is useful for making sure that enabled plugins that run on write (e.g., the Scrub and Zero plugins) are run on the file.

stats

```
beet stats [-e] [QUERY]
```

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

The `-e` (`--exact`) option makes the calculation of total file size more accurate but slower.

fields

```
beet fields
```

Show the item and album metadata fields available for use in *Queries* and *Path Formats*. Includes any template fields provided by plugins.

config

```
beet config [-pd]
beet config -e
```

Show or edit the user configuration. This command does one of three things:

- With no options, print a YAML representation of the current user configuration. With the `--default` option, beets' default options are also included in the dump.
- The `--path` option instead shows the path to your configuration file. This can be combined with the `--default` flag to show where beets keeps its internal defaults.
- With the `--edit` option, beets attempts to open your config file for editing. It first tries the `$EDITOR` environment variable and then a fallback option depending on your platform: `open` on OS X, `xdg-open` on Unix, and direct invocation on Windows.

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.
- `-c FILE`: read a specified YAML *configuration file*.

Beets also uses the `BEETSDIR` environment variable to look for configuration and data.

Shell Completion

Beets includes support for shell command completion. The command `beet completion` prints out a bash 3.2 script; to enable completion put a line like this into your `.bashrc` or similar file:

```
eval "$(beet completion)"
```

Or, to avoid slowing down your shell startup time, you can pipe the `beet completion` output to a file and source that instead.

You will also need to source the bash-completion script, which is probably available via your package manager. On OS X, you can install it via Homebrew with `brew install bash-completion`; Homebrew will give you instructions for sourcing the script.

The completion script suggests names of subcommands and (after typing `-`) options of the given command. If you are using a command that accepts a query, the script will also complete field names.

```
beet list ar[TAB]
# artist:  artist_credit:  artist_sort:  artpath:
beet list artp[TAB]
beet list artpath\:
```

(Don't worry about the slash in front of the colon: this is a escape sequence for the shell and won't be seen by beets.)

Completion of plugin commands only works for those plugins that were enabled when running `beet completion`. If you add a plugin later on you will want to re-generate the script.

If you use zsh, take a look instead at the included completion script.

1.2.2 Configuration

Beets has an extensive configuration system that lets you customize nearly every aspect of its operation. To configure beets, you create a file called `config.yaml`. The location of the file depend on your platform (type `beet config -p` to see the path on your system):

- On Unix-like OSes, write `~/.config/beets/config.yaml`.
- On Windows, use `%APPDATA%\beets\config.yaml`. This is usually in a directory like `C:\Users\You\AppData\Roaming`.
- On OS X, you can use either the Unix location or `~/Library/Application Support/beets/config.yaml`.

You can launch your text editor to create or update your configuration by typing `beet config -e`. (See the *config* command for details.) It is also possible to customize the location of the configuration file and even use multiple layers of configuration. See Configuration Location, below.

The config file uses YAML syntax. You can use the full power of YAML, but most configuration options are simple key/value pairs. This means your config file will look like this:

```
option: value
another_option: foo
bigger_option:
    key: value
    foo: bar
```

In YAML, you will need to use spaces (not tabs!) to indent some lines. If you have questions about more sophisticated syntax, take a look at the YAML documentation.

The rest of this page enumerates the dizzying litany of configuration options available in beets. You might also want to see an *example*.

Global Options

These options control beets' global operation.

library

Path to the beets library file. By default, beets will use a file called `library.db` alongside your configuration file.

directory

The directory to which files will be copied/moved when adding them to the library. Defaults to a folder called `Music` in your home directory.

plugins

A space-separated list of plugin module names to load. See *Using Plugins*.

pluginpath

Directories to search for plugins. Each Python file or directory in a plugin path represents a plugin and should define a subclass of `BeetsPlugin`. A plugin can then be loaded by adding the filename to the *plugins* configuration. The plugin path can either be a single string or a list of strings—so, if you have multiple paths, format them as a YAML list like so:

```
pluginpath:
    - /path/one
    - /path/two
```

ignore

A list of glob patterns specifying file and directory names to be ignored when importing. By default, this consists of `.*`, `*~`, and `System Volume Information` (i.e., beets ignores Unix-style hidden files, backup files, and a directory that appears at the root of some Windows filesystems).

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 dots, which hide files on Unix, and trailing whitespace, which is illegal on Windows). To override these substitutions, specify a mapping from regular expression to replacement strings. For example, `[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]': _
    '[<>:"\?\*\|]': _
    '\.$': _
    '\s+$': ''
    '^\s+': ''
```

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.

Note that paths might contain special characters such as typographical quotes (“”). With the configuration above, those will not be replaced as they don't match the typewriter quote (`"`). To also strip these special characters, you can either add them to the replacement list or use the *asciify_paths* configuration option below.

asciify_paths

Convert all non-ASCII characters in paths to ASCII equivalents.

For example, if your path template for singletons is `singletons/$title` and the title of a track is "Café", then the track will be saved as `singletons/Cafe.mp3`. The changes take place before applying the *replace* configuration and are roughly equivalent to wrapping all your path templates in the `%asciify{}` *template function*.

Default: `no`.

art_filename

When importing album art, the name of the file (without extension) where the cover art image should be placed. This is a template string, so you can use any of the syntax available to *Path Formats*. Defaults to `cover` (i.e., images will be named `cover.jpg` or `cover.png` and placed in the album's directory).

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.

list_format_item

Format to use when listing *individual items* with the *list* command and other commands that need to print out items. Defaults to `$artist - $album - $title`. The `-f` command-line option overrides this setting.

list_format_album

Format to use when listing *albums* with *list* and other commands. Defaults to `$albumartist - $album`. The `-f` command-line option overrides this setting.

original_date

Either `yes` or `no`, indicating whether matched albums should have their `year`, `month`, and `day` fields set to the release date of the *original* version of an album rather than the selected version of the release. That is, if this option is turned on, then `year` will always equal `original_year` and so on. Default: `no`.

per_disc_numbering

A boolean controlling the track numbering style on multi-disc releases. By default (`per_disc_numbering: no`), tracks are numbered per-release, so the first track on the second disc has track number N+1 where N is the number of tracks on the first disc. If this `per_disc_numbering` is enabled, then the first track on each disc always has track number 1.

If you enable `per_disc_numbering`, you will likely want to change your *Path Format Configuration* also to include `$disc` before `$track` to make filenames sort correctly in album directories. For example, you might want to use a path format like this:

```
paths:
    default: $albumartist/$album%aunique{}/$disc-$track $title
```

terminal_encoding

The text encoding, as known to Python, to use for messages printed to the standard output. By default, this is determined automatically from the locale environment variables.

clutter

When beets imports all the files in a directory, it tries to remove the directory if it's empty. A directory is considered empty if it only contains files whose names match the glob patterns in *clutter*, which should be a list of strings. The default list consists of "Thumbs.DB" and ".DS_Store".

max_filename_length

Set the maximum number of characters in a filename, after which names will be truncated. By default, beets tries to ask the filesystem for the correct maximum.

id3v23

By default, beets writes MP3 tags using the ID3v2.4 standard, the latest version of ID3. Enable this option to instead use the older ID3v2.3 standard, which is preferred by certain older software such as Windows Media Player.

Importer Options

The options that control the *import* command are indented under the `import:` key. For example, you might have a section in your configuration file that looks like this:

```
import:
    write: yes
    copy: yes
    resume: no
```

These options are available in this section:

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.

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

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

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.

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`.

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.

none_rec_action

Either `ask` (default), `asis` or `skip`. Specifies what should happen during an interactive import session when there is no recommendation. Useful when you are only interested in processing medium and strong recommendations interactively.

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.

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`.

default_action

One of `apply`, `skip`, `asis`, or `none`, indicating which option should be the *default* when selecting an action for a given match. This is the action that will be taken when you type return without an option letter. The default is `apply`.

languages

A list of locale names to search for preferred aliases. For example, setting this to "en" uses the transliterated artist name "Pyotr Ilyich Tchaikovsky" instead of the Cyrillic script for the composer's name when tagging from MusicBrainz. Defaults to an empty list, meaning that no language is preferred.

detail

Whether the importer UI should show detailed information about each match it finds. When enabled, this mode prints out the title of every track, regardless of whether it matches the original metadata. (The default behavior only shows changes.) Default: `no`.

group_albums

By default, the beets importer groups tracks into albums based on the directories they reside in. This option instead uses files' metadata to partition albums. Enable this option if you have directories that contain tracks from many albums mixed together.

The `--group-albums` or `-g` option to the *import* command is equivalent, and the *G* interactive option invokes the same workflow.

Note: The *import log* currently contains less information in album-grouping mode. (Specifically, no directory names recorded because directories are not used for grouping in this mode.)

Default: `no`.

autotag

By default, the beets importer always attempts to autotag new music. If most of your collection consists of obscure music, you may be interested in disabling autotagging by setting this option to `no`. (You can re-enable it with the `-a` flag to the *import* command.)

Default: `yes`.

MusicBrainz Options

If you run your own MusicBrainz server, you can instruct beets to use it instead of the main server. Use the `host` and `ratelimit` options under a `musicbrainz:` header, like so:

```
musicbrainz:
    host: localhost:5000
    ratelimit: 100
```

The `host` key, of course, controls the Web server hostname (and port, optionally) that will be contacted by beets (default: musicbrainz.org). The `ratelimit` option, an integer, controls the number of Web service requests per second (default: 1). **Do not change the rate limit setting** if you're using the main MusicBrainz server—on this public server, you're limited to one request per second.

Autotagger Matching Options

You can configure some aspects of the logic beets uses when automatically matching MusicBrainz results under the `match:` section. To control how *tolerant* the autotagger is of differences, use the `strong_rec_thresh` option, which reflects the distance threshold below which beets will make a "strong recommendation" that the metadata be used. Strong recommendations are accepted automatically (except in "timid" mode), so you can use this to make beets ask your opinion more or less often.

The threshold is a *distance* value between 0.0 and 1.0, so you can think of it as the opposite of a *similarity* value. For example, if you want to automatically accept any matches above 90% similarity, use:

```
match:
    strong_rec_thresh: 0.10
```

The default strong recommendation threshold is 0.04.

The `medium_rec_thresh` and `rec_gap_thresh` options work similarly. When a match is above the *medium* recommendation threshold or the distance between it and the next-best match is above the *gap* threshold, the importer will suggest that match but not automatically confirm it. Otherwise, you'll see a list of options to choose from.

max_rec

As mentioned above, autotagger matches have *recommendations* that control how the UI behaves for a certain quality of match. The recommendation for a certain match is based on the overall distance calculation. But you can also control the recommendation when a specific distance penalty is applied by defining *maximum* recommendations for each field:

To define maxima, use keys under `max_rec:` in the `match` section. The defaults are "medium" for missing and unmatched tracks and "strong" (i.e., no maximum) for everything else:

```
match:
    max_rec:
        missing_tracks: medium
        unmatched_tracks: medium
```

If a recommendation is higher than the configured maximum and the indicated penalty is applied, the recommendation is downgraded. The setting for each field can be one of `none`, `low`, `medium` or `strong`. When the maximum recommendation is `strong`, no "downgrading" occurs. The available penalty names here are:

- source
- artist
- album

- media
- mediums
- year
- country
- label
- catalognum
- albumdisambig
- album_id
- tracks
- missing_tracks
- unmatched_tracks
- track_title
- track_artist
- track_index
- track_length
- track_id

preferred

In addition to comparing the tagged metadata with the match metadata for similarity, you can also specify an ordered list of preferred countries and media types.

A distance penalty will be applied if the country or media type from the match metadata doesn't match. The specified values are preferred in descending order (i.e., the first item will be most preferred). Each item may be a regular expression, and will be matched case insensitively. The number of media will be stripped when matching preferred media (e.g. "2x" in "2xCD").

You can also tell the autotagger to prefer matches that have a release year closest to the original year for an album.

Here's an example:

```
match:
    preferred:
        countries: ['US', 'GB|UK']
        media: ['CD', 'Digital Media|File']
        original_year: yes
```

By default, none of these options are enabled.

ignored

You can completely avoid matches that have certain penalties applied by adding the penalty name to the `ignored` setting:

```
match:
    ignored: missing_tracks unmatched_tracks
```

The available penalties are the same as those for the *max_rec* setting.

required

You can avoid matches that lack certain required information. Add the tags you want to enforce to the `required` setting:

```
match:
    required: year label catalognum country
```

No tags are required by default.

Path Format Configuration

You can also configure the directory hierarchy beets uses to store music. These settings appear under the `paths:` key. 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*). 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.

Configuration Location

The beets configuration file is usually located in a standard location that depends on your OS, but there are a couple of ways you can tell beets where to look.

Environment Variable

First, you can set the `BEETSDIR` environment variable to a directory containing a `config.yaml` file. This replaces your configuration in the default location. This also affects where auxiliary files, like the library database, are stored by default (that's where relative paths are resolved to). This environment variable is useful if you need to manage multiple beets libraries with separate configurations.

Command-Line Option

Alternatively, you can use the `--config` command-line option to indicate a YAML file containing options that will then be merged with your existing options (from `BEETSDIR` or the default locations). This is useful if you want to keep your configuration mostly the same but modify a few options as a batch. For example, you might have different strategies for importing files, each with a different set of importer options.

Default Location

In the absence of a `BEETSDIR` variable, beets searches a few places for your configuration, depending on the platform:

- On Unix platforms, including OS X:`~/.config/beets` and then `$XDG_CONFIG_DIR/beets`, if the environment variable is set.
- On OS X, we also search `~/Library/Application Support/beets` before the Unixy locations.
- On Windows: `~\AppData\Roaming\beets`, and then `%APPDATA%\beets`, if the environment variable is set.

Beets uses the first directory in your platform's list that contains `config.yaml`. If no config file exists, the last path in the list is used.

Example

Here's an example file:

```
library: /var/music.blb
directory: /var/mp3
import:
    copy: yes
    write: yes
    resume: ask
    quiet_fallback: skip
    timid: no
    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
```

1.2.3 Path Formats

The `paths:` section of the config file (see *Configuration*) 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.

Template 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. See the *asciify_paths* configuration option.
- `%aunique{identifiers,disambiguators}`: Provides a unique string to disambiguate similar albums in the database. See *Album Disambiguation*, below.
- `%time{date_time,format}`: Return the date and time in any format accepted by strftime. For example, to get the year some music was added to your library, use `%time{$added,%Y}`.

Plugins can extend beets with more template functions (see *Template functions and values provided by 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 `%aunique` 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 *Template functions and values provided by plugins*).

Ordinary metadata:

- title
- artist

- artist_sort: The "sort name" of the track artist (e.g., "Beatles, The" or "White, Jack").
- artist_credit: The track-specific artist credit name, which may be a variation of the artist's "canonical" name.
- album
- albumartist: The artist for the entire album, which may be different from the artists for the individual tracks.
- albumartist_sort
- albumartist_credit
- genre
- composer
- grouping
- year, month, day: The release date of the specific release.
- original_year, original_month, original_day: The release date of the original version of the album.
- track
- tracktotal
- disc
- disctotal
- lyrics
- comments
- bpm
- comp: Compilation flag.
- 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

Library metadata:

- mtime: The modification time of the audio file.
- added: The date and time that the music was added to your library.
- path: The item's filename.

Template functions and values provided by plugins

Beets plugins can provide additional fields and functions to templates. See the *Plugins* page for a full list of plugins. Some plugin-provided constructs include:

- `$missing` by *Missing Plugin*: The number of missing tracks per album.
- `%bucket{text}` by *Bucket Plugin*: Substitute a string by the range it belongs to.
- `%the{text}` by *The Plugin*: Moves English articles to ends of strings.

The *Inline Plugin* lets you define template fields in your beets configuration file using Python snippets. And for more advanced processing, you can go all-in and write a dedicated plugin to register your own fields and functions (see *Writing Plugins*).

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.

Numeric Range Queries

For numeric fields, such as year, bitrate, and track, you can query using one- or two-sided intervals. That is, you can find music that falls within a *range* of values. To use ranges, write a query that has two dots (`..`) at the beginning, middle, or end of a string of numbers. Dots in the beginning let you specify a maximum (e.g., `..7`); dots at the end mean a minimum (`4..`); dots in the middle mean a range (`4..7`).

For example, this command finds all your albums that were released in the '90s:

```
$ beet list -a year:1990..1999
```

and this command finds MP3 files with bitrates of 128k or lower:

```
$ beet list format:MP3 bitrate:..128000
```

Date and Date Range Queries

Date-valued fields, such as *added* and *mtime*, have a special query syntax that lets you specify years, months, and days as well as ranges between dates.

Dates are written separated by hyphens, like `year-month-day`, but the month and day are optional. If you leave out the day, for example, you will get matches for the whole month.

Date *intervals*, like the numeric intervals described above, are separated by two dots (`..`). You can specify a start, an end, or both.

Here is an example that finds all the albums added in 2008:

```
$ beet ls -a 'added:2008'
```

Find all items added in the years 2008, 2009 and 2010:

```
$ beet ls 'added:2008..2010'
```

Find all items added before the year 2010:

```
$ beet ls 'added:..2009'
```

Find all items added on or after 2008-12-01 but before 2009-10-12:

```
$ beet ls 'added:2008-12..2009-10-11'
```

Find all items with a file modification time between 2008-12-01 and 2008-12-03:

```
$ beet ls 'mtime:2008-12-01..2008-12-02'
```

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 extend beets' core functionality. They add new commands, fetch additional data during import, provide new metadata sources, and much more. If beets by itself doesn't do what you want it to, you may just need to enable a plugin—or, if you want to do something new, *writing a plugin* is easy if you know a little Python.

1.3.1 Using Plugins

To use one of the plugins included with beets (see the rest of this page for a list), just use the *plugins* option in your *config.yaml*: file, like so:

```
plugins: inline discogs web
```

The value for *plugins* can be a space-separated list of plugin names or a YAML list like `[foo, bar]`. You can see which plugins are currently enabled by typing `beet version`.

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

```
$ pip install pyacoustid
```

Then, you will need to install Chromaprint, either as a dynamic library or in the form of a command-line tool (`fpcalc`).

Installing the Binary Command-Line Tool The simplest way to get up and running, especially on Windows, is to download the appropriate Chromaprint binary package and place the `fpcalc` (or `fpcalc.exe`) on your shell search path. On Windows, this means something like `C:\\Program Files`. On OS X or Linux, put the executable somewhere like `/usr/local/bin`.

Installing the Library On OS X and Linux, you can also use a library installed by your package manager, which has some advantages (automatic upgrades, etc.). The Chromaprint site has links to packages for major Linux distributions. If you use Homebrew on Mac OS X, you can install the library with `brew install chromaprint`.

You will also need a mechanism for decoding audio files supported by the audioread library:

- OS X has a number of decoders already built into Core Audio, so there's no need to install anything.
- On Linux, you can install GStreamer for Python, FFmpeg, or MAD and pymad. 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`. If you use GStreamer, be sure to install its codec plugins also.
- On Windows, try the Gstreamer "WinBuilds" from the OSSBuild project.

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`.

Using

Once you have all the dependencies sorted out, you can enable fingerprinting by editing your *configuration file*. Put `chroma` on your `plugins:` line. With that, beets will use fingerprinting the next time you run `beet import`.

You can also use the `beet fingerprint` command to generate fingerprints for items already in your library. (Provide a query to fingerprint a subset of your library.) The generated fingerprints will be stored in the library database. If you have the `import.write` config option enabled, they will also be written to files' metadata.

You can disable fingerprinting on import by setting the `auto` option to false, like so:

```
chroma:
    auto: no
```

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 `config.yaml` 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, Lyrics.com, and, optionally, the Google custom search API.

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 `auto` 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.

The `-f` option forces the command to fetch lyrics, even for tracks that already have lyrics.

Configuring

To disable automatic lyric fetching during import, set the `auto` option to false, like so:

```
lyrics:
    auto: no
```

By default, if no lyrics are found, the file will be left unchanged. To specify a placeholder for the lyrics tag when none are found, use the `fallback` configuration option:

```
lyrics:
    fallback: 'No lyrics found'
```

Activate Google custom search

Using the Google backend requires BeautifulSoup, which you can install using pip by typing:

```
pip install beautifulsoup4
```

You also need to register for a Google API key. Set the `google_API_key` configuration option to your key. This enables the Google backend.

Optionally, you can define a custom search engine. Get your search engine's token and use it for your `google_engine_ID` configuration option. By default, beets use a list of sources known to be scrapeable.

Here's an example of `config.yaml`:

```
lyrics:
  google_API_key: AZERTYUIOPQSDFGHJKLMWXCVBN1234567890_ab
  google_engine_ID: 009217259823014548361:lndtuqkycfu
```

Note that the Google custom search API is limited to 100 queries per day. After that, the lyrics plugin will fall back on its other data sources.

EchoNest Tempo Plugin

Note: A newer *Echo Nest Plugin* is available that supersedes this plugin. In addition to the tempo, the new plugin can fetch the Echo Nest's full complement of acoustic attributes. This older tempo-specific plugin is **deprecated**.

The `echonest_tempo` plugin fetches and stores a track's tempo (the "bpm" field) from the Echo Nest API.

Installing Dependencies

This plugin requires the pyechonest library in order to talk to the EchoNest API.

There are packages for most major linux distributions, you can download the library from the Echo Nest, or you can install the library from pip, like so:

```
$ pip install pyechonest
```

Configuring

Beets includes its own Echo Nest API key, but you can apply for your own for free from the EchoNest. To specify your own API key, add the key to your *configuration file* as the value for `apikey` under the key `echonest_tempo` like so:

```
echonest_tempo:
    apikey: YOUR_API_KEY
```

In addition, the `autofetch` config option lets you disable automatic tempo fetching during import. To do so, add this to your `config.yaml`:

```
echonest_tempo:
    auto: no
```

Fetch Tempo During Import

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

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

Fetching Tempo Manually

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

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

Echo Nest Plugin

The `echonest` plugin fetches acoustic attributes from the Echo Nest. It automatically fills in the following attributes:

- danceability
- energy
- liveness
- loudness
- speechiness
- bpm

All attributes except `bpm` are stored in flexible attributes (i.e., not in files' metadata). See the Echo Nest's page on acoustic attributes for a detailed description. (Their name for `bpm` is `tempo`.)

Installing Dependencies

This plugin requires the pyechonest library in order to talk to the Echo Nest API. At least version 8.0.1 is required.

There are packages for most major linux distributions, you can download the library from the Echo Nest, or you can install the library from pip, like so:

```
$ pip install pyechonest
```

To transcode music for server-side analysis (optional, of course), install the ffmpeg command-line tool.

To get fingerprinting working, you'll need to install the command-line codegen tool for ENMFP or Echoprint, the two fingerprinting algorithms supported by the Echo Nest. Please note that fingerprinting is not required if `upload` and `convert` is enabled, which is the default (but it can be faster than uploading).

Configuring

Beets includes its own Echo Nest API key, but you can apply for your own for free from the Echo Nest. To specify your own API key, add the key to your *configuration file* as the value for `apikey` under the key `echonest_tempo` like so:

```
echonest:
    apikey: YOUR_API_KEY
```

In addition, the `auto` config option lets you disable automatic metadata fetching during import. To do so, add this to your `config.yaml`:

```
echonest:
    auto: no
```

The `echonest` plugin tries to upload files to the Echo Nest server if it can not be identified by other means. If you don't want that, disable the `upload` config option like so:

```
echonest:
    upload: no
```

The Echo Nest server only supports a limited range of file formats. The plugin automatically converts unsupported files to `ogg`. If you don't want that, disable the `convert` config option like so:

```
echonest:
    convert: no
```

The Echo Nest server does not allow uploading of files with sizes greater than 50MB. The plugin automatically truncates large files to their first 5 minutes. If you don't want that, disable the `truncate` config option like so:

```
echonest:
    truncate: no
```

To enable fingerprinting, you'll need to tell the plugin where to find the Echoprint or ENMFP codegen binary. Use the `codegen` key under the `echonest` section like so:

```
echonest:
    codegen: /usr/bin/echoprint-codegen
```

Running Manually

In addition to running automatically on import, the plugin can also be run manually from the command line. Use the command `beet echonest [QUERY]` to fetch acoustic attributes for albums matching a certain query.

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, the appropriate formulae are in homebrew-versions, so run `brew tap homebrew/versions` and then `brew install gst-python010`. (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).

You will also need the various GStreamer plugin packages to make everything work. See the *Chromaprint/Acoustid Plugin* documentation for more information on installing GStreamer plugins.

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 *configuration file* and add `bpd` to your `plugins:` line.

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 `config.yaml` 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.

The `update` command regenerates the directory tree from the beets database.

Unimplemented Commands

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

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 `config.yaml` by putting `mpdupdate` on your `plugins` line. Then, you'll probably want to configure the specifics of your MPD server. You can do that using an `mpd:` section in your `config.yaml`, which looks like this:

```
mpd:
    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.

If you want to communicate with MPD over a Unix domain socket instead over TCP, just give the path to the socket in the filesystem for the `host` setting. (Any `host` value starting with a slash or a tilde is interpreted as a domain socket.)

MPDStats Plugin

`mpdstats` is a plugin for beets that collects statistics about your listening habits from MPD. It collects the following information about tracks:

- play_count: The number of times you *fully* listened to this track.
- skip_count: The number of times you *skipped* this track.
- last_played: UNIX timestamp when you last played this track.
- rating: A rating based on *play_count* and *skip_count*.

Installing Dependencies

This plugin requires the python-mpd library in order to talk to the MPD server.

Install the library from pip, like so:

```
$ pip install python-mpd
```

Configuring

To use it, enable it in your `config.yaml` by putting `mpdstats` on your `plugins` line. Then, you'll probably want to configure the specifics of your MPD server. You can do that using an `mpd:` section in your `config.yaml`, which looks like this:

```
mpd:
    host: localhost
    port: 6600
    password: seekrit
```

If your MPD library is at another location then the beets library (e.g., because one is mounted on a NFS share), you can specify the `music_directory` in the config like this:

```
mpdstats:
    music_directory: /PATH/TO/YOUR/FILES
```

If you don't want the plugin to update the rating, you can disable it with:

```
mpdstats:
    rating: False
```

If you want to change the way the rating is calculated, you can set the `rating_mix` option like this:

```
mpdstats:
    rating_mix: 1.0
```

For details, see below.

Usage

Now use the `mpdstats` command to fire it up:

```
$ beet mpdstats
```

A Word on Ratings

Ratings are calculated based on the *play_count*, *skip_count* and the last *action* (play or skip). It consists in one part of a *stable_rating* and in another part on a *rolling_rating*. The *stable_rating* is calculated like this:

```
stable_rating = (play_count + 1.0) / (play_count + skip_count + 2.0)
```

So if the *play_count* equals the *skip_count*, the *stable_rating* is always 0.5. More *play_counts* adjust the rating up to 1.0. More *skip_counts* adjust it down to 0.0. One of the disadvantages of this rating system, is that it doesn't really cover *recent developments*. e.g. a song that you loved last year and played over 50 times will keep a high rating even if you skipped it the last 10 times. That's were the *rolling_rating* comes in.

If a song has been fully played, the *rolling_rating* is calculated like this:

```
rolling_rating = old_rating + (1.0 - old_rating) / 2.0
```

If a song has been skipped, like this:

```
rolling_rating = old_rating - old_rating / 2.0
```

So *rolling_rating* adapts pretty fast to *recent developments*. But it's too fast. Taking the example from above, your old favorite with 50 plays will get a negative rating (<0.5) the first time you skip it. Also not good.

To take the best of both worlds, we mix the ratings together with the `rating_mix` factor. A `rating_mix` of 0.0 means all *rolling* and 1.0 means all *stable*. We found 0.75 to be a good compromise, but fell free to play with that.

Warning

This has only been tested with MPD versions >= 0.16. It may not work on older versions. If that is the case, please report an issue.

FetchArt Plugin

The `fetchart` plugin retrieves album art images from various sources on the Web and stores them as image files.

To use the plugin, first enable it in your configuration (see *Using Plugins*). Then, install the requests library by typing:

```
pip install requests
```

The plugin uses requests to fetch album art from the Web.

Fetching Album Art During Import

When the plugin is enabled, it automatically gets album art for every album you import.

By default, beets stores album art image files alongside the music files for an album in a file called `cover.jpg`. To customize the name of this file, use the *art_filename* config option.

To disable automatic art downloading, just put this in your configuration file:

```
fetchart:
    auto: no
```

Manually Fetching Album Art

Use the `fetchart` command to download album art after albums have already been imported:

```
$ beet fetchart [-f] [query]
```

By default, the command will only look for album art when the album doesn't already have it; the `-f` or `--force` switch makes it search for art in Web databases regardless. If you specify a query, only matching albums will be processed; otherwise, the command processes every album in your library.

Image Resizing

A maximum image width can be configured as `maxwidth` to downscale fetched images if they are too big. The resize operation reduces image width to `maxwidth` pixels. The height is recomputed so that the aspect ratio is preserved.

Beets can resize images using PIL, ImageMagick, or a server-side resizing proxy. If either PIL or ImageMagick is installed, beets will use those; otherwise, it falls back to the resizing proxy. If the resizing proxy is used, no resizing is performed for album art found on the filesystem—only downloaded art is resized. Server-side resizing can also be slower than local resizing, so consider installing one of the two backends for better performance.

When using ImageMagic, beets looks for the `convert` executable in your path. On some versions of Windows, the program can be shadowed by a system-provided `convert.exe`. On these systems, you may need to modify your `%PATH%` environment variable so that ImageMagick comes first or use PIL instead.

Album Art Sources

Currently, this plugin searches for art in the local filesystem as well as on the Cover Art Archive, Amazon, and AlbumArt.org (in that order).

When looking for local album art, beets checks for image files located in the same folder as the music files you're importing. Beets prefers to use an image file whose name contains "cover", "front", "art", "album" or "folder", but in the absence of well-known names, it will use any image file in the same folder as your music files.

You can change the list of filename keywords using the `cover_names` config option. Or, to use *only* filenames containing the keywords and not fall back to any image, set `cautious` to true. For example:

```
fetchart:
    cautious: true
    cover_names: front back
```

By default, remote (Web) art sources are only queried if no local art is found in the filesystem. To query remote sources every time, set the `remote_priority` configuration option to true, which will cause beets to prefer remote cover art over any local image files.

When you choose to apply changes during an import, beets will search for art as described above. For "as-is" imports (and non-autotagged imports using the `-A` flag), beets only looks for art on the local filesystem.

Google Image Search You can optionally search for cover art on Google Images. This option uses the first hit for a search query consisting of the artist and album name. It is therefore approximate: "incorrect" image matches are possible (although unlikely).

To enable gathering art from Google, enable the `google_search` option in your config file:

```
fetchart:
    google_search: true
```

Embedding Album Art

This plugin fetches album art but does not embed images into files' tags. To do that, use the *EmbedArt Plugin*. (You'll want to have both plugins enabled.)

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 (see *Plugins*). You'll also want to enable the *FetchArt Plugin* to obtain the images to be embedded. Art will be embedded after each album is added to the library.

This behavior can be disabled with the `auto` 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 [-f IMAGE] QUERY`: embed images into the every track on the albums matching the query. If the `-f` (`--file`) option is given, then use a specific image file from the filesystem; otherwise, each album embeds its own currently associated album art.
- `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 `auto` option lets you disable automatic album art embedding. To do so, add this to your `config.yaml`:

```
embedart:
    auto: no
```

A maximum image width can be configured as `maxwidth` to downscale images before embedding them (the original image file is not altered). The resize operation reduces image width to `maxwidth` pixels. The height is recomputed so that the aspect ratio is preserved. PIL or ImageMagick is required to use the `maxwidth` config option. See also *Image Resizing* for further caveats about image resizing.

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.

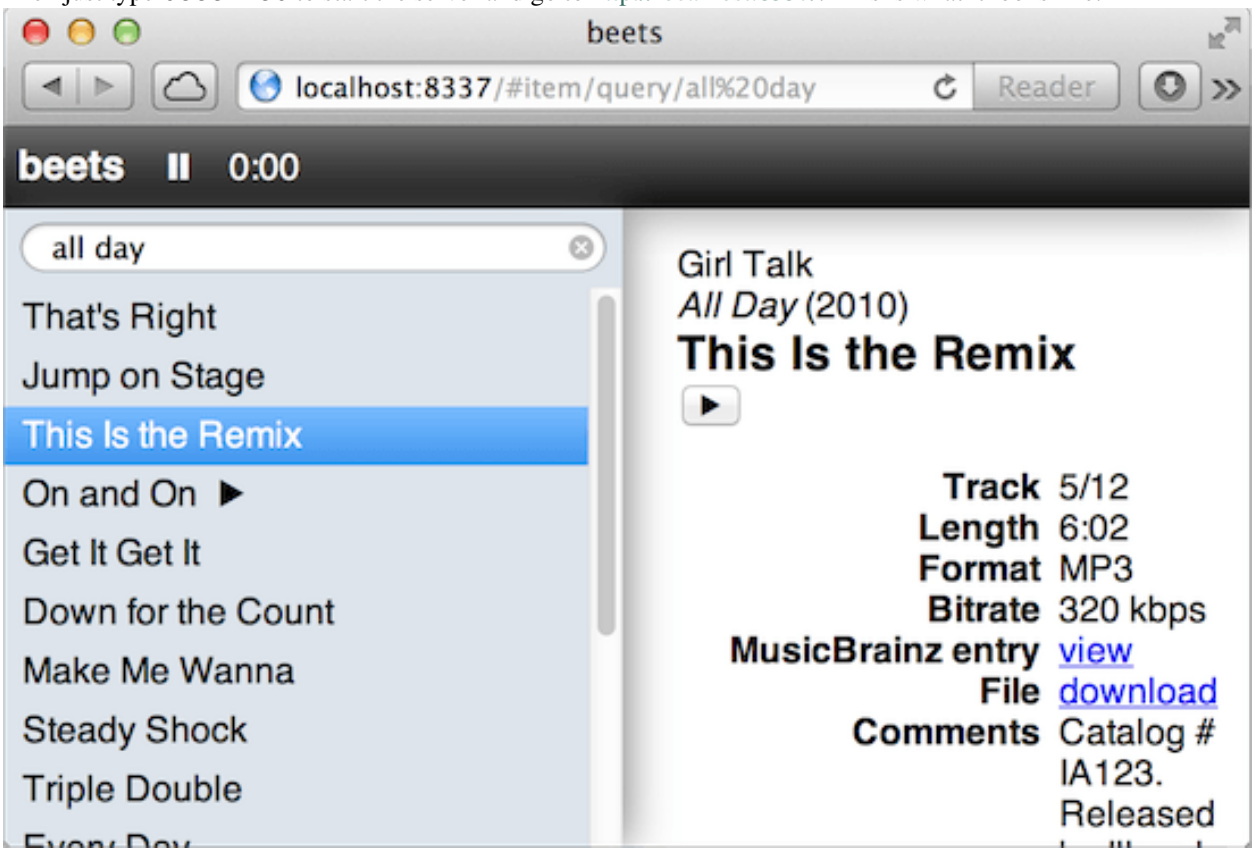
Install

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

Put `web` on your `plugins` line in your configuration file 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 *configuration file*.

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

JSON API

GET /item/ Responds with a list of all tracks in the beets library.

```
{
  "items": [
    {
      "id": 6,
      "title": "A Song",
      ...
    }, {
      "id": 12,
      "title": "Another Song",
      ...
    }
    ...
  ]
}
```

GET /item/6 Looks for an item with id *6* in the beets library and responds with its JSON representation.

```
{
  "id": 6,
  "title": "A Song",
  ...
}
```

If there is no item with that id responds with a *404* status code.

GET /item/6,12,13 Response with a list of tracks with the ids *6*, *12* and *13*. The format of the response is the same as for GET /item/. It is *not guaranteed* that the response includes all the items requested. If a track is not found it is silently dropped from the response.

GET /item/query/querystring Returns a list of tracks matching the query. The *querystring* must be a valid query as described in *Queries*.

```
{
  "results": [
    { "id" : 6,  "title": "A Song" },
    { "id" : 12, "title": "Another Song" }
  ]
}
```

GET /item/6/file Sends the media file for the track. If the item or its corresponding file do not exist a *404* status code is returned.

Albums For albums, the following endpoints are provided:

- `GET /album/`
- `GET /album/5`

- `GET /album/5,7`
- `GET /album/query/querystring`

The interface and response format is similar to the item API, except replacing the encapsulation key `"items"` with `"albums"` when requesting `/album/` or `/album/5,7`. In addition we can request the cover art of an album with `GET /album/5/art`.

`GET /stats` Responds with the number of tracks and albums in the database.

```
{
  "items": 5,
  "albums": 3
}
```

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 *config file*.

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
```

. . . or go for no whitelist altogether by setting the option to *false*.

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.

By default, beets will always fetch new genres, even if the files already have once. To instead leave genres in place in when they pass the whitelist, set the `force` option to *no*.

If no genre is found, the file will be left unchanged. To instead specify a fallback genre, use the `fallback` configuration option. You can, of course, use the empty string as a fallback, like so:

```
lastgenre:
    fallback: ''
```

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, set the `canonical` configuration value:

```
lastgenre:
    canonical: true
```

Setting this value to *true* 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.

Genre Source

When looking up genres for albums or individual tracks, you can choose whether to use Last.fm tags on the album, the artist, or the track. For example, you might want all the albums for a certain artist to carry the same genre. Set the `source` configuration value to "album", "track", or "artist", like so:

```
lastgenre:
    source: artist
```

The default is "album". When set to "track", the plugin will fetch *both* album-level and track-level genres for your music when importing albums.

Multiple Genres

By default, the plugin chooses the most popular tag on Last.fm as a genre. If you prefer to use a *list* of popular genre tags, you can increase the number of the `count` config option:

```
lastgenre:
    count: 3
```

Lists of up to *count* genres will then be used instead of single genres. The genres are separated by commas by default, but you can change this with the `separator` config option:

```
lastgenre:
    separator: ' / '
```

Last.fm provides a popularity factor, a.k.a. *weight*, for each tag ranging from 100 for the most popular tag down to 0 for the least popular. The plugin uses this weight to discard unpopular tags. The default is to ignore tags with a weight less then 10. You can change this by setting the `min_weight` config option:

```
lastgenre:
    min_weight: 15
```

Running Manually

In addition to running automatically on import, the plugin can also be run manually from the command line. Use the command `beet lastgenre [QUERY]` to fetch genres for albums matching a certain query.

To disable automatic genre fetching on import, set the `auto` config option to false.

ReplayGain Plugin

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

Installation

This plugin can use one of two backends to compute the ReplayGain values: GStreamer and mp3gain (and its cousin, aacgain). mp3gain can be easier to install but GStreamer support more audio formats.

Once installed, this plugin analyzes all files during the import process. This can be a slow process; to instead analyze after the fact, disable automatic analysis and use the `beet replaygain` command (see below).

GStreamer To use GStreamer for ReplayGain analysis, you will of course need to install GStreamer and plugins for compatibility with your audio files. You will need at least GStreamer 1.0 and PyGObject 3.x (a.k.a. python-gi).

Then, enable the plugin (see *Using Plugins*) and specify the GStreamer backend by adding this to your configuration file:

```
replaygain:
    backend: gstreamer
```

mp3gain and aacgain In order to use this backend, you will need to install the mp3gain command-line tool or the aacgain fork thereof. Here are some hints:

- On Mac OS X, you can use Homebrew. Type `brew install aacgain`.
- On Linux, mp3gain is probably in your repositories. On Debian or Ubuntu, for example, you can run `apt-get install mp3gain`.
- On Windows, download and install the original mp3gain.

Then, enable the plugin (see *Using Plugins*) and specify the "command" backend in your configuration file:

```
replaygain:
    backend: command
```

If beets doesn't automatically find the `mp3gain` or `aacgain` executable, you can configure the path explicitly like so:

```
replaygain:
    command: /Applications/MacMP3Gain.app/Contents/Resources/aacgain
```

Configuration

Available configuration options for the `replaygain` section in your configuration file include:

- **overwrite**: 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 *configuration file*, like so:

  ```
  replaygain:
      overwrite: yes
  ```

- **targetlevel**: The target loudness level can be modified to any number of decibels with the `targetlevel` option (default: 89 dB).

These options only work with the "command" backend:

- **apply**: If you use a player that does not support ReplayGain specifications, you can force the volume normalization by applying the gain to the file via the `apply` option. This is a lossless and reversible operation with no transcoding involved.
- **noclip**: The use of ReplayGain can cause clipping if the average volume of a song is below the target level. By default, a "prevent clipping" option named `noclip` is enabled to reduce the amount of ReplayGain adjustment to whatever amount would keep clipping from occurring.

Manual Analysis

By default, the plugin will analyze all items an albums as they are implemented. However, you can also manually analyze files that are already in your library. Use the `beet replaygain` command:

```
$ beet replaygain [-a] [QUERY]
```

The `-a` flag analyzes whole albums instead of individual tracks. Provide a query (see *Queries*) to indicate which items or albums to analyze.

ReplayGain analysis is not fast, so you may want to disable it during import. Use the `auto` config option to control this:

```
replaygain:
    auto: no
```

Inline Plugin

The `inline` plugin lets you use Python 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 *Configuration*).

To use inline field definitions, first enable the plugin by putting `inline` on your `plugins` line in your configuration file. Then, make a `item_fields:` block in your config file. Under this key, 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 or function body. The Python code 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 of expressions:

```
item_fields:
    initial: albumartist[0].upper() + u'.'
    disc_and_track: u'%02i.%02i' % (disc, track) if
                    disctotal > 1 else u'%02i' % (track)
```

Note that YAML syntax allows newlines in values if the subsequent lines are indented.

These examples define `$initial` and `$disc_and_track` fields that can be referenced in path templates like so:

```
paths:
    default: $initial/$artist/$album%aunique{}/$disc_and_track $title
```

Block Definitions

If you need to use statements like `import`, you can write a Python function body instead of a single expression. In this case, you'll need to `return` a result for the value of the path field, like so:

```
item_fields:
    filename: |
        import os
        from beets.util import bytestring_path
        return bytestring_path(os.path.basename(path))
```

You might want to use the YAML syntax for "block literals," in which a leading `|` character indicates a multi-line block of text.

Album Fields

The above examples define fields for *item* templates, but you can also define fields for *album* templates. Use the `album_fields` configuration section. In this context, all existing album fields are available as variables along with `items`, which is a list of items in the album.

This example defines a `$bitrate` field for albums as the average of the tracks' fields:

```
album_fields:
    bitrate: |
        total = 0
        for item in items:
            total += item.bitrate
        return total / len(items)
```

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 all types of tags entirely and then write the database-tracked metadata to the file.

This behavior can be disabled with the `auto` 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, `auto`, which lets you disable automatic metadata stripping. To do so, add this to your `config.yaml`:

```
scrub:
    auto: no
```

Rewrite Plugin

The `rewrite` plugin lets you easily substitute values in your templates and 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. 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 templates expand to "Jimi Hendrix" 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 templating; it does not modify files' metadata tags or the values tracked by beets' library database.

Random Plugin

The `random` 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 `random` (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`.

If the `-e` option is passed, the random choice will be even among artists (the albumartist field). This makes sure that your anthology of Bob Dylan won't make you listen to Bob Dylan 50% of the time.

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 *configuration file* under a `musicbrainz` section:

```
musicbrainz:
    user: you
    pass: seekrit
```

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 *configuration file*. Then set a few options under the `importfeeds:` section in the config file.

The `dir` configuration option can be set to specify another folder than the default library directory. This is where your playlist will be created.

The `relative_to` configuration option can be set to make the m3u paths relative to another folder than where the playlist is being written. If you're using importfeeds to generate a playlist for MPD, you should set this to the root of your music library.

The `absolute_path` configuration option can be set to use absolute paths instead of relative paths. Some applications may need this to work properly.

Three different types of outputs coexist, specify the ones you want to use by setting the `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 `dir` folder on the iTunes dock icon.

Here's an example configuration for this plugin:

```
importfeeds:
    formats: m3u link
    dir: ~/imports/
    relative_to: ~/Music/
    m3u_name: newfiles.m3u
```

The Plugin

The `the` plugin allows you to move patterns in path formats. It's suitable, for example, for moving articles from string start to the end. This is useful for quick search on filesystems and generally looks good. Plugin DOES NOT change tags. By default plugin supports English "the, a, an", but custom regexp patterns can be added by user. How it works:

```
The Something -> Something, The
A Band -> Band, A
An Orchestra -> Orchestra, An
```

To use plugin, enable it by including `the` into `plugins` line of your beets config. The plugin provides a template function called `%the` for use in path format expressions:

```
paths:
    default: %the{$albumartist}/($year) $album/$track $title
```

The default configuration moves all English articles to the end of the string, but you can override these defaults to make more complex changes:

```
the:
    # handle "The" (on by default)
    the: yes
    # handle "A/An" (on by default)
    a: yes
    # format string, {0} - part w/o article, {1} - article
    # spaces already trimmed from ends of both parts
    # default is '{0}, {1}'
    format: '{0}, {1}'
```

```
    # strip instead of moving to the end, default is off
    strip: no
    # custom regexp patterns, space-separated
    patterns: ...
```

Custom patterns are case-insensitive regular expressions. Patterns can be matched anywhere in the string (not just the beginning), so use ^ if you intend to match leading words.

Fuzzy Search Plugin

The `fuzzy` plugin provides a prefixed query that search you library using fuzzy pattern matching. This can be useful if you want to find a track with complicated characters in the title.

First, enable the plugin named `fuzzy` (see *Plugins*). You'll then be able to use the ~ prefix to use fuzzy matching:

```
$ beet ls '~Vareoldur'
Sigur Rós - Valtari - Varðeldur
```

The plugin provides config options that let you choose the prefix and the threshold.:

```
fuzzy:
    threshold: 0.8
    prefix: '@'
```

A threshold value of 1.0 will show only perfect matches and a value of 0.0 will match everything.

The default prefix ~ needs to be escaped or quoted in most shells. If this bothers you, you can change the prefix in your config file.

Zero Plugin

The `zero` plugin allows you to null fields in files' metadata tags. Fields can be nulled unconditionally or conditioned on a pattern match. For example, the plugin can strip useless comments like "ripped by MyGreatRipper." This plugin only affects files' tags; the beets database is unchanged.

To use the plugin, enable it by including `zero` in the `plugins` line of your configuration file. To configure the plugin, use a `zero:` section in your configuration file. Set `fields` to the (whitespace-separated) list of fields to change. You can get the list of available fields by running `beet fields`. To conditionally filter a field, use `field: [regexp, regexp]` to specify regular expressions.

For example:

```
zero:
    fields: month day genre comments
    comments: [EAC, LAME, from.+collection, 'ripped by']
    genre: [rnb, 'power metal']
```

If a custom pattern is not defined for a given field, the field will be nulled unconditionally.

Note that the plugin currently does not zero fields when importing "as-is".

IHate Plugin

The `ihate` plugin allows you to automatically skip things you hate during import or warn you about them. You specify queries (see *Queries*) and the plugin skips (or warns about) albums or items that match any query.

To use the plugin, first enable it in your configuration (see *Using Plugins*). Then, add an `ihate:` section to your configuration file:

```
ihate:
    # Print a warning message for these.
    warn:
        - artist:rnb
        - genre: soul
        # Only warn about tribute albums in rock genre.
        - genre:rock album:tribute
    # Never import any of this.
    skip:
        - genre::russian\srock
        - genre:polka
        - artist:manowar
        - album:christmas
```

The plugin trusts your decision in "as-is" imports.

Convert Plugin

The `convert` plugin lets you convert parts of your collection to a directory of your choice, transcoding audio and embedding album art along the way. It can transcode to and from any format using a configurable command line.

Installation

Enable the `convert` plugin in your configuration (see *Plugins*). By default, the plugin depends on FFmpeg to transcode the audio, so you might want to install it.

Usage

To convert a part of your collection, run `beet convert QUERY`. This will display all items matching `QUERY` and ask you for confirmation before starting the conversion. The command will then transcode all the matching files to the destination directory given by the `-d` (`--dest`) option or the `dest` configuration. The path layout mirrors that of your library, but it may be customized through the `paths` configuration.

The plugin uses a command-line program to transcode the audio. With the `-f` (`--format`) option you can choose the transcoding command and customize the available commands *through the configuration*.

The `-a` (or `--album`) option causes the command to match albums instead of tracks.

By default, the command places converted files into the destination directory and leaves your library pristine. To instead back up your original files into the destination directory and keep converted files in your library, use the `-k` (or `--keep-new`) option.

To test your configuration without taking any actions, use the `--pretend` flag. The plugin will print out the commands it will run instead of executing them.

Configuration

The plugin offers several configuration options, all of which live under the `convert:` section:

- `dest` sets the directory the files will be converted (or copied) to. A destination is required—you either have to provide it in the config file or on the command-line using the `-d` flag.

- `embed` indicates whether or not to embed album art in converted items. Default: true.
- If you set `max_bitrate`, all lossy files with a higher bitrate will be transcoded and those with a lower bitrate will simply be copied. Note that this does not guarantee that all converted files will have a lower bitrate—that depends on the encoder and its configuration.
- `auto` gives you the option to import transcoded versions of your files automatically during the `import` command. With this option enabled, the importer will transcode all non-MP3 files over the maximum bitrate before adding them to your library.
- `quiet` mode prevents the plugin from announcing every file it processes. Default: false.
- `paths` lets you specify the directory structure and naming scheme for the converted files. Use the same format as the top-level `paths` section (see *Path Format Configuration*). By default, the plugin reuses your top-level path format settings.
- Finally, `threads` determines the number of threads to use for parallel encoding. By default, the plugin will detect the number of processors available and use them all.

Configuring the transcoding command You can customize the transcoding command through the `formats` map and select a command with the `--format` command-line option or the `format` configuration.:

```
convert:
    format: speex
    formats:
        speex:
            command: ffmpeg -i $source -y -acodec speex $dest
            extension: spx
        wav: ffmpeg -i $source -y -acodec pcm_s16le $dest
```

In this example `beet convert` will use the *speex* command by default. To convert the audio to *wav*, run `beet convert -f wav`. This will also use the format key (*wav*) as the file extension.

Each entry in the `formats` map consists of a key (the name of the format) as well as the command and the possibly the file extension. `extension` is the filename extension to be used for newly transcoded files. If only the command is given as a string, the file extension defaults to the format's name. `command` is the command-line to use to transcode audio. The tokens `$source` and `$dest` in the command are replaced with the paths to the existing and new file.

The plugin in comes with default commands for the most common audio formats: *mp3*, *alac*, *flac*, *aac*, *opus*, *ogg*, *wmv*. For details have a look at the output of `beet config -d`.

For a one-command-fits-all solution use the `convert.command` and `convert.extension` options. If these are set the formats are ignored and the given command is used for all conversions.:

```
convert:
    command: ffmpeg -i $source -y -vn -aq 2 $dest
    extension: mp3
```

Info Plugin

The `info` plugin provides a command that dumps the current tag values for any file format supported by beets. It works like a supercharged version of mp3info or id3v2.

Enable the plugin and then type:

```
$ beet info /path/to/music.flac
```

and the plugin will enumerate all the tags in the specified file. It also accepts multiple filenames in a single command-line.

Play Plugin

The `play` plugin allows you to pass the results of a query to a music player in the form of an m3u playlist.

To use the plugin, enable it in your configuration (see *Using Plugins*). Then use it by invoking the `beet play` command with a query. The command will create a temporary m3u file and open it using an appropriate application. You can query albums instead of tracks using the `-a` option.

By default, the playlist is opened using the `open` command on OS X, `xdg-open` on other Unixes, and `start` on Windows. To configure the command, you can use a `play:` section in your configuration file:

```
play:
    command: /Applications/VLC.app/Contents/MacOS/VLC
```

You can also specify additional space-separated options to command (like you would on the command-line):

```
play:
    command: /usr/bin/command --option1 --option2 some_other_option
```

When using the `-a` option, the m3u will have the paths to each track on the matched albums. If you wish to have folders instead, you can change that by setting `use_files: False` in your configuration file.

Enable beets' verbose logging to see the command's output if you need to debug.

Smart Playlist Plugin

`smartplaylist` is a plugin to generate smart playlists in m3u format based on beets queries every time your library changes. This plugin is specifically created to work well with MPD's playlist functionality.

To use it, enable the plugin by putting `smartplaylist` in the `plugins` section in your `config.yaml`. Then configure your smart playlists like the following example:

```
smartplaylist:
    relative_to: ~/Music
    playlist_dir: ~/.mpd/playlists
    playlists:
        - name: all.m3u
          query: ''

        - name: beatles.m3u
          query: 'artist:Beatles'
```

If you intend to use this plugin to generate playlists for MPD, you should set `relative_to` to your MPD music directory (by default, `relative_to` is `None`, and the absolute paths to your music files will be generated).

`playlist_dir` is where the generated playlist files will be put.

You can generate as many playlists as you want by adding them to the `playlists` section, using beets query syntax (see *Queries*) for `query` and the file name to be generated for `name`. The query will be split using shell-like syntax, so if you need to use spaces in the query, be sure to quote them (e.g., `artist:"The Beatles"`). If you have existing files with the same names, you should back them up—they will be overwritten when the plugin runs.

For more advanced usage, you can use template syntax (see *Path Formats*) in the `name` field. For example:

```
- name: 'ReleasedIn$year.m3u'
  query: 'year::201(0|1)'
```

This will query all the songs in 2010 and 2011 and generate the two playlist files *ReleasedIn2010.m3u* and *ReleasedIn2011.m3u* using those songs.

You can also gather the results of several queries by putting them in a list. (Items that match both queries are not included twice.) For example:

```
- name: 'BeatlesUniverse.m3u'
  query: ['artist:beatles', 'genre:"beatles cover"']
```

For querying albums instead of items (mainly useful with extensible fields), use the `album_query` field. `query` and `album_query` can be used at the same time. The following example gathers single items but also items belonging to albums that have a `for_travel` extensible field set to 1:

```
- name: 'MyTravelPlaylist.m3u'
  album_query: 'for_travel:1'
  query: 'for_travel:1'
```

By default, all playlists are automatically regenerated after every beets command that changes the library database. This can be disabled by specifying `auto: no`. To force regeneration, you can invoke it manually from the command line:

```
$ beet splupdate
```

which will generate your new smart playlists.

You can also use this plugin together with the *MPDUpdate Plugin*, in order to automatically notify MPD of the playlist change, by adding `mpdupdate` to the `plugins` line in your config file *after* the `smartplaylist` plugin.

MBSync Plugin

This plugin provides the `mbsync` command, which lets you fetch metadata from MusicBrainz for albums and tracks that already have MusicBrainz IDs. This is useful for updating tags as they are fixed in the MusicBrainz database, or when you change your mind about some config options that change how tags are written to files. If you have a music library that is already nicely tagged by a program that also uses MusicBrainz like Picard, this can speed up the initial import if you just import "as-is" and then use `mbsync` to get up-to-date tags that are written to the files according to your beets configuration.

Usage

Enable the plugin and then run `beet mbsync QUERY` to fetch updated metadata for a part of your collection (or omit the query to run over your whole library).

This plugin treats albums and singletons (non-album tracks) separately. It first processes all matching singletons and then proceeds on to full albums. The same query is used to search for both kinds of entities.

The command has a few command-line options:

- To preview the changes that would be made without applying them, use the `-p` (`--pretend`) flag.
- By default, files will be moved (renamed) according to their metadata if they are inside your beets library directory. To disable this, use the `-M` (`--nomove`) command-line option.
- If you have the *import.write* configuration option enabled, then this plugin will write new metadata to files' tags. To disable this, use the `-W` (`--nowrite`) option.

Missing Plugin

This plugin adds a new command, `missing` or `miss`, which finds and lists, for every album in your collection, which or how many tracks are missing. Listing missing files requires one network call to MusicBrainz. Merely counting missing files avoids any network calls.

Installation

Enable the plugin by putting `missing` on your `plugins` line in *config file*:

```
plugins:
    missing
    ...
```

Configuration

By default, the `beet missing` command lists the names of tracks that your library is missing from each album. You can customize the output format, count the number of missing tracks per album, or total up the number of missing tracks over your whole library. These options can either be specified in the config file:

```
missing:
    format: $albumartist - $album - $title
    count: no
    total: no
```

or on the command-line:

```
-f FORMAT, --format=FORMAT
                      print with custom FORMAT
-c, --count           count missing tracks per album
-t, --total           count total of missing tracks
```

format The `format` option (default: *list_format_item*) lets you specify a specific format with which to print every track. This uses the same template syntax as beets' *path formats*. The usage is inspired by, and therefore similar to, the *list* command.

count The `count` option (default: false) prints a count of missing tracks per album, with `format` defaulting to `$albumartist - $album: $missing`.

total The `total` option (default: false) prints a single count of missing tracks in all albums

Template Fields

With this plugin enabled, the `$missing` template field expands to the number of tracks missing from each album.

Examples

List all missing tracks in your collection:

```
beet missing
```

List all missing tracks from 2008:

```
beet missing year:2008
```

Print out a unicode histogram of the missing track years using spark:

```
beet missing -f '$year' | spark
```

Print out a listing of all albums with missing tracks, and respective counts:

```
beet missing -c
```

Print out a count of the total number of missing tracks:

```
beet missing -t
```

Call this plugin from other beet commands:

```
beet ls -a -f '$albumartist - $album: $missing'
```

TODO

- Add caching.

Duplicates Plugin

This plugin adds a new command, `duplicates` or `dup`, which finds and lists duplicate tracks or albums in your collection.

Installation

Enable the plugin by putting `duplicates` on your `plugins` line in your *config file*:

```
plugins: duplicates
```

Configuration

By default, the `beet duplicates` command lists the names of tracks in your library that are duplicates. It assumes that Musicbrainz track and album ids are unique to each track or album. That is, it lists every track or album with an ID that has been seen before in the library.

You can customize the output format, count the number of duplicate tracks or albums, and list all tracks that have duplicates or just the duplicates themselves. These options can either be specified in the config file:

```
duplicates:
    checksum: no
    copy: no
    keys: mb_trackid album
    album: no
    count: no
    delete: no
    format: "$albumartist - $album - $title"
    full: no
    move: no
    path: no
    tag: no
```

or on the command-line:

```
-h, --help            show this help message and exit
-f FMT, --format=FMT  print with custom format
-a, --album           show duplicate albums instead of tracks
-c, --count           count duplicate tracks or albums
-C PROG, --checksum=PROG
                      report duplicates based on arbitrary command
-d, --delete          delete items from library and disk
-F, --full            show all versions of duplicate tracks or albums
-k, --keys            report duplicates based on keys
-m DEST, --move=DEST  move items to dest
-o DEST, --copy=DEST  copy items to dest
-p, --path            print paths for matched items or albums
-t TAG, --tag=TAG     tag matched items with 'k=v' attribute
```

format The `format` option (default: *list_format_item*) lets you specify a specific format with which to print every track or album. This uses the same template syntax as beets' *path formats*. The usage is inspired by, and therefore similar to, the *list* command.

path Convenience wrapper for `-f \$path`.

count The `count` option (default: false) prints a count of duplicate tracks or albums, with `format` hard-coded to `$albumartist - $album - $title: $count` or `$albumartist - $album: $count` (for the `-a` option).

album The `album` option (default: false) lists duplicate albums instead of tracks.

full The `full` option (default: false) lists every track or album that has duplicates, not just the duplicates themselves.

keys The `keys` option (default: `[mb_trackid, mb_albumid]`) defines in which track or album fields duplicates are to be searched. By default, the plugin uses the musicbrainz track and album IDs for this purpose. Using the `keys` option (as a YAML list in the configuration file, or as space-delimited strings in the command-line), you can extend this behavior to consider other attributes.

checksum The `checksum` option (default: `ffmpeg -i {file} -f crc -`) enables the use of any arbitrary command to compute a checksum of items. It overrides the `keys` option the first time it is run; however, because it caches the resulting checksums as `flexattrs` in the database, you can use `--keys=name_of_the_checksumming_program any_other_keys` the second time around.

copy The `copy` option (default: `no`) takes a destination base directory into which it will copy matched items.

move The `move` option (default: `no`) takes a destination base directory into which it will move matched items.

delete The `delete` option (default: `no`) removes matched items from the library and from the disk.

tag The `tag` option (default: `no`) takes a `key=value` string, and adds a new `key` attribute with `value` value as a flexattr to the database.

Examples

List all duplicate tracks in your collection:

```
beet duplicates
```

List all duplicate tracks from 2008:

```
beet duplicates year:2008
```

Print out a unicode histogram of duplicate track years using spark:

```
beet duplicates -f '$year' | spark
```

Print out a listing of all albums with duplicate tracks, and respective counts:

```
beet duplicates -ac
```

The same as the above but include the original album, and show the path:

```
beet duplicates -acf '$path'
```

Get tracks with the same title, artist, and album:

```
beet duplicates -k title albumartist album
```

Compute Adler CRC32 or MD5 checksums, storing them as flexattrs, and report back duplicates based on those values:

```
beet dup -C 'ffmpeg -i {file} -f crc -'
beet dup -C 'md5sum {file}'
```

Copy highly danceable items to `party` directory:

```
beet dup --copy /tmp/party
```

Move likely duplicates to `trash` directory:

```
beet dup --move ${HOME}/.Trash
```

Delete items (careful!), if they're Nickelback:

```
beet duplicates --delete --keys albumartist albumartist:nickelback
```

Tag duplicate items with some flag:

```
beet duplicates --tag dup=1
```

TODO

- better duplicate disambiaguation strategies (eg, based on bitrate, etc)

Discogs Plugin

The `discogs` plugin extends the autotagger's search capabilities to include matches from the Discogs database.

Installation

First, enable the `discogs` plugin (see *Plugins*). Then, install the discogs-client library by typing:

```
pip install discogs-client
```

That's it! Matches from Discogs will now show up during import alongside matches from MusicBrainz.

If you have a Discogs ID for an album you want to tag, you can also enter it at the "enter Id" prompt in the importer.

Beatport Plugin

Warning: As of October 2013, Beatport has closed their API. We've contacted them to attempt to gain access as a "partner." Until this happens, though, this plugin won't work.

The `beatport` plugin adds support for querying the Beatport catalogue during the autotagging process. This can potentially be helpful for users whose collection includes a lot of diverse electronic music releases, for which both MusicBrainz and (to a lesser degree) Discogs show no matches.

Installation

To see matches from the `beatport` plugin, you first have to enable it in your configuration (see *Plugins*). Then, install the requests library (which we need for querying the Beatport API) by typing:

```
pip install requests
```

And you're done. Matches from Beatport should now show up alongside matches from MusicBrainz and other sources.

If you have a Beatport ID or a URL for a release or track you want to tag, you can just enter one of the two at the "enter Id" prompt in the importer.

FromFilename Plugin

The `fromfilename` plugin helps to tag albums that are missing tags altogether but where the filenames contain useful information like the artist and title.

When you attempt to import a track that's missing a title, this plugin will look at the track's filename and guess its track number, title, and artist. These will be used to search in MusicBrainz and match track ordering.

To use the plugin, just enable it by putting `fromfilename` on the `plugins:` line in your config file. There are currently no configuration options.

FtInTitle Plugin

The `ftintitle` plugin automatically moved "featured" artists from the `artist` field to the `title` field.

According to MusicBrainz style, featured artists are part of the artist field. That means that, if you tag your music using MusicBrainz, you'll have tracks in your library like "Tellin' Me Things" by the artist "Blakroc feat. RZA". If you prefer to tag this as "Tellin' Me Things feat. RZA" by "Blakroc", then this plugin is for you.

To use the plugin, just enable it and run the command:

```
$ beet ftintitle [QUERY]
```

The query is optional; if it's left off, the transformation will be applied to your entire collection.

If you prefer to remove featured artists entirely instead of adding them to the title field, either use the `-d` flag to the command or set the `ftintitle.drop` config option.

Key Finder Plugin

The *keyfinder* plugin uses the KeyFinder program to detect the musical key of track from its audio data and store it in the *initial_key* field of you database. If enabled, it does so automatically when importing music or through the `beet keyfinder [QUERY]` command.

There are a couple of configuration options to customize the behavior of the plugin. By default they are:

```
keyfinder:
    bin: KeyFinder
    auto: yes
    overwrite: no
```

- `bin`: The name of the *KeyFinder* program on your system or a path to the binary. If you installed the KeyFinder GUI on a Mac, for example, you want something like `/Applications/KeyFinder.app/Contents/MacOS/KeyFinder`.
- `auto`: If set to *yes*, the plugin will analyze every file on import. Otherwise, you need to use the `beet keyfinder` command explicitly.
- `overwrite`: If set to *no*, the import hook and the command will skip any file that already has an *initial_key* in the database.

Bucket Plugin

The `bucket` plugin groups your files into buckets folders representing *ranges*. This kind of organization can classify your music by periods of time (e.g,. *1960s*, *1970s*, etc.), or to divide overwhelmingly large folders into smaller subfolders by grouping albums or artists alphabetically (e.g., *A-F*, *G-M*, *N-Z*).

To use the plugin, enable `bucket` in your configuration file (see *Using Plugins*). The plugin provides a *template function* called `%bucket` for use in path format expressions:

```
paths:
    default: /%bucket{$year}/%bucket{$artist}/$albumartist-$album-$year
```

Then, define your ranges in the `bucket:` section of the config file:

```
bucket:
    bucket_alpha: ['A-F', 'G-M', 'N-Z']
    bucket_year:  ['1980s', '1990s', '2000s']
```

The `bucket_year` parameter is used for all substitutions occuring on the `$year` field, while `bucket_alpha` takes care of textual fields.

The definition of a range is somewhat loose, and multiple formats are allowed:

- For alpha ranges: the range is defined by the lowest and highest (ASCII-wise) alphanumeric characters in the string you provide. For example, *ABCD*, *A-D*, *A->D*, and *[AD]* are all equivalent.
- For year ranges: digits characters are extracted and the two extreme years define the range. For example, *1975-77*, *1975,76,77* and *1975-1977* are equivalent. If no upper bound is given, the range is extended to current year (unless a later range is defined). For example, *1975* encompasses all years from 1975 until now.

If you want to group your files into multiple year ranges, you don't have to enumerate them all in *bucket_year* parameter but can activate the `extrapolate` option instead. This option will generate year bucket names by reproducing characteristics of declared buckets:

```
bucket:
    bucket_year: ['2000-05']
    extrapolate: true
```

The above configuration creates five-year ranges for any input year.

If the automatic range of an alpha bucket is not sufficient an overriding regular expression can be used:

```
bucket:
   bucket_alpha: ['A - D', 'E - L', 'M - R', 'S - Z']
   bucket_alpha_regex:
     'A - D': ^[0-9a-dA-D...äÄ]
```

The *A - D* bucket now matches also all artists starting with ä or Ä and 0 to 9 and ... (three dots). The other buckets work as ranges (see above).

ImportAdded Plugin

The `importadded` plugin is useful when an existing collection is imported and the time when albums and items were added should be preserved.

The MTIME (modification time) of files that are imported into the library are assumed to represent the time when the items were originally added.

The `item.added` field is populated as follows:

- For singleton items with no album, `item.added` is set to the item's file mtime before it was imported.
- For items that are part of an album, `album.added` and `item.added` is set to the oldest mtime of the files in the album before they were imported. The mtime of album directories are ignored.

This plugin can optionally be configured to also preserve mtimes:

```
importadded:
    preserve_mtimes: yes # default: no
```

File modification times are preserved as follows:

- For all items:
  - `item.mtime` is set to the mtime of the file from which the item is imported from.
  - The mtime of the file `item.path` is set to `item.mtime`.

Note that there is no `album.mtime` field in the database and that the mtime of album directories on disk aren't preserved.

BPM Plugin

This `bpm` plugin lets you to get the tempo (beats per minute) of a song by tapping out the beat on your keyboard.

Usage

First, enable the plugin `bpm` as described in *Plugins*. Then, play a song you want to measure in your favorite media player and type:

```
beet bpm <song>
```

You'll be prompted to press Enter three times to the rhythm. This typically allows to determine the BPM within 5% accuracy.

The plugin works best if you wrap it in a script that gets the playing song. for instance, with `mpc` you can do something like:

```
beet bpm $(mpc |head -1|tr -d "-")
```

Credit

This plugin is inspired by a similar feature present in the Banshee media player.

Spotify Plugin

The `spotify` plugin generates Spotify playlists from tracks in your library. Using the Spotify Web API, any tracks that can be matched with a Spotify ID are returned, and the results can be either pasted in to a playlist or opened directly in the Spotify app.

Why Use This Plugin?

- You're a Beets user and Spotify user already.
- You have playlists or albums you'd like to make available in Spotify from Beets without having to search for each artist/album/track.
- You want to check which tracks in your library are available on Spotify.

Basic Usage

First, enable the plugin (see *Using Plugins*). Then, use the `spotify` command with a beets query:

```
beet spotify [OPTIONS...] QUERY
```

Here's an example:

```
$ beet spotify "In The Lonely Hour"
Processing 14 tracks...
http://open.spotify.com/track/19w0OHr8SiZzRhjpnjctJ4
http://open.spotify.com/track/3PRLM4FzhplXfySa4B7bxS
[...]
```

Command-line options include:

- `-m MODE` or `--mode=MODE` where `MODE` is either "list" or "open" controls whether to print out the playlist (for copying and pasting) or open it in the Spotify app. (See below.)
- `--show-failures` or `-f`: List the tracks that did not match a Spotify ID.

Configuring

The default options should work as-is, but there are some options you can put in config.yaml:

- `mode`: See the section below on modes.
- `region_filter`: Use the 2-character country abbreviation to limit results to that market.
- `show_failures`: Show the artist/album/track for each lookup that does not return a Spotify ID (and therefore cannot be added to a playlist).
- `tiebreak`: How to choose the track if there is more than one identical result. For example, there might be multiple releases of the same album. Currently, this defaults to "popularity", "first" simply chooses the first in the list returned by Spotify.
- `regex`: An array of regex transformations to perform on the track/album/artist fields before sending them to Spotify. Can be useful for changing certain abbreviations, like ft. -> feat. See the examples below.
- `artist_field` / `album_field` / `track_field`: These allow the user to choose a different field to send to Spotify when looking up the track, album and artist. Most users will not want to change this.

Example Configuration

```
spotify:
    # Default is list, shows the plugin output. Open attempts to open
    # directly in Spotify (only tested on Mac).
    mode: "open"

    # Filter tracks by only that market (2-letter code)
    region_filter: "US"

    # Display the tracks that did not match a Spotify ID.
    show_faiulres: on

    # Need to break ties when then are multiple tracks.  Default is
    popularity.
    tiebreak: "first"

    # Which beets fields to use for lookups.
    artist_field: "albumartist"
    album_field: "album"
    track_field: "title"

    regex: [
        {
            field: "albumartist", # Field in the item object to regex.
            search: "Something", # String to look for.
            replace: "Replaced" # Replacement value.
        },
        {
            field: "title",
            search: "Something Else",
            replace: "AlsoReplaced"
        }
    ]
```

Spotify Plugin Modes

- `list`: The default mode is to print out the playlist as a list of links. This list can then be pasted in to a new or existing Spotify playlist.
- `open`: This mode actually sends a link to your default browser with instructions to open Spotify with the playlist you created. Until this has been tested on all platforms, it will remain optional.

1.3.2 Autotagger Extensions

- *Chromaprint/Acoustid Plugin*: Use acoustic fingerprinting to identify audio files with missing or incorrect metadata.
- *Discogs Plugin*: Search for releases in the Discogs database.
- *FromFilename Plugin*: Guess metadata for untagged tracks from their filenames.

1.3.3 Metadata

- *Lyrics Plugin*: Automatically fetch song lyrics.
- *Echo Nest Plugin*: Automatically fetch acoustic attributes from the Echo Nest (tempo, energy, danceability, ...).
- *LastGenre Plugin*: Fetch genres based on Last.fm tags.
- *MBSync Plugin*: Fetch updated metadata from MusicBrainz
- *FetchArt Plugin*: Fetch album cover art from various sources.
- *EmbedArt Plugin*: Embed album art images into files' metadata.
- *ReplayGain Plugin*: Calculate volume normalization for players that support it.
- *Scrub Plugin*: Clean extraneous metadata from music files.
- *Zero Plugin*: Nullify fields by pattern or unconditionally.
- *FtInTitle Plugin*: Move "featured" artists from the artist field to the title field.
- *MPDStats Plugin*: Connect to MPD and update the beets library with play statistics (last_played, play_count, skip_count, rating).
- *Key Finder Plugin*: Use the KeyFinder program to detect the musical key from the audio.
- *ImportAdded Plugin*: Use file modification times for guessing the value for the *added* field in the database.
- *BPM Plugin*: Determine bpm from keystrokes

1.3.4 Path Formats

- *Inline Plugin*: Use Python snippets to customize path format strings.
- *Rewrite Plugin*: Substitute values in path formats.
- *The Plugin*: Move patterns in path formats (i.e., move "a" and "the" to the end).
- *Bucket Plugin*: Group your files into bucket directories that cover different field values ranges.

1.3.5 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.
- *Smart Playlist Plugin*: Generate smart playlists based on beets queries.
- *Play Plugin*: Play beets queries in your music player.

1.3.6 Miscellaneous

- *Web Plugin*: An experimental Web-based GUI for beets.
- *Random Plugin*: Randomly choose albums and tracks from your library.
- *Fuzzy Search Plugin*: Search albums and tracks with fuzzy string matching.
- *MusicBrainz Collection Plugin*: Maintain your MusicBrainz collection list.
- *IHate Plugin*: Automatically skip albums and tracks during the import process.
- *BPD Plugin*: A music player for your beets library that emulates MPD and is compatible with MPD clients.
- *Convert Plugin*: Transcode music and embed album art while exporting to a different directory.
- *Info Plugin*: Print music files' tags to the console.
- *Missing Plugin*: List missing tracks.
- *Duplicates Plugin*: List duplicate tracks or albums.
- *Spotify Plugin*: Create Spotify playlists from the Beets library

1.3.7 Other Plugins

In addition to the plugins that come with beets, there are several plugins that are maintained by the beets community. To use an external plugin, there are two options for installation:

- 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 *pluginpath* config variable to point to the directory containing the plugin. (See *Configuration*.)

Once the plugin is installed, enable it by placing its name on the *plugins* line in your config file.

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.
- beets-artistcountry fetches the artist's country of origin from MusicBrainz.
- dsedivec has two plugins: `edit` and `moveall`.
- beet-amazon adds Amazon.com as a tagger data source.
- copyartifacts helps bring non-music files along during import.
- beets-check automatically checksums your files to detect corruption.

1.4 FAQ

Here are some answers to frequently-asked questions from IRC and elsewhere. Got a question that isn't answered here? Try IRC, the mailing list, or *filing an issue* in the bug tracker.

1.4.1 How do I...

...rename my files according to a new path format configuration?

Just run the *move* command. Use a *query* to rename a subset of your music or leave the query off to rename everything.

...find all the albums I imported "as-is"?

Enable the *import log* to automatically record whenever you skip an album or accept one "as-is".

Alternatively, you can find all the albums in your library that are missing MBIDs using a command like this:

```
beet ls -a mb_albumid::^$
```

Assuming your files didn't have MBIDs already, then this will roughly correspond to those albums that didn't get autotagged.

...create "Disc N" directories for multi-disc albums?

Use the *Inline Plugin* along with the `%if{}` function to accomplish this:

```
plugins: inline
paths:
    default: $albumartist/$album%aunique{}/%if{$multidisc,Disc $disc/}$track $title
```

```
item_fields:
    multidisc: 1 if disctotal > 1 else 0
```

. . . import a multi-disc album?

As of 1.0b11, beets tags multi-disc albums as a *single unit*. To get a good match, it needs to treat all of the album's parts together as a single release.

To help with this, the importer uses a simple heuristic to guess when a directory represents a multi-disc album that's been divided into multiple subdirectories. When it finds a situation like this, it collapses all of the items in the subdirectories into a single release for tagging.

The heuristic works by looking at the names of directories. If multiple subdirectories of a common parent directory follow the pattern "(title) disc (number) (...)" and the *prefix* (everything up to the number) is the same, the directories are collapsed together. One of the key words "disc" or "CD" must be present to make this work.

If you have trouble tagging a multi-disc album, consider the `--flat` flag (which treats a whole tree as a single album) or just putting all the tracks into a single directory to force them to be tagged together.

. . . enter a MusicBrainz ID?

An MBID looks like one of these:

- `http://musicbrainz.org/release/ded77dcf-7279-457e-955d-625bd3801b87`
- `d569deba-8c6b-4d08-8c43-d0e5a1b8c7f3`

Beets can recognize either the hex-with-dashes UUID-style string or the full URL that contains it (as of 1.0b11).

You can get these IDs by searching on the MusicBrainz web site and going to a *release* page (when tagging full albums) or a *recording* page (when tagging singletons). Then, copy the URL of the page and paste it into beets.

Note that MusicBrainz has both "releases" and "release groups," which link together different versions of the same album. Use *release* IDs here.

. . . upgrade to the latest version of beets?

Run a command like this:

```
pip install -U beets
```

The `-U` flag tells pip to upgrade beets to the latest version. If you want a specific version, you can specify with using `==` like so:

```
pip install beets==1.0rc2
```

. . . run the latest source version of beets?

Beets sees regular releases (about every six weeks or so), but sometimes it's helpful to run on the "bleeding edge". To run the latest source:

1. Uninstall beets. If you installed using `pip`, you can just run `pip uninstall beets`.
2. Install from source. There are a few easy ways to do this:
    - Use `pip` to install the latest snapshot tarball: just type `pip install https://github.com/sampsyo/beets/tarball/master`.

- Grab the source using Mercurial (`hg clone https://bitbucket.org/adrian/beets`) or git (`git clone https://github.com/sampsyo/beets.git`). Then `cd beets` and type `python setup.py install`.
- Use `pip` to install an "editable" version of beets based on an automatic source checkout. For example, run `pip install -e hg+https://bitbucket.org/adrian/beets#egg=beets` to clone beets from BitBucket using Mercurial and install it, allowing you to modify the source in-place to try out changes.

More details about the beets source are available on the *developer documentation* pages.

. . . report a bug in beets?

We use the issue tracker on GitHub. Enter a new issue there to report a bug. Please follow these guidelines when reporting an issue:

- Most importantly: if beets is crashing, please include the traceback. Tracebacks can be more readable if you put them in a pastebin (e.g., Gist or Hastebin), especially when communicating over IRC or email.
- Turn on beets' debug output (using the -v option: for example, `beet -v import ...`) and include that with your bug report. Look through this verbose output for any red flags that might point to the problem.
- If you can, try installing the latest beets source code to see if the bug is fixed in an unreleased version. You can also look at the *latest changelog entries* for descriptions of the problem you're seeing.
- Try to narrow your problem down to something specific. Is a particular plugin causing the problem? (You can disable plugins to see whether the problem goes away.) Is a some music file or a single album leading to the crash? (Try importing individual albums to determine which one is causing the problem.) Is some entry in your configuration file causing it? Et cetera.
- If you do narrow the problem down to a particular audio file or album, include it with your bug report so the developers can run tests.

If you've never reported a bug before, Mozilla has some well-written general guidelines for good bug reports.

. . . find the configuration file (config.yaml)?

You create this file yourself; beets just reads it. See *Configuration*.

. . . avoid using special characters in my filenames?

Use the `%asciify{}` function in your path formats. See *Template Functions*.

1.4.2 Why does beets. . .

. . . complain that it can't find a match?

There are a number of possibilities:

- First, make sure the album is 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.)
- If the album in question is a multi-disc release, see the relevant FAQ answer above.
- The music files' metadata might be insufficient. Try using the "enter search" or "enter ID" options to help the matching process find the right MusicBrainz entry.

- If you have a lot of files that are missing metadata, consider using *acoustic fingerprinting* or *filename-based guesses* for that music.

If none of these situations apply and you're still having trouble tagging something, please *file a bug report*.

. . . appear to be missing some plugins?

Please make sure you're using the latest version of beets—you might be using a version earlier than the one that introduced the plugin. In many cases, the plugin may be introduced in beets "trunk" (the latest source version) and might not be released yet. Take a look at *the changelog* to see which version added the plugin. (You can type `beet version` to check which version of beets you have installed.)

If you want to live on the bleeding edge and use the latest source version of beets, you can check out the source (see the next question).

To see the beets documentation for your version (and avoid confusion with new features in trunk), select your version from the left-hand sidebar (or the buttons at the bottom of the window).

. . . ignore control-C during an import?

Typing a ^C (control-C) control sequence will not halt beets' multithreaded importer while it is waiting at a prompt for user input. Instead, hit "return" (dismissing the prompt) after typing ^C. Alternatively, just type a "b" for "aBort" at most prompts. Typing ^C *will* work if the importer interface is between prompts.

Also note that beets may take some time to quit after ^C is typed; it tries to clean up after itself briefly even when canceled.

(For developers: this is because the UI thread is blocking on `raw_input` and cannot be interrupted by the main thread, which is trying to close all pipeline stages in the exception handler by setting a flag. There is no simple way to remedy this.)

. . . not change my ID3 tags?

Beets writes ID3v2.4 tags by default. Some software, including Windows (i.e., Windows Explorer and Windows Media Player) and id3lib/id3v2, don't support v2.4 tags. When using 2.4-unaware software, it might look like the tags are unmodified or missing completely.

To enable ID3v2.3 tags, enable the *id3v23* config option.

. . . complain that a file is "unreadable"?

Beets will log a message like "unreadable file: /path/to/music.mp3" when it encounters files that *look* like music files (according to their extension) but seem to be broken. Most of the time, this is because the file is corrupted. To check whether the file is intact, try opening it in another media player (e.g., VLC) to see whether it can read the file. You can also use specialized programs for checking file integrity—for example, type `metaflac --list music.flac` to check FLAC files.

If beets still complains about a file that seems to be valid, file a bug and we'll look into it. There's always a possibility that there's a bug "upstream" in the Mutagen library used by beets, in which case we'll forward the bug to that project's tracker.

. . . seem to "hang" after an import finishes?

Probably not. Beets uses a *multithreaded importer* that overlaps many different activities: it can prompt you for decisions while, in the background, it talks to MusicBrainz and copies files. This means that, even after you make your last decision, there may be a backlog of files to be copied into place and tags to be written. (Plugin tasks, like looking up lyrics and genres, also run at this time.) If beets pauses after you see all the albums go by, have patience.

. . . put a bunch of underscores in my filenames?

When naming files, beets replaces certain characters to avoid causing problems on the filesystem. For example, leading dots can confusingly hide files on Unix and several non-alphanumeric characters are forbidden on Windows.

The *replace* config option controls which replacements are made. By default, beets makes filenames safe for all known platforms by replacing several patterns with underscores. This means that, even on Unix, filenames are made Windows-safe so that network filesystems (such as SMB) can be used safely.

Most notably, Windows forbids trailing dots, so a folder called "M.I.A." will be rewritten to "M.I.A_" by default. Change the `replace` config if you don't want this behavior and don't need Windows-safe names.

. . . say "command not found"?

You need to put the `beet` program on your system's search path. If you installed using pip, the command `pip show -f beets` can show you where `beet` was placed on your system. If you need help extending your `$PATH`, try this Super User answer.

1.5 For Developers

This section contains information for developers. Read on if you're interested in hacking beets itself or creating plugins for it.

1.5.1 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 `config.yaml` 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, 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) 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 finishes (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`
- *item_copied*: called with an `Item` object whenever its file is copied. Parameters: `item`, `source` path, `destination` path
- *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`
- *before_item_moved*: called with an `Item` object immediately before its file is moved. Parameters: `item`, `source` path, `destination` path
- *item_moved*: called with an `Item` object whenever its file is moved. Parameters: `item`, `source` path, `destination` path
- *item_removed*: called with an `Item` object every time an item (singleton or album's part) is removed from the library (even when its file is not deleted from disk).
- *write*: called with an `Item` object just before a file's metadata is written to disk (i.e., just before the file on disk is opened). Event handlers may raise a `library.FileOperationError` exception to abort the write operation. Beets will catch that exception, print an error message and continue.
- *after_write*: called with an `Item` object after a file's metadata is written to disk (i.e., just after the file on disk is closed).
- *import_task_start*: called when before an import task begins processing. Parameters: `task` (an *ImportTask*) and `session` (an *ImportSession*).
- *import_task_apply*: called after metadata changes have been applied in an import task. Parameters: `task` and `session`.
- *import_task_choice*: called after a decision has been made about an import task. This event can be used to initiate further interaction with the user. Use `task.choice_flag` to determine or change the action to be taken. Parameters: `task` and `session`.
- *import_task_files*: called after an import task finishes manipulating the filesystem (copying and moving files, writing metadata tags). Parameters: `task` and `session`.
- *library_opened*: called after beets starts up and initializes the main Library object. Parameter: `lib`.
- *database_change*: a modification has been made to the library database. The change might not be committed yet. Parameter: `lib`.
- *cli_exit*: called just before the `beet` command-line program exits. Parameter: `lib`.

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 four 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 (an Item object) and `info` is the TrackInfo object that is proposed as a match. Should return a `(dist, dist_max)` pair of floats indicating the distance.
- `album_distance(self, items, album_info, mapping)`: like the above, but compares a list of items (representing an album) to an album-level MusicBrainz entry. `items` is a list of Item objects; `album_info` is an AlbumInfo object; and `mapping` is a dictionary that maps Items to their corresponding TrackInfo objects.
- `candidates(self, items, artist, album, va_likely)`: 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, artist, album)`: given a *singleton* item, return a list of `TrackInfo` objects for candidate tracks to be compared and matched.
- `album_for_id(self, album_id)`: given an ID from user input or an album's tags, return a candidate AlbumInfo object (or None).
- `track_for_id(self, track_id)`: given an ID from user input or a file's tags, return a candidate TrackInfo object (or None).

When implementing these functions, you may want 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 `config.yaml` file. You can read configuration values in two ways. The first is to use *self.config* within your plugin class. This gives you a view onto the configuration values in a section with the same name as your plugin's module. For example, if your plugin is in `greatplugin.py`, then *self.config* will refer to options under the `greatplugin:` section of the config file.

For example, if you have a configuration value called "foo", then users can put this in their `config.yaml`:

```
greatplugin:
    foo: bar
```

To access this value, say `self.config['foo'].get()` at any point in your plugin's code. The *self.config* object is a *view* as defined by the Confit library.

If you want to access configuration values *outside* of your plugin's section, import the *config* object from the *beets* module. That is, just put `from beets import config` at the top of your plugin and access values from there.

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 register new functions by adding them to the `template_funcs` dictionary.

Here's an example:

```
class MyPlugin(BeetsPlugin):
    def __init__(self):
        super(MyPlugin, self).__init__()
        self.template_funcs['initial'] = _tmpl_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 `$name` in templates. To add a new field, add a function that takes an `Item` object to the `template_fields` dictionary on the plugin object. Here's an example that adds a `$disc_and_track` field:

```
class MyPlugin(BeetsPlugin):
    def __init__(self):
        super(MyPlugin, self).__init__()
        self.template_fields['disc_and_track'] = _tmpl_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.

This field works for *item* templates. Similarly, you can register *album* template fields by adding a function accepting an `Album` argument to the `album_template_fields` dict.

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 `MediaFile` class uses `MediaField` descriptors to provide access to file tags. Have a look at the `beets.mediafile` source code to learn how to use this descriptor class. If you have created a descriptor you can add it through your plugins `add_media_field()` method.

`BeetsPlugin.`**`add_media_field`**`(`*name*, *descriptor*`)`

Add a field that is synchronized between media files and items.

When a media field is added `item.write()` will set the name property of the item's MediaFile to `item[name]` and save the changes. Similarly `item.read()` will set `item[name]` to the value of the name property of the media file.

`descriptor` must be an instance of `mediafile.MediaField`.

Here's an example plugin that provides a meaningless new field "foo":

```
class FooPlugin(BeetsPlugin):
    def __init__(self):
        field = mediafile.MediaField(
            mediafile.MP3DescStorageStyle(u'foo')
            mediafile.StorageStyle(u'foo')
        )
```

```
        self.add_media_field('foo', field)

FooPlugin()
item = Item.from_path('/path/to/foo/tag.mp3')
assert item['foo'] == 'spam'

item['foo'] == 'ham'
item.write()
# The "foo" tag of the file is now "ham"
```

Add Import Pipeline Stages

Many plugins need to add high-latency operations to the import workflow. For example, a plugin that fetches lyrics from the Web would, ideally, not block the progress of the rest of the importer. Beets allows plugins to add stages to the parallel import pipeline.

Each stage is run in its own thread. Plugin stages run after metadata changes have been applied to a unit of music (album or track) and before file manipulation has occurred (copying and moving files, writing tags to disk). Multiple stages run in parallel but each stage processes only one task at a time and each task is processed by only one stage at a time.

Plugins provide stages as functions that take two arguments: `config` and `task`, which are `ImportConfig` and `ImportTask` objects (both defined in `beets.importer`). Add such a function to the plugin's `import_stages` field to register it:

```
from beets.plugins import BeetsPlugin
class ExamplePlugin(BeetsPlugin):
    def __init__(self):
        super(ExamplePlugin, self).__init__()
        self.import_stages = [self.stage]
    def stage(self, config, task):
        print('Importing something!')
```

Extend the Query Syntax

You can add new kinds of queries to beets' *query syntax* indicated by a prefix. As an example, beets already supports regular expression queries, which are indicated by a colon prefix—plugins can do the same.

To do so, define a subclass of the `Query` type from the `beets.dbcore.query` module. Then, in the `queries` method of your plugin class, return a dictionary mapping prefix strings to query classes.

One simple kind of query you can extend is the `FieldQuery`, which implements string comparisons on fields. To use it, create a subclass inheriting from that class and override the `value_match` class method. (Remember the `@classmethod` decorator!) The following example plugin declares a query using the `@` prefix to delimit exact string matches. The plugin will be used if we issue a command like `beet ls @something` or `beet ls artist:@something`:

```
from beets.plugins import BeetsPlugin
from beets.dbcore import FieldQuery

class ExactMatchQuery(FieldQuery):
    @classmethod
    def value_match(self, pattern, val):
        return pattern == val

class ExactMatchPlugin(BeetsPlugin):
```

```
    def queries(self):
        return {
            '@': ExactMatchQuery
        }
```

1.5.2 API Documentation

This page describes the internal API of beets' core. It's a work in progress—since beets is an application first and a library second, its API has been mainly undocumented until recently. Please file bugs if you run across incomplete or incorrect docs here.

The `Library` object is the central repository for data in beets. It represents a database containing songs, which are `Item` instances, and groups of items, which are `Album` instances.

The Library Class

class `beets.library.`**Library**(*path*, *directory*[, *path_formats*[, *replacements*]])

A database of music containing songs and albums.

items(*query=None*)

Get a sorted list of `Item` objects matching the given query.

albums(*query=None*)

Get a sorted list of `Album` objects matching the given query.

get_item(*id*)

Fetch an `Item` by its ID. Returns *None* if no match is found.

get_album(*item_or_id*)

Given an album ID or an item associated with an album, return an `Album` object for the album. If no such album exists, returns *None*.

add(*obj*)

Add the `Item` or `Album` object to the library database. Return the object's new id.

add_album(*items*)

Create a new album consisting of a list of items.

The items are added to the database if they don't yet have an ID. Return a new `Album` object. The list items must not be empty.

transaction()

Get a `Transaction` object for interacting directly with the underlying SQLite database.

Transactions

The `Library` class provides the basic methods necessary to access and manipulate its contents. To perform more complicated operations atomically, or to interact directly with the underlying SQLite database, you must use a *transaction*. For example:

```
lib = Library()
with lib.transaction() as tx:
    items = lib.items(query)
    lib.add_album(list(items))
```

class beets.dbcore.db.**Transaction**(*db*)
A context manager for safe, concurrent access to the database. All SQL commands should be executed through a transaction.

mutate(*statement*, *subvals=()*)
Execute an SQL statement with substitution values and return the row ID of the last affected row.

query(*statement*, *subvals=()*)
Execute an SQL statement with substitution values and return a list of rows from the database.

script(*statements*)
Execute a string containing multiple SQL statements.

Model Classes

The two model entities in beets libraries, `Item` and `Album`, share a base class, `Model`, that provides common functionality and ORM-like abstraction.

The fields model classes can be accessed using attributes (dots, as in `item.artist`) or items (brackets, as in `item['artist']`). The `Model` base class provides some methods that resemble *dict* objects.

Model base

class beets.dbcore.**Model**(*db=None*, ***values*)
An abstract object representing an object in the database. Model objects act like dictionaries (i.e., the allow subscript access like `obj['field']`). The same field set is available via attribute access as a shortcut (i.e., `obj.field`). Three kinds of attributes are available:

- **Fixed attributes** come from a predetermined list of field names. These fields correspond to SQLite table columns and are thus fast to read, write, and query.
- **Flexible attributes** are free-form and do not need to be listed ahead of time.
- **Computed attributes** are read-only fields computed by a getter function provided by a plugin.

Access to all three field types is uniform: `obj.field` works the same regardless of whether `field` is fixed, flexible, or computed.

Model objects can optionally be associated with a *Library* object, in which case they can be loaded and stored from the database. Dirty flags are used to track which fields need to be stored.

add(*db=None*)
Add the object to the library database. This object must be associated with a database; you can provide one via the *db* parameter or use the currently associated database.

The object's *id* and *added* fields are set along with any current field values.

clear_dirty()
Mark all fields as *clean* (i.e., not needing to be stored to the database).

evaluate_template(*template*, *for_path=False*)
Evaluate a template (a string or a *Template* object) using the object's fields. If *for_path* is true, then no new path separators will be added to the template.

formatted
A *dict*-like view containing formatted values.

get(*key*, *default=None*)
Get the value for a given key or *default* if it does not exist.

items()
Iterate over (key, value) pairs that this object contains. Computed fields are not included.

keys(*computed=False*)
Get a list of available field names for this object. The *computed* parameter controls whether computed (plugin-provided) fields are included in the key list.

load()
Refresh the object's metadata from the library database.

remove()
Remove the object's associated rows from the database.

store()
Save the object's metadata into the library database.

update(*values*)
Assign all values in the given dict.

Item

class beets.library.**Item**(*db=None*, ***values*)

current_mtime()
Returns the current mtime of the file, rounded to the nearest integer.

destination(*fragment=False*, *basedir=None*, *platform=None*, *path_formats=None*)
Returns the path in the library directory designated for the item (i.e., where the file ought to be). fragment makes this method return just the path fragment underneath the root library directory; the path is also returned as Unicode instead of encoded as a bytestring. basedir can override the library's base directory for the destination.

classmethod from_path(*path*)
Creates a new item from the media file at the specified path.

get_album()
Get the Album object that this item belongs to, if any, or None if the item is a singleton or is not associated with a library.

move(*copy=False*, *basedir=None*, *with_album=True*)
Move the item to its designated location within the library directory (provided by destination()). Subdirectories are created as needed. If the operation succeeds, the item's path field is updated to reflect the new location.

If copy is True, moving the file is copied rather than moved.

basedir overrides the library base directory for the destination.

If the item is in an album, the album is given an opportunity to move its art. (This can be disabled by passing with_album=False.)

The item is stored to the database if it is in the database, so any dirty fields prior to the move() call will be written as a side effect. You probably want to call save() to commit the DB transaction.

move_file(*dest*, *copy=False*)
Moves or copies the item's file, updating the path value if the move succeeds. If a file exists at `dest`, then it is slightly modified to be unique.

read(*read_path=None*)
Read the metadata from the associated file.

If *read_path* is specified, read metadata from that file instead. Updates all the properties in *_media_fields* from the media file.

Raises a *ReadError* if the file could not be read.

remove(*delete=False*, *with_album=True*)
Removes the item. If *delete*, then the associated file is removed from disk. If *with_album*, then the item's album (if any) is removed if it the item was the last in the album.

try_write(*path=None*)
Calls *write()* but catches and logs *FileOperationError* exceptions.

Returns *False* an exception was caught and *True* otherwise.

update(*values*)
Set all key/value pairs in the mapping. If mtime is specified, it is not reset (as it might otherwise be).

write(*path=None*)
Write the item's metadata to a media file.

All fields in *_media_fields* are written to disk according to the values on this object.

Can raise either a *ReadError* or a *WriteError*.

Album

class beets.library.**Album**(*db=None*, ***values*)
Provides access to information about albums stored in a library. Reflects the library's "albums" table, including album art.

art_destination(*image*, *item_dir=None*)
Returns a path to the destination for the album art image for the album. *image* is the path of the image that will be moved there (used for its extension).

The path construction uses the existing path of the album's items, so the album must contain at least one item or item_dir must be provided.

item_dir()
Returns the directory containing the album's first item, provided that such an item exists.

item_keys = ['added', 'albumartist', 'albumartist_sort', 'albumartist_credit', 'album', 'genre', 'year', 'month', 'day',
List of keys that are set on an album's items.

items()
Returns an iterable over the items associated with this album.

move(*copy=False*, *basedir=None*)
Moves (or copies) all items to their destination. Any album art moves along with them. basedir overrides the library base directory for the destination. The album is stored to the database, persisting any modifications to its metadata.

move_art(*copy=False*)
Move or copy any existing album art so that it remains in the same directory as the items.

remove(*delete=False*, *with_items=True*)
Removes this album and all its associated items from the library. If delete, then the items' files are also deleted from disk, along with any album art. The directories containing the album are also removed (recursively) if empty. Set with_items to False to avoid removing the album's items.

set_art(*path*, *copy=True*)
Sets the album's cover art to the image at the given path. The image is copied (or moved) into place, replacing any existing art.

store()
Update the database with the album information. The album's tracks are also updated.

1.5.3 MediaFile

class beets.mediafile.**MediaFile**(*path*, *id3v23=False*)
Represents a multimedia file on disk and provides access to its metadata.

__init__(*path*, *id3v23=False*)
Constructs a new *MediaFile* reflecting the file at path. May throw *UnreadableFileError*.

By default, MP3 files are saved with ID3v2.4 tags. You can use the older ID3v2.3 standard by specifying the *id3v23* option.

classmethod fields()
Get the names of all writable properties that reflect metadata tags (i.e., those that are instances of MediaField).

classmethod readable_fields()
Get all metadata fields: the writable ones from fields() and also other audio properties.

save()
Write the object's tags back to the file.

update(*dict*)
Set all field values from a dictionary.

For any key in *dict* that is also a field to store tags the method retrieves the corresponding value from *dict* and updates the *MediaFile*. If a key has the value *None*, the corresponding property is deleted from the *MediaFile*.

class beets.mediafile.**MediaField**(**styles*, ***kwargs*)
A descriptor providing access to a particular (abstract) metadata field.

__init__(**styles*, ***kwargs*)
Creates a new MediaField.

Parameters

- **styles** – *StorageStyle* instances that describe the strategy for reading and writing the field in particular formats. There must be at least one style for each possible file format.
- **out_type** – the type of the value that should be returned when getting this property.

class beets.mediafile.**StorageStyle**(*key*, *as_type=<type 'unicode'>*, *suffix=None*, *float_places=2*)
A strategy for storing a value for a certain tag format (or set of tag formats). This basic StorageStyle describes simple 1:1 mapping from raw values to keys in a Mutagen file object; subclasses describe more sophisticated translations or format-specific access strategies.

MediaFile uses a StorageStyle via three methods: get(), set(), and delete(). It passes a Mutagen file object to each.

Internally, the StorageStyle implements get() and set() using two steps that may be overridden by subtypes. To get a value, the StorageStyle first calls fetch() to retrieve the value corresponding to a key and then deserialize() to convert the raw Mutagen value to a consumable Python value. Similarly, to set a field, we call serialize() to encode the value and then store() to assign the result into the Mutagen object.

Each StorageStyle type has a class-level *formats* attribute that is a list of strings indicating the formats that the style applies to. MediaFile only uses StorageStyles that apply to the correct type for a given audio file.

delete(*mutagen_file*)
Remove the tag from the file.

deserialize(*mutagen_value*)
Given a raw value stored on a Mutagen object, decode and return the represented value.

fetch(*mutagen_file*)
Retrieve the raw value of for this tag from the Mutagen file object.

formats = ['FLAC', 'OggOpus', 'OggTheora', 'OggSpeex', 'OggVorbis', 'OggFlac', 'APEv2File', 'WavPack', 'Musepac
List of mutagen classes the StorageStyle can handle.

get(*mutagen_file*)
Get the value for the field using this style.

serialize(*value*)
Convert the external Python value to a type that is suitable for storing in a Mutagen file object.

set(*mutagen_file*, *value*)
Assign the value for the field using this style.

store(*mutagen_file*, *value*)
Store a serialized value in the Mutagen file object.

1.6 Changelog

1.6.1 1.3.7 (August 22, 2014)

This release of beets fixes all the bugs, and you can be confident that you will never again find any bugs in beets, ever. It also adds support for plain old AIFF files and adds three more plugins, including a nifty one that lets you measure a song's tempo by tapping out the beat on your keyboard. The importer deals more elegantly with duplicates and you can broaden your cover art search to the entire web with Google Image Search.

The big new features are:

- Support for AIFF files. Tags are stored as ID3 frames in one of the file's IFF chunks. Thanks to Evan Purkhiser for contributing support to Mutagen.
- The new *ImportAdded Plugin* reads files' modification times to set their "added" date. Thanks to Stig Inge Lea Bjørnsen.
- The new *BPM Plugin* lets you manually measure the tempo of a playing song. Thanks to aroquen.
- The new *Spotify Plugin* generates playlists for your Spotify account. Thanks to Olin Gay.
- A new *required* configuration option for the importer skips matches that are missing certain data. Thanks to oprietop.
- When the importer detects duplicates, it now shows you some details about the potentially-replaced music so you can make an informed decision. Thanks to Howard Jones.
- *FetchArt Plugin*: You can now optionally search for cover art on Google Image Search. Thanks to Lemutar.
- A new *asciify_paths* configuration option replaces all non-ASCII characters in paths.

And the multitude of little improvements and fixes:

- Compatibility with the latest version of Mutagen, 1.23.
- *Web Plugin*: Lyrics now display readably with correct line breaks. Also, the detail view scrolls to reveal all of the lyrics. Thanks to Meet Udeshi.

- *Play Plugin*: The `command` config option can now contain arguments (rather than just an executable). Thanks to Alessandro Ghedini.
- Fix an error when using the *modify* command to remove a flexible attribute. Thanks to Pierre Rust.
- *Info Plugin*: The command now shows audio properties (e.g., bitrate) in addition to metadata. Thanks Alessandro Ghedini.
- Avoid a crash on Windows when writing to files with special characters in their names.
- *Play Plugin*: Playing albums now generates filenames by default (as opposed to directories) for better compatibility. The `use_folders` option restores the old behavior. Thanks to Lucas Duailibe.
- Fix an error when importing an empty directory with the `--flat` option.
- *MPDStats Plugin*: The last song in a playlist is now correctly counted as played. Thanks to Johann Klähn.
- *Zero Plugin*: Prevent accidental nulling of dangerous fields (IDs and paths). Thanks to brunal.
- The *remove* command now shows the paths of files that will be deleted. Thanks again to brunal.
- Don't display changes for fields that are not in the restricted field set. This fixes *write* showing changes for fields that are not written to the file.
- The *write* command avoids displaying the item name if there are no changes for it.
- When using both the *Convert Plugin* and the *Scrub Plugin*, avoid scrubbing the source file of conversions. (Fix a regression introduced in the previous release.)
- *ReplayGain Plugin*: Logging is now quieter during import. Thanks to Yevgeny Bezman.
- *FetchArt Plugin*: When loading art from the filesystem, we now prioritize covers with more keywords in them. This means that `cover-front.jpg` will now be taken before `cover-back.jpg` because it contains two keywords rather than one. Thanks to Fabrice Laporte.
- *LastGenre Plugin*: Remove duplicates from canonicalized genre lists. Thanks again to Fabrice Laporte.
- The importer now records its progress when skipping albums. This means that incremental imports will no longer try to import albums again after you've chosen to skip them, and erroneous invitations to resume "interrupted" imports should be reduced. Thanks to jcassette.
- *Bucket Plugin*: You can now customize the definition of alphanumeric "ranges" using regular expressions. And the heuristic for detecting years has been improved. Thanks to sotho.
- Already-imported singleton tracks are skipped when resuming an import.
- *Chromaprint/Acoustid Plugin*: A new `auto` configuration option disables fingerprinting on import. Thanks to ddettrittus.
- *Convert Plugin*: A new `--format` option to can select the transcoding preset from the command-line.
- *Convert Plugin*: Transcoding presets can now omit their filename extensions (extensions default to the name of the preset).
- *Convert Plugin*: A new `--pretend` option lets you preview the commands the plugin will execute without actually taking any action. Thanks to Dietrich Daroch.
- Fix a crash when a float-valued tag field only contained a + or − character.
- Fixed a regression in the core that caused the *Scrub Plugin* not to work in `auto` mode. Thanks to Harry Khanna.
- The *write* command now has a `--force` flag. Thanks again to Harry Khanna.
- *MBSync Plugin*: Track alignment now works with albums that have multiple copies of the same recording. Thanks to Rui Gonçalves.

1.6.2 1.3.6 (May 10, 2014)

This is primarily a bugfix release, but it also brings two new plugins: one for playing music in desktop players and another for organizing your directories into "buckets." It also brings huge performance optimizations to queries—your `beet ls` commands will now go much faster.

New features:

- The new *Play Plugin* lets you start your desktop music player with the songs that match a query. Thanks to David Hamp-Gonsalves.
- The new *Bucket Plugin* provides a `%bucket{}` function for path formatting to generate folder names representing ranges of years or initial letter. Thanks to Fabrice Laporte.
- Item and album queries are much faster.
- *FtInTitle Plugin*: A new option lets you remove featured artists entirely instead of moving them to the title. Thanks to SUTJael.

And those all-important bug fixes:

- *MBSync Plugin*: Fix a regression in 1.3.5 that broke the plugin entirely.
- *Shell completion* now searches more common paths for its `bash_completion` dependency.
- Fix encoding-related logging errors in *Convert Plugin* and *ReplayGain Plugin*.
- *ReplayGain Plugin*: Suppress a deprecation warning emitted by later versions of PyGI.
- Fix a crash when reading files whose iTunes SoundCheck tags contain non-ASCII characters.
- The `%if{}` template function now appropriately interprets the condition as false when it contains the string "false". Thanks to Ayberk Yilmaz.
- *Convert Plugin*: Fix conversion for files that include a video stream by ignoring it. Thanks to brunal.
- *FetchArt Plugin*: Log an error instead of crashing when tag manipulation fails.
- *Convert Plugin*: Log an error instead of crashing when embedding album art fails.
- *Convert Plugin*: Embed cover art into converted files. Previously they were embedded into the source files.
- New plugin event: *before_item_moved*. Thanks to Robert Speicher.

1.6.3 1.3.5 (April 15, 2014)

This is a short-term release that adds some great new stuff to beets. There's support for tracking and calculating musical keys, the ReplayGain plugin was expanded to work with more music formats via GStreamer, we can now import directly from compressed archives, and the lyrics plugin is more robust.

One note for upgraders and packagers: this version of beets has a new dependency in enum34, which is a backport of the new enum standard library module.

The major new features are:

- Beets can now import *zip*, *tar*, and *rar* archives. Just type `beet import music.zip` to have beets transparently extract the files to import.
- *ReplayGain Plugin*: Added support for calculating ReplayGain values with GStreamer as well the mp3gain program. This enables ReplayGain calculation for any audio format. Thanks to Yevgeny Bezman.
- *Lyrics Plugin*: Lyrics should now be found for more songs. Searching is now sensitive to featured artists and parenthesized title suffixes. When a song has multiple titles, lyrics from all the named songs are now concatenated. Thanks to Fabrice Laporte and Paul Phillips.

In particular, a full complement of features for supporting musical keys are new in this release:

- A new *initial_key* field is available in the database and files' tags. You can set the field manually using a command like `beet modify initial_key=Am`.
- The *Echo Nest Plugin* sets the *initial_key* field if the data is available.
- A new *Key Finder Plugin* runs a command-line tool to get the key from audio data and store it in the *initial_key* field.

There are also many bug fixes and little enhancements:

- *Echo Nest Plugin*: Truncate files larger than 50MB before uploading for analysis.
- *FetchArt Plugin*: Fix a crash when the server does not specify a content type. Thanks to Lee Reinhardt.
- *Convert Plugin*: The `--keep-new` flag now works correctly and the library includes the converted item.
- The importer now logs a message instead of crashing when errors occur while opening the files to be imported.
- *EmbedArt Plugin*: Better error messages in exceptional conditions.
- Silenced some confusing error messages when searching for a non-MusicBrainz ID. Using an invalid ID (of any kind—Discogs IDs can be used there too) at the "Enter ID:" importer prompt now just silently returns no results. More info is in the verbose logs.
- *MBSync Plugin*: Fix application of album-level metadata. Due to a regression a few releases ago, only track-level metadata was being updated.
- On Windows, paths on network shares (UNC paths) no longer cause "invalid filename" errors.
- *ReplayGain Plugin*: Fix crashes when attempting to log errors.
- The *modify* command can now accept query arguments that contain = signs. An argument is considered a query part when a : appears before any =s. Thanks to mook.

1.6.4 1.3.4 (April 5, 2014)

This release brings a hodgepodge of medium-sized conveniences to beets. A new *config* command manages your configuration, we now have *bash completion*, and the *modify* command can delete attributes. There are also some significant performance optimizations to the autotagger's matching logic.

One note for upgraders: if you use the *FetchArt Plugin*, it has a new dependency, the requests module.

New stuff:

- Added a *config* command to manage your configuration. It can show you what you currently have in your config file, point you at where the file should be, or launch your text editor to let you modify the file. Thanks to geigerzaehler.
- Beets now ships with a shell command completion script! See *Shell Completion*. Thanks to geigerzaehler.
- The *modify* command now allows removing flexible attributes. For example, `beet modify artist:beatles oldies!` deletes the `oldies` attribute from matching items. Thanks to brilnius.
- Internally, beets has laid the groundwork for supporting multi-valued fields. Thanks to geigerzaehler.
- The importer interface now shows the URL for MusicBrainz matches. Thanks to johtso.
- *Smart Playlist Plugin*: Playlists can now be generated from multiple queries (combined with "or" logic). Album-level queries are also now possible and automatic playlist regeneration can now be disabled. Thanks to brilnius.
- *Echo Nest Plugin*: Echo Nest similarity now weights the tempo in better proportion to other metrics. Also, options were added to specify custom thresholds and output formats. Thanks to Adam M.

- Added the *after_write* plugin event.
- *LastGenre Plugin*: Separator in genre lists can now be configured. Thanks to brilnius.
- We now only use "primary" aliases for artist names from MusicBrainz. This eliminates some strange naming that could occur when the *languages* config option was set. Thanks to Filipe Fortes.
- The performance of the autotagger's matching mechanism is vastly improved. This should be noticeable when matching against very large releases such as box sets.
- The *import* command can now accept individual files as arguments even in non-singleton mode. Files are imported as one-track albums.

Fixes:

- Error messages involving paths no longer escape non-ASCII characters (for legibility).
- Fixed a regression that made it impossible to use the *modify* command to add new flexible fields. Thanks to brilnius.
- *Echo Nest Plugin*: Avoid crashing when the audio analysis fails. Thanks to Pedro Silva.
- *Duplicates Plugin*: Fix checksumming command execution for files with quotation marks in their names. Thanks again to Pedro Silva.
- Fix a crash when importing with both of the *group_albums* and *incremental* options enabled. Thanks to geigerzaehler.
- Give a sensible error message when `BEETSDIR` points to a file. Thanks again to geigerzaehler.
- Fix a crash when reading WMA files whose boolean-valued fields contain strings. Thanks to johtso.
- *FetchArt Plugin*: The plugin now sends "beets" as the User-Agent when making scraping requests. This helps resolve some blocked requests. The plugin now also depends on the requests Python library.
- The *write* command now only shows the changes to fields that will actually be written to a file.
- *Duplicates Plugin*: Spurious reports are now avoided for tracks with missing values (e.g., no MBIDs). Thanks to Pedro Silva.
- The default *replace* sanitation options now remove leading whitespace by default. Thanks to brilnius.
- *ImportFeeds Plugin*: Fix crash when importing albums containing `/` with the `m3u_multi` format.
- Avoid crashing on Mutagen bugs while writing files' tags.
- *Convert Plugin*: Display a useful error message when the FFmpeg executable can't be found.

1.6.5 1.3.3 (February 26, 2014)

Version 1.3.3 brings a bunch changes to how item and album fields work internally. Along with laying the groundwork for some great things in the future, this brings a number of improvements to how you interact with beets. Here's what's new with fields in particular:

- Plugin-provided fields can now be used in queries. For example, if you use the *Inline Plugin* to define a field called `era`, you can now filter your library based on that field by typing something like `beet list era:goldenage`.
- Album-level flexible attributes and plugin-provided attributes can now be used in path formats (and other item-level templates).
- *Date-based queries* are now possible. Try getting every track you added in February 2014 with `beet ls added:2014-02` or in the whole decade with `added:2010..`. Thanks to Stig Inge Lea Bjørnsen.

- The *modify* command is now better at parsing and formatting fields. You can assign to boolean fields like `comp`, for example, using either the words "true" or "false" or the numerals 1 and 0. Any boolean-esque value is normalized to a real boolean. The *update* and *write* commands also got smarter at formatting and colorizing changes.

For developers, the short version of the story is that Item and Album objects provide *uniform access* across fixed, flexible, and computed attributes. You can write `item.foo` to access the `foo` field without worrying about where the data comes from.

Unrelated new stuff:

- The importer has a new interactive option (*G* for "Group albums"), command-line flag (`--group-albums`), and config option (*group_albums*) that lets you split apart albums that are mixed together in a single directory. Thanks to geigerzaehler.
- A new `--config` command-line option lets you specify an additional configuration file. This option *combines* config settings with your default config file. (As part of this change, the `BEETSDIR` environment variable no longer combines—it *replaces* your default config file.) Thanks again to geigerzaehler.
- *IHate Plugin*: The plugin's configuration interface was overhauled. Its configuration is now much simpler—it uses beets queries instead of an ad-hoc per-field configuration. This is *backwards-incompatible*—if you use this plugin, you will need to update your configuration. Thanks to BrainDamage.

Other little fixes:

- *Echo Nest Plugin*: Tempo (BPM) is now always stored as an integer. Thanks to Heinz Wiesinger.
- Fix Python 2.6 compatibility in some logging statements in *Chromaprint/Acoustid Plugin* and *LastGenre Plugin*.
- Prevent some crashes when things go really wrong when writing file metadata at the end of the import process.
- New plugin events: `item_removed` (thanks to Romuald Conty) and `item_copied` (thanks to Stig Inge Lea Bjørnsen).
- The `pluginpath` config option can now point to the directory containing plugin code. (Previously, it awkwardly needed to point at a directory containing a `beetsplug` directory, which would then contain your code. This is preserved as an option for backwards compatibility.) This change should also work around a long-standing issue when using `pluginpath` when beets is installed using pip. Many thanks to geigerzaehler.
- *Web Plugin*: The `/item/` and `/album/` API endpoints now produce full details about albums and items, not just lists of IDs. Thanks to geigerzaehler.
- Fix a potential crash when using image resizing with the *FetchArt Plugin* or *EmbedArt Plugin* without ImageMagick installed.
- Also, when invoking `convert` for image resizing fails, we now log an error instead of crashing.
- *FetchArt Plugin*: The `beet fetchart` command can now associate local images with albums (unless `--force` is provided). Thanks to brilnius.
- *FetchArt Plugin*: Command output is now colorized. Thanks again to brilnius.
- The *modify* command avoids writing files and committing to the database when nothing has changed. Thanks once more to brilnius.
- The importer now uses the album artist field when guessing existing metadata for albums (rather than just the track artist field). Thanks to geigerzaehler.
- *FromFilename Plugin*: Fix a crash when a filename contained only a track number (e.g., `02.mp3`).
- *Convert Plugin*: Transcoding should now work on Windows.
- *Duplicates Plugin*: The `move` and `copy` destination arguments are now treated as directories. Thanks to Pedro Silva.

- The *modify* command now skips confirmation and prints a message if no changes are necessary. Thanks to brilnius.
- *FetchArt Plugin*: When using the `remote_priority` config option, local image files are no longer completely ignored.
- *Echo Nest Plugin*: Fix an issue causing the plugin to appear twice in the output of the `beet version` command.
- *LastGenre Plugin*: Fix an occasional crash when no tag weight was returned by Last.fm.
- *MPDStats Plugin*: Restore the `last_played` field. Thanks to Johann Klähn.
- The *modify* command's output now clearly shows when a file has been deleted.
- Album art in files with Vorbis Comments is now marked with the "front cover" type. Thanks to Jason Lefley.

1.6.6 1.3.2 (December 22, 2013)

This update brings new plugins for fetching acoustic metrics and listening statistics, many more options for the duplicate detection plugin, and flexible options for fetching multiple genres.

The "core" of beets gained a new built-in command: *beet write* updates the metadata tags for files, bringing them back into sync with your database. Thanks to Heinz Wiesinger.

We added some plugins and overhauled some existing ones:

- The new *Echo Nest Plugin* plugin can fetch a wide range of acoustic attributes from The Echo Nest, including the "speechiness" and "liveness" of each track. The new plugin supersedes an older version (`echonest_tempo`) that only fetched the BPM field. Thanks to Pedro Silva and Peter Schnebel.
- The *Duplicates Plugin* got a number of new features, thanks to Pedro Silva:
  - The `keys` option lets you specify the fields used detect duplicates.
  - You can now use checksumming (via an external command) to find duplicates instead of metadata via the `checksum` option.
  - The plugin can perform actions on the duplicates it find. The new `copy`, `move`, `delete`, `delete_file`, and `tag` options perform those actions.
- The new *MPDStats Plugin* collects statistics about your listening habits from MPD. Thanks to Peter Schnebel and Johann Klähn.
- *LastGenre Plugin*: The new `multiple` option has been replaced with the `count` option, which lets you limit the number of genres added to your music. (No more thousand-character genre fields!) Also, the `min_weight` field filters out nonsense tags to make your genres more relevant. Thanks to Peter Schnebel and rashley60.
- *Lyrics Plugin*: A new `--force` option optionally re-downloads lyrics even when files already have them. Thanks to Bitdemon.

As usual, there are also innumerable little fixes and improvements:

- When writing ID3 tags for ReplayGain normalization, tags are written with both upper-case and lower-case TXXX frame descriptions. Previous versions of beets used only the upper-case style, which seems to be more standard, but some players (namely, Quod Libet and foobar2000) seem to only use lower-case names.
- *Missing Plugin*: Avoid a possible error when an album's `tracktotal` field is missing.
- *FtInTitle Plugin*: Fix an error when the sort artist is missing.
- *EchoNest Tempo Plugin*: The plugin should now match songs more reliably (i.e., fewer "no tempo found" messages). Thanks to Peter Schnebel.

- *Convert Plugin*: Fix an "Item has no library" error when using the `auto` config option.
- *Convert Plugin*: Fix an issue where files of the wrong format would have their transcoding skipped (and files with the right format would be needlessly transcoded). Thanks to Jakob Schnitzer.
- Fix an issue that caused the *id3v23* option to work only occasionally.
- Also fix using *id3v23* in conjunction with the `scrub` and `embedart` plugins. Thanks to Chris Cogburn.
- *IHate Plugin*: Fix an error when importing singletons. Thanks to Mathijs de Bruin.
- The *clutter* option can now be a whitespace-separated list in addition to a YAML list.
- Values for the *replace* option can now be empty (i.e., null is equivalent to the empty string).
- *LastGenre Plugin*: Fix a conflict between canonicalization and multiple genres.
- When a match has a year but not a month or day, the autotagger now "zeros out" the month and day fields after applying the year.
- For plugin developers: added an `optparse` callback utility function for performing actions based on arguments. Thanks to Pedro Silva.
- *Scrub Plugin*: Fix scrubbing of MPEG-4 files. Thanks to Yevgeny Bezman.

1.6.7 1.3.1 (October 12, 2013)

This release boasts a host of new little features, many of them contributed by beets' amazing and prolific community. It adds support for Opus files, transcoding to any format, and two new plugins: one that guesses metadata for "blank" files based on their filenames and one that moves featured artists into the title field.

Here's the new stuff:

- Add Opus audio support. Thanks to Rowan Lewis.
- *Convert Plugin*: You can now transcode files to any audio format, rather than just MP3. Thanks again to Rowan Lewis.
- The new *FromFilename Plugin* guesses tags from the filenames during import when metadata tags themselves are missing. Thanks to Jan-Erik Dahlin.
- The *FtInTitle Plugin*, by @Verrus, is now distributed with beets. It helps you rewrite tags to move "featured" artists from the artist field to the title field.
- The MusicBrainz data source now uses track artists over recording artists. This leads to better metadata when tagging classical music. Thanks to Henrique Ferreiro.
- *LastGenre Plugin*: You can now get multiple genres per album or track using the `multiple` config option. Thanks to rashley60 on GitHub.
- A new *id3v23* config option makes beets write MP3 files' tags using the older ID3v2.3 metadata standard. Use this if you want your tags to be visible to Windows and some older players.

And some fixes:

- *FetchArt Plugin*: Better error message when the image file has an unrecognized type.
- *MusicBrainz Collection Plugin*: Detect, log, and skip invalid MusicBrainz IDs (instead of failing with an API error).
- *Info Plugin*: Fail gracefully when used erroneously with a directory.
- *EchoNest Tempo Plugin*: Fix an issue where the plugin could use the tempo from the wrong song when the API did not contain the requested song.

- Fix a crash when a file's metadata included a very large number (one wider than 64 bits). These huge numbers are now replaced with zeroes in the database.
- When a track on a MusicBrainz release has a different length from the underlying recording's length, the track length is now used instead.
- With *per_disc_numbering* enabled, the `tracktotal` field is now set correctly (i.e., to the number of tracks on the disc).
- *Scrub Plugin*: The `scrub` command now restores album art in addition to other (database-backed) tags.
- *MPDUpdate Plugin*: Domain sockets can now begin with a tilde (which is correctly expanded to `$HOME`) as well as a slash. Thanks to Johann Klähn.
- *LastGenre Plugin*: Fix a regression that could cause new genres found during import not to be persisted.
- Fixed a crash when imported album art was also marked as "clutter" where the art would be deleted before it could be moved into place. This led to a "image.jpg not found during copy" error. Now clutter is removed (and directories pruned) much later in the process, after the `import_task_files` hook.
- *Missing Plugin*: Fix an error when printing missing track names. Thanks to Pedro Silva.
- Fix an occasional KeyError in the *update* command introduced in 1.3.0.
- *Scrub Plugin*: Avoid preserving certain non-standard ID3 tags such as NCON.

1.6.8 1.3.0 (September 11, 2013)

Albums and items now have **flexible attributes**. This means that, when you want to store information about your music in the beets database, you're no longer constrained to the set of fields it supports out of the box (title, artist, track, etc.). Instead, you can use any field name you can think of and treat it just like the built-in fields.

For example, you can use the *modify* command to set a new field on a track:

```
$ beet modify mood=sexy artist:miguel
```

and then query your music based on that field:

```
$ beet ls mood:sunny
```

or use templates to see the value of the field:

```
$ beet ls -f '$title: $mood'
```

While this feature is nifty when used directly with the usual command-line suspects, it's especially useful for plugin authors and for future beets features. Stay tuned for great things built on this flexible attribute infrastructure.

One side effect of this change: queries that include unknown fields will now match *nothing* instead of *everything*. So if you type `beet ls fieldThatDoesNotExist:foo`, beets will now return no results, whereas previous versions would spit out a warning and then list your entire library.

There's more detail than you could ever need on the beets blog.

1.6.9 1.2.2 (August 27, 2013)

This is a bugfix release. We're in the midst of preparing for a large change in beets 1.3, so 1.2.2 resolves some issues that came up over the last few weeks. Stay tuned!

The improvements in this release are:

- A new plugin event, `item_moved`, is sent when files are moved on disk. Thanks to dsedivec.

- *Lyrics Plugin*: More improvements to the Google backend by Fabrice Laporte.
- *BPD Plugin*: Fix for a crash when searching, thanks to Simon Chopin.
- Regular expression queries (and other query types) over paths now work. (Previously, special query types were ignored for the `path` field.)
- *FetchArt Plugin*: Look for images in the Cover Art Archive for the release group in addition to the specific release. Thanks to Filipe Fortes.
- Fix a race in the importer that could cause files to be deleted before they were imported. This happened when importing one album, importing a duplicate album, and then asking for the first album to be replaced with the second. The situation could only arise when importing music from the library directory and when the two albums are imported close in time.

1.6.10 1.2.1 (June 22, 2013)

This release introduces a major internal change in the way that similarity scores are handled. It means that the importer interface can now show you exactly why a match is assigned its score and that the autotagger gained a few new options that let you customize how matches are prioritized and recommended.

The refactoring work is due to the continued efforts of Tai Lee. The changes you'll notice while using the autotagger are:

- The top 3 distance penalties are now displayed on the release listing, and all album and track penalties are now displayed on the track changes list. This should make it clear exactly which metadata is contributing to a low similarity score.
- When displaying differences, the colorization has been made more consistent and helpful: red for an actual difference, yellow to indicate that a distance penalty is being applied, and light gray for no penalty (e.g., case changes) or disambiguation data.

There are also three new (or overhauled) configuration options that let you customize the way that matches are selected:

- The *ignored* setting lets you instruct the importer not to show you matches that have a certain penalty applied.
- The *preferred* collection of settings specifies a sorted list of preferred countries and media types, or prioritizes releases closest to the original year for an album.
- The *max_rec* settings can now be used for any distance penalty component. The recommendation will be downgraded if a non-zero penalty is being applied to the specified field.

And some little enhancements and bug fixes:

- Multi-disc directory names can now contain "disk" (in addition to "disc"). Thanks to John Hawthorn.
- *Web Plugin*: Item and album counts are now exposed through the API for use with the Tomahawk resolver. Thanks to Uwe L. Korn.
- Python 2.6 compatibility for *Beatport Plugin*, *Missing Plugin*, and *Duplicates Plugin*. Thanks to Wesley Bitter and Pedro Silva.
- Don't move the config file during a null migration. Thanks to Theofilos Intzoglou.
- Fix an occasional crash in the *Beatport Plugin* when a length field was missing from the API response. Thanks to Timothy Appnel.
- *Scrub Plugin*: Handle and log I/O errors.
- *Lyrics Plugin*: The Google backend should now turn up more results. Thanks to Fabrice Laporte.
- *Random Plugin*: Fix compatibility with Python 2.6. Thanks to Matthias Drochner.

1.6.11 1.2.0 (June 5, 2013)

There's a *lot* of new stuff in this release: new data sources for the autotagger, new plugins to look for problems in your library, tracking the date that you acquired new music, an awesome new syntax for doing queries over numeric fields, support for ALAC files, and major enhancements to the importer's UI and distance calculations. A special thanks goes out to all the contributors who helped make this release awesome.

For the first time, beets can now tag your music using additional **data sources** to augment the matches from MusicBrainz. When you enable either of these plugins, the importer will start showing you new kinds of matches:

- New *Discogs Plugin*: Get matches from the Discogs database. Thanks to Artem Ponomarenko and Tai Lee.
- New *Beatport Plugin*: Get matches from the Beatport database. Thanks to Johannes Baiter.

We also have two other new plugins that can scan your library to check for common problems, both by Pedro Silva:

- New *Duplicates Plugin*: Find tracks or albums in your library that are **duplicated**.
- New *Missing Plugin*: Find albums in your library that are **missing tracks**.

There are also three more big features added to beets core:

- Your library now keeps track of **when music was added** to it. The new `added` field is a timestamp reflecting when each item and album was imported and the new `%time{}` template function lets you format this timestamp for humans. Thanks to Lucas Duailibe.
- When using queries to match on quantitative fields, you can now use **numeric ranges**. For example, you can get a list of albums from the '90s by typing `beet ls year:1990..1999` or find high-bitrate music with `bitrate:128000..`. See *Numeric Range Queries*. Thanks to Michael Schuerig.
- **ALAC files** are now marked as ALAC instead of being conflated with AAC audio. Thanks to Simon Luijk.

In addition, the importer saw various UI enhancements, thanks to Tai Lee:

- More consistent format and colorization of album and track metadata.
- Display data source URL for matches from the new data source plugins. This should make it easier to migrate data from Discogs or Beatport into MusicBrainz.
- Display album disambiguation and disc titles in the track listing, when available.
- Track changes are highlighted in yellow when they indicate a change in format to or from the style of *per_disc_numbering*. (As before, no penalty is applied because the track number is still "correct", just in a different format.)
- Sort missing and unmatched tracks by index and title and group them together for better readability.
- Indicate MusicBrainz ID mismatches.

The calculation of the similarity score for autotagger matches was also improved, again thanks to Tai Lee. These changes, in general, help deal with the new metadata sources and help disambiguate between similar releases in the same MusicBrainz release group:

- Strongly prefer releases with a matching MusicBrainz album ID. This helps beets re-identify the same release when re-importing existing files.
- Prefer releases that are closest to the tagged `year`. Tolerate files tagged with release or original year.
- The new `preferred_media` config option lets you prefer a certain media type when the `media` field is unset on an album.
- Apply minor penalties across a range of fields to differentiate between nearly identical releases: `disctotal`, `label`, `catalognum`, `country` and `albumdisambig`.

As usual, there were also lots of other great littler enhancements:

- *Random Plugin*: A new `-e` option gives an equal chance to each artist in your collection to avoid biasing random samples to prolific artists. Thanks to Georges Dubus.
- The *modify* now correctly converts types when modifying non-string fields. You can now safely modify the "comp" flag and the "year" field, for example. Thanks to Lucas Duailibe.
- *Convert Plugin*: You can now configure the path formats for converted files separately from your main library. Thanks again to Lucas Duailibe.
- The importer output now shows the number of audio files in each album. Thanks to jayme on GitHub.
- Plugins can now provide fields for both Album and Item templates, thanks to Pedro Silva. Accordingly, the *Inline Plugin* can also now define album fields. For consistency, the `pathfields` configuration section has been renamed `item_fields` (although the old name will still work for compatibility).
- Plugins can also provide metadata matches for ID searches. For example, the new Discogs plugin lets you search for an album by its Discogs ID from the same prompt that previously just accepted MusicBrainz IDs. Thanks to Johannes Baiter.
- The *fields* command shows template fields provided by plugins. Thanks again to Pedro Silva.
- *MPDUpdate Plugin*: You can now communicate with MPD over a Unix domain socket. Thanks to John Hawthorn.

And a batch of fixes:

- Album art filenames now respect the *replace* configuration.
- Friendly error messages are now printed when trying to read or write files that go missing.
- The *modify* command can now change albums' album art paths (i.e., `beet modify artpath=...` works). Thanks to Lucas Duailibe.
- *Zero Plugin*: Fix a crash when nulling out a field that contains None.
- Templates can now refer to non-tag item fields (e.g., `$id` and `$album_id`).
- *Lyrics Plugin*: Lyrics searches should now turn up more results due to some fixes in dealing with special characters.

1.6.12 1.1.0 (April 29, 2013)

This final release of 1.1 brings a little polish to the betas that introduced the new configuration system. The album art and lyrics plugins also got a little love.

If you're upgrading from 1.0.0 or earlier, this release (like the 1.1 betas) will automatically migrate your configuration to the new system. See *Upgrading from 1.0*.

- *EmbedArt Plugin*: The `embedart` command now embeds each album's associated art by default. The `--file` option invokes the old behavior, in which a specific image file is used.
- *Lyrics Plugin*: A new (optional) Google Custom Search backend was added for finding lyrics on a wide array of sites. Thanks to Fabrice Laporte.
- When automatically detecting the filesystem's maximum filename length, never guess more than 200 characters. This prevents errors on systems where the maximum length was misreported. You can, of course, override this default with the *max_filename_length* option.
- *FetchArt Plugin*: Two new configuration options were added: `cover_names`, the list of keywords used to identify preferred images, and `cautious`, which lets you avoid falling back to images that don't contain those keywords. Thanks to Fabrice Laporte.

- Avoid some error cases in the `update` command and the `embedart` and `mbsync` plugins. Invalid or missing files now cause error logs instead of crashing beets. Thanks to Lucas Duailibe.
- *Lyrics Plugin*: Searches now strip "featuring" artists when searching for lyrics, which should increase the hit rate for these tracks. Thanks to Fabrice Laporte.
- When listing the items in an album, the items are now always in track-number order. This should lead to more predictable listings from the *ImportFeeds Plugin*.
- *Smart Playlist Plugin*: Queries are now split using shell-like syntax instead of just whitespace, so you can now construct terms that contain spaces.
- *LastGenre Plugin*: The `force` config option now defaults to true and controls the behavior of the import hook. (Previously, new genres were always forced during import.)
- *Web Plugin*: Fix an error when specifying the hostname on the command line.
- *Web Plugin*: The underlying API was expanded slightly to support Tomahawk collections. And file transfers now have a "Content-Length" header. Thanks to Uwe L. Korn.
- *LastGenre Plugin*: Fix an error when using genre canonicalization.

1.6.13 1.1b3 (March 16, 2013)

This third beta of beets 1.1 brings a hodgepodge of little new features (and internal overhauls that will make improvements easier in the future). There are new options for getting metadata in a particular language and seeing more detail during the import process. There's also a new plugin for synchronizing your metadata with MusicBrainz. Under the hood, plugins can now extend the query syntax.

New configuration options:

- *languages* controls the preferred languages when selecting an alias from MusicBrainz. This feature requires python-musicbrainz-ngs 0.3 or later. Thanks to Sam Doshi.
- *detail* enables a mode where all tracks are listed in the importer UI, as opposed to only changed tracks.
- The `--flat` option to the `beet import` command treats an entire directory tree of music files as a single album. This can help in situations where a multi-disc album is split across multiple directories.
- *ImportFeeds Plugin*: An option was added to use absolute, rather than relative, paths. Thanks to Lucas Duailibe.

Other stuff:

- A new *MBSync Plugin* provides a command that looks up each item and track in MusicBrainz and updates your library to reflect it. This can help you easily correct errors that have been fixed in the MB database. Thanks to Jakob Schnitzer.
- *Fuzzy Search Plugin*: The `fuzzy` command was removed and replaced with a new query type. To perform fuzzy searches, use the ~ prefix with *list* or other commands. Thanks to Philippe Mongeau.
- As part of the above, plugins can now extend the query syntax and new kinds of matching capabilities to beets. See *Extend the Query Syntax*. Thanks again to Philippe Mongeau.
- *Convert Plugin*: A new `--keep-new` option lets you store transcoded files in your library while backing up the originals (instead of vice-versa). Thanks to Lucas Duailibe.
- *Convert Plugin*: Also, a new `auto` config option will transcode audio files automatically during import. Thanks again to Lucas Duailibe.
- *Chromaprint/Acoustid Plugin*: A new `fingerprint` command lets you generate and store fingerprints for items that don't yet have them. One more round of applause for Lucas Duailibe.

- *EchoNest Tempo Plugin*: API errors now issue a warning instead of exiting with an exception. We also avoid an error when track metadata contains newlines.
- When the importer encounters an error (insufficient permissions, for example) when walking a directory tree, it now logs an error instead of crashing.
- In path formats, null database values now expand to the empty string instead of the string "None".
- Add "System Volume Information" (an internal directory found on some Windows filesystems) to the default ignore list.
- Fix a crash when ReplayGain values were set to null.
- Fix a crash when iTunes Sound Check tags contained invalid data.
- Fix an error when the configuration file (`config.yaml`) is completely empty.
- Fix an error introduced in 1.1b1 when importing using timid mode. Thanks to Sam Doshi.
- *Convert Plugin*: Fix a bug when creating files with Unicode pathnames.
- Fix a spurious warning from the Unidecode module when matching albums that are missing all metadata.
- Fix Unicode errors when a directory or file doesn't exist when invoking the import command. Thanks to Lucas Duailibe.
- *MusicBrainz Collection Plugin*: Show friendly, human-readable errors when MusicBrainz exceptions occur.
- *EchoNest Tempo Plugin*: Catch socket errors that are not handled by the Echo Nest library.
- *Chromaprint/Acoustid Plugin*: Catch Acoustid Web service errors when submitting fingerprints.

1.6.14 1.1b2 (February 16, 2013)

The second beta of beets 1.1 uses the fancy new configuration infrastructure to add many, many new config options. The import process is more flexible; filenames can be customized in more detail; and more. This release also supports Windows Media (ASF) files and iTunes Sound Check volume normalization.

This version introduces one **change to the default behavior** that you should be aware of. Previously, when importing new albums matched in MusicBrainz, the date fields (`year`, `month`, and `day`) would be set to the release date of the *original* version of the album, as opposed to the specific date of the release selected. Now, these fields reflect the specific release and `original_year`, etc., reflect the earlier release date. If you want the old behavior, just set *original_date* to true in your config file.

New configuration options:

- *default_action* lets you determine the default (just-hit-return) option is when considering a candidate.
- *none_rec_action* lets you skip the prompt, and automatically choose an action, when there is no good candidate. Thanks to Tai Lee.
- *max_rec* lets you define a maximum recommendation for albums with missing/extra tracks or differing track lengths/numbers. Thanks again to Tai Lee.
- *original_date* determines whether, when importing new albums, the `year`, `month`, and `day` fields should reflect the specific (e.g., reissue) release date or the original release date. Note that the original release date is always available as `original_year`, etc.
- *clutter* controls which files should be ignored when cleaning up empty directories. Thanks to Steinþór Pálsson.
- *LastGenre Plugin*: A new configuration option lets you choose to retrieve artist-level tags as genres instead of album- or track-level tags. Thanks to Peter Fern and Peter Schnebel.

- *max_filename_length* controls truncation of long filenames. Also, beets now tries to determine the filesystem's maximum length automatically if you leave this option unset.
- *FetchArt Plugin*: The `remote_priority` option searches remote (Web) art sources even when local art is present.
- You can now customize the character substituted for path separators (e.g., /) in filenames via `path_sep_replace`. The default is an underscore. Use this setting with caution.

Other new stuff:

- Support for Windows Media/ASF audio files. Thanks to Dave Hayes.
- New *Smart Playlist Plugin*: generate and maintain m3u playlist files based on beets queries. Thanks to Dang Mai Hai.
- ReplayGain tags on MPEG-4/AAC files are now supported. And, even more astonishingly, ReplayGain values in MP3 and AAC files are now compatible with iTunes Sound Check. Thanks to Dave Hayes.
- Track titles in the importer UI's difference display are now either aligned vertically or broken across two lines for readability. Thanks to Tai Lee.
- Albums and items have new fields reflecting the *original* release date (`original_year`, `original_month`, and `original_day`). Previously, when tagging from MusicBrainz, *only* the original date was stored; now, the old fields refer to the *specific* release date (e.g., when the album was reissued).
- Some changes to the way candidates are recommended for selection, thanks to Tai Lee:
  - According to the new *max_rec* configuration option, partial album matches are downgraded to a "low" recommendation by default.
  - When a match isn't great but is either better than all the others or the only match, it is given a "low" (rather than "medium") recommendation.
  - There is no prompt default (i.e., input is required) when matches are bad: "low" or "none" recommendations or when choosing a candidate other than the first.
- The importer's heuristic for coalescing the directories in a multi-disc album has been improved. It can now detect when two directories alongside each other share a similar prefix but a different number (e.g., "Album Disc 1" and "Album Disc 2") even when they are not alone in a common parent directory. Thanks once again to Tai Lee.
- Album listings in the importer UI now show the release medium (CD, Vinyl, 3xCD, etc.) as well as the disambiguation string. Thanks to Peter Schnebel.
- *LastGenre Plugin*: The plugin can now get different genres for individual tracks on an album. Thanks to Peter Schnebel.
- When getting data from MusicBrainz, the album disambiguation string (`albumdisambig`) now reflects both the release and the release group.
- *MPDUpdate Plugin*: Sends an update message whenever *anything* in the database changes—not just when importing. Thanks to Dang Mai Hai.
- When the importer UI shows a difference in track numbers or durations, they are now colorized based on the *suffixes* that differ. For example, when showing the difference between 2:01 and 2:09, only the last digit will be highlighted.
- The importer UI no longer shows a change when the track length difference is less than 10 seconds. (This threshold was previously 2 seconds.)
- Two new plugin events were added: *database_change* and *cli_exit*. Thanks again to Dang Mai Hai.
- Plugins are now loaded in the order they appear in the config file. Thanks to Dang Mai Hai.

- *BPD Plugin*: Browse by album artist and album artist sort name. Thanks to Steinþór Pálsson.
- *EchoNest Tempo Plugin*: Don't attempt a lookup when the artist or track title is missing.
- Fix an error when migrating the `.beetsstate` file on Windows.
- A nicer error message is now given when the configuration file contains tabs. (YAML doesn't like tabs.)
- Fix the `-l` (log path) command-line option for the `import` command.

1.6.15 1.1b1 (January 29, 2013)

This release entirely revamps beets' configuration system. The configuration file is now a YAML document and is located, along with other support files, in a common directory (e.g., `~/.config/beets` on Unix-like systems). If you're upgrading from an earlier version, please see *Upgrading from 1.0*.

- Renamed plugins: The `rdm` plugin has been renamed to `random` and `fuzzy_search` has been renamed to `fuzzy`.
- Renamed config options: Many plugins have a flag dictating whether their action runs at import time. This option had many names (`autofetch`, `autoembed`, etc.) but is now consistently called `auto`.
- Reorganized import config options: The various `import_*` options are now organized under an `import:` heading and their prefixes have been removed.
- New default file locations: The default filename of the library database is now `library.db` in the same directory as the config file, as opposed to `~/.beetsmusic.blb` previously. Similarly, the runtime state file is now called `state.pickle` in the same directory instead of `~/.beetsstate`.

It also adds some new features:

- *Inline Plugin*: Inline definitions can now contain statements or blocks in addition to just expressions. Thanks to Florent Thoumie.
- Add a configuration option, *terminal_encoding*, controlling the text encoding used to print messages to standard output.
- The MusicBrainz hostname (and rate limiting) are now configurable. See *MusicBrainz Options*.
- You can now configure the similarity thresholds used to determine when the autotagger automatically accepts a metadata match. See *Autotagger Matching Options*.
- *ImportFeeds Plugin*: Added a new configuration option that controls the base for relative paths used in m3u files. Thanks to Philippe Mongeau.

1.6.16 1.0.0 (January 29, 2013)

After fifteen betas and two release candidates, beets has finally hit one-point-oh. Congratulations to everybody involved. This version of beets will remain stable and receive only bug fixes from here on out. New development is ongoing in the betas of version 1.1.

- *Scrub Plugin*: Fix an incompatibility with Python 2.6.
- *Lyrics Plugin*: Fix an issue that failed to find lyrics when metadata contained "real" apostrophes.
- *ReplayGain Plugin*: On Windows, emit a warning instead of crashing when analyzing non-ASCII filenames.
- Silence a spurious warning from version 0.04.12 of the Unidecode module.

1.6.17 1.0rc2 (December 31, 2012)

This second release candidate follows quickly after rc1 and fixes a few small bugs found since that release. There were a couple of regressions and some bugs in a newly added plugin.

- *EchoNest Tempo Plugin*: If the Echo Nest API limit is exceeded or a communication error occurs, the plugin now waits and tries again instead of crashing. Thanks to Zach Denton.
- *FetchArt Plugin*: Fix a regression that caused crashes when art was not available from some sources.
- Fix a regression on Windows that caused all relative paths to be "not found".

1.6.18 1.0rc1 (December 17, 2012)

The first release candidate for beets 1.0 includes a deluge of new features contributed by beets users. The vast majority of the credit for this release goes to the growing and vibrant beets community. A million thanks to everybody who contributed to this release.

There are new plugins for transcoding music, fuzzy searches, tempo collection, and fiddling with metadata. The ReplayGain plugin has been rebuilt from scratch. Album art images can now be resized automatically. Many other smaller refinements make things "just work" as smoothly as possible.

With this release candidate, beets 1.0 is feature-complete. We'll be fixing bugs on the road to 1.0 but no new features will be added. Concurrently, work begins today on features for version 1.1.

- New plugin: *Convert Plugin* **transcodes** music and embeds album art while copying to a separate directory. Thanks to Jakob Schnitzer and Andrew G. Dunn.
- New plugin: *Fuzzy Search Plugin* lets you find albums and tracks using **fuzzy string matching** so you don't have to type (or even remember) their exact names. Thanks to Philippe Mongeau.
- New plugin: *EchoNest Tempo Plugin* fetches **tempo** (BPM) information from The Echo Nest. Thanks to David Brenner.
- New plugin: *The Plugin* adds a template function that helps format text for nicely-sorted directory listings. Thanks to Blemjhoo Tezoulbr.
- New plugin: *Zero Plugin* **filters out undesirable fields** before they are written to your tags. Thanks again to Blemjhoo Tezoulbr.
- New plugin: *IHate Plugin* automatically skips (or warns you about) importing albums that match certain criteria. Thanks once again to Blemjhoo Tezoulbr.
- *ReplayGain Plugin*: This plugin has been completely overhauled to use the mp3gain or aacgain command-line tools instead of the failure-prone Gstreamer ReplayGain implementation. Thanks to Fabrice Laporte.
- *FetchArt Plugin* and *EmbedArt Plugin*: Both plugins can now **resize album art** to avoid excessively large images. Use the `maxwidth` config option with either plugin. Thanks to Fabrice Laporte.
- *Scrub Plugin*: Scrubbing now removes *all* types of tags from a file rather than just one. For example, if your FLAC file has both ordinary FLAC tags and ID3 tags, the ID3 tags are now also removed.
- *stats* command: New `--exact` switch to make the file size calculation more accurate (thanks to Jakob Schnitzer).
- *list* command: Templates given with `-f` can now show items' and albums' paths (using `$path`).
- The output of the *update*, *remove*, and *modify* commands now respects the *list_format_album* and *list_format_item* config options. Thanks to Mike Kazantsev.
- The *art_filename* option can now be a template rather than a simple string. Thanks to Jarrod Beardwood.

- Fix album queries for `artpath` and other non-item fields.
- Null values in the database can now be matched with the empty-string regular expression, `^$`.
- Queries now correctly match non-string values in path format predicates.
- When autotagging a various-artists album, the album artist field is now used instead of the majority track artist.
- *LastGenre Plugin*: Use the albums' existing genre tags if they pass the whitelist (thanks to Fabrice Laporte).
- *LastGenre Plugin*: Add a `lastgenre` command for fetching genres post facto (thanks to Jakob Schnitzer).
- *FetchArt Plugin*: Local image filenames are now used in alphabetical order.
- *FetchArt Plugin*: Fix a bug where cover art filenames could lack a `.jpg` extension.
- *Lyrics Plugin*: Fix an exception with non-ASCII lyrics.
- *Web Plugin*: The API now reports file sizes (for use with the Tomahawk resolver).
- *Web Plugin*: Files now download with a reasonable filename rather than just being called "file" (thanks to Zach Denton).
- *ImportFeeds Plugin*: Fix error in symlink mode with non-ASCII filenames.
- *MusicBrainz Collection Plugin*: Fix an error when submitting a large number of releases (we now submit only 200 releases at a time instead of 350). Thanks to Jonathan Towne.
- *EmbedArt Plugin*: Made the method for embedding art into FLAC files standard-compliant. Thanks to Daniele Sluijters.
- Add the track mapping dictionary to the `album_distance` plugin function.
- When an exception is raised while reading a file, the path of the file in question is now logged (thanks to Mike Kazantsev).
- Truncate long filenames based on their *bytes* rather than their Unicode *characters*, fixing situations where encoded names could be too long.
- Filename truncation now incorporates the length of the extension.
- Fix an assertion failure when the MusicBrainz main database and search server disagree.
- Fix a bug that caused the *LastGenre Plugin* and other plugins not to modify files' tags even when they successfully change the database.
- Fix a VFS bug leading to a crash in the *BPD Plugin* when files had non-ASCII extensions.
- Fix for changing date fields (like "year") with the *modify* command.
- Fix a crash when input is read from a pipe without a specified encoding.
- Fix some problem with identifying files on Windows with Unicode directory names in their path.
- Fix a crash when Unicode queries were used with `import -L` re-imports.
- Fix an error when fingerprinting files with Unicode filenames on Windows.
- Warn instead of crashing when importing a specific file in singleton mode.
- Add human-readable error messages when writing files' tags fails or when a directory can't be created.
- Changed plugin loading so that modules can be imported without unintentionally loading the plugins they contain.

1.6.19 1.0b15 (July 26, 2012)

The fifteenth (!) beta of beets is compendium of small fixes and features, most of which represent long-standing requests. The improvements include matching albums with extra tracks, per-disc track numbering in multi-disc albums, an overhaul of the album art downloader, and robustness enhancements that should keep beets running even when things go wrong. All these smaller changes should help us focus on some larger changes coming before 1.0.

Please note that this release contains one backwards-incompatible change: album art fetching, which was previously baked into the import workflow, is now encapsulated in a plugin (the *FetchArt Plugin*). If you want to continue fetching cover art for your music, enable this plugin after upgrading to beets 1.0b15.

- The autotagger can now find matches for albums when you have **extra tracks** on your filesystem that aren't present in the MusicBrainz catalog. Previously, if you tried to match album with 15 audio files but the MusicBrainz entry had only 14 tracks, beets would ignore this match. Now, beets will show you matches even when they are "too short" and indicate which tracks from your disk are unmatched.
- Tracks on multi-disc albums can now be **numbered per-disc** instead of per-album via the *per_disc_numbering* config option.
- The default output format for the `beet list` command is now configurable via the *list_format_item* and *list_format_album* config options. Thanks to Fabrice Laporte.
- Album **cover art fetching** is now encapsulated in the *FetchArt Plugin*. Be sure to enable this plugin if you're using this functionality. As a result of this new organization, the new plugin has gained a few new features:
  - "As-is" and non-autotagged imports can now have album art imported from the local filesystem (although Web repositories are still not searched in these cases).
  - A new command, `beet fetchart`, allows you to download album art post-import. If you only want to fetch art manually, not automatically during import, set the new plugin's `autofetch` option to `no`.
  - New album art sources have been added.
- Errors when communicating with MusicBrainz now log an error message instead of halting the importer.
- Similarly, filesystem manipulation errors now print helpful error messages instead of a messy traceback. They still interrupt beets, but they should now be easier for users to understand. Tracebacks are still available in verbose mode.
- New metadata fields for artist credits: `artist_credit` and `albumartist_credit` can now contain release- and recording-specific variations of the artist's name. See *Available Values*.
- Revamped the way beets handles concurrent database access to avoid nondeterministic SQLite-related crashes when using the multithreaded importer. On systems where SQLite was compiled without `usleep(3)` support, multithreaded database access could cause an internal error (with the message "database is locked"). This release synchronizes access to the database to avoid internal SQLite contention, which should avoid this error.
- Plugins can now add parallel stages to the import pipeline. See *Writing Plugins*.
- Beets now prints out an error when you use an unrecognized field name in a query: for example, when running `beet ls -a artist:foo` (because `artist` is an item-level field).
- New plugin events:
  - `import_task_choice` is called after an import task has an action assigned.
  - `import_task_files` is called after a task's file manipulation has finished (copying or moving files, writing metadata tags).
  - `library_opened` is called when beets starts up and opens the library database.
- *LastGenre Plugin*: Fixed a problem where path formats containing `$genre` would use the old genre instead of the newly discovered one.

- Fix a crash when moving files to a Samba share.
- *MPDUpdate Plugin*: Fix TypeError crash (thanks to Philippe Mongeau).
- When re-importing files with `import_copy` enabled, only files inside the library directory are moved. Files outside the library directory are still copied. This solves a problem (introduced in 1.0b14) where beets could crash after adding files to the library but before finishing copying them; during the next import, the (external) files would be moved instead of copied.
- Artist sort names are now populated correctly for multi-artist tracks and releases. (Previously, they only reflected the first artist.)
- When previewing changes during import, differences in track duration are now shown as "2:50 vs. 3:10" rather than separated with -> like track numbers. This should clarify that beets isn't doing anything to modify lengths.
- Fix a problem with query-based path format matching where a field-qualified pattern, like `albumtype_soundtrack`, would match everything.
- *Chromaprint/Acoustid Plugin*: Fix matching with ambiguous Acoustids. Some Acoustids are identified with multiple recordings; beets now considers any associated recording a valid match. This should reduce some cases of errant track reordering when using chroma.
- Fix the ID3 tag name for the catalog number field.
- *Chromaprint/Acoustid Plugin*: Fix occasional crash at end of fingerprint submission and give more context to "failed fingerprint generation" errors.
- Interactive prompts are sent to stdout instead of stderr.
- *EmbedArt Plugin*: Fix crash when audio files are unreadable.
- *BPD Plugin*: Fix crash when sockets disconnect (thanks to Matteo Mecucci).
- Fix an assertion failure while importing with moving enabled when the file was already at its destination.
- Fix Unicode values in the `replace` config option (thanks to Jakob Borg).
- Use a nicer error message when input is requested but stdin is closed.
- Fix errors on Windows for certain Unicode characters that can't be represented in the MBCS encoding. This required a change to the way that paths are represented in the database on Windows; if you find that beets' paths are out of sync with your filesystem with this release, delete and recreate your database with `beet import -AWC /path/to/music`.
- Fix `import` with relative path arguments on Windows.

1.6.20 1.0b14 (May 12, 2012)

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.

- 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 *Configuration* 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.

1.6.21 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).

1.6.22 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 *Configuration*.
- 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.

1.6.23 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 *Configuration*.
- 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.

1.6.24 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.

1.6.25 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.

1.6.26 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.

1.6.27 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.

1.6.28 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.

1.6.29 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.

1.6.30 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.

1.6.31 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 *Configuration* 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!

1.6.32 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.

1.6.33 1.0b1 (June 17, 2010)

Initial release.