beets Documentation

Release 1.6.0

Adrian Sampson

Jan 29, 2022

Contents

1 Contents **3**
- 1.1 Guides . . . 3
- 1.2 Reference . . . 14
- 1.3 Plugins . . . 45
- 1.4 FAQ . . . 119
- 1.5 Contributing . . . 124
- 1.6 For Developers . . . 129
- 1.7 Changelog . . . 144

Index **211**

Welcome to the documentation for beets, the media library management system for obsessive 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 Libera.Chat, drop by the discussion board, 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 works on Python 3.6 or later.

- **macOS** 11 (Big Sur) includes Python 3.8 out of the box. You can opt for a more recent Python installing it via Homebrew (`brew install python3`). There's also a MacPorts port. Run `port install beets` or `port install beets-full` to include many third-party plugins.
- 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**, there's a beets port can be installed with `pkg_add beets`.

- For **Slackware**, there's a SlackBuild available.
- On **Fedora** 22 or later, there's a DNF package you can install with `sudo dnf install beets beets-plugins beets-doc`.
- On **Solus**, run `eopkg install beets`.
- On **NixOS**, there's a package you can install with `nix-env -i beets`.

If you have pip, just say `pip install beets` (or `pip install --user beets` if you run into permissions problems).

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 by Hand on macOS 10.11 and Higher

Starting with version 10.11 (El Capitan), macOS has a new security feature called System Integrity Protection (SIP) that prevents you from modifying some parts of the system. This means that some `pip` commands may fail with a permissions error. (You probably *won't* run into this if you've installed Python yourself with Homebrew or otherwise. You can also try MacPorts.)

If this happens, you can install beets for the current user only by typing `pip install --user beets`. If you do that, you might want to add `~/Library/Python/3.6/bin` to your `$PATH`.

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 3.6). The installer should give you the option to "add Python to PATH." Check this box. If you do that, you can skip the next step.
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:\Python36;C:\Python36\Scripts`. You may need to adjust these paths to point to your Python installation.
3. Now install beets by running: `pip install beets`
4. 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. Download the beets.reg file and open it in a text file to make sure the paths to Python match your system. Then double-click the file add the necessary keys to your registry. You can then right-click a directory and choose "Import with beets".

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.db
```

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 config's format is YAML. You'll want to configure your text editor to use spaces, not real tabs, for indentation. Also, ~ means your home directory in these paths, even on Windows.)

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

The next step is to import your music files into the beets library database. Because this can involve modifying files and moving them around, data loss is always a possibility, so now would be a good time to make sure you have a recent backup of all your music. We'll wait.

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
```

By default, a search term will match any of a handful of *common 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 also has an `-a` option, which searches for albums instead of songs:

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

There's also an `-f` option (for *format*) that lets you specify what gets displayed in the results of a search:

```
$ beet ls -a forever -f "[$format] $album ($year) - $artist - $title"
[MP3] For Emma, Forever Ago (2009) - Bon Iver - Flume
[AAC] Freezepop Forever (2011) - Freezepop - Harebrained Scheme
```

In the format option, field references like *$format* and *$year* are filled in with data from each result. You can see a full list of available fields by running `beet fields`.

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
```

(continues on next page)

(continued from previous page)

```
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 discussion board 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* or *I* options 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 -m`: move imported files to your music directory (overrides the `-c` option)
- `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, enter Id, 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, I 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.
- *I*: Enter a metadata backend ID to use as search in the database. Use this option to specify a backend entity (for example, a MusicBrainz release or recording) directly, by pasting its ID or the full URL. You can also specify several IDs by separating them by a space.
- *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 all, Remove old, Merge all?
```

Beets wants to keep you safe from duplicates, which can be a real pain, so you have four 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; you can remove the existing music and choose the new stuff; or you can merge all the new and old tracks into a single album. If you choose that "remove" 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 "merge", beets will try re-importing the existing and new tracks as one bundle together. This is particularly helpful when you have an album that's missing some tracks and then want to import the remaining songs. The importer will ask you the same questions as it would if you were importing all tracks at once.

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 we haven't made the process clear, please post on the discussion board and we'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.

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 to store this. Set the field using the *modify* command:

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

By default beets will show you the changes that are about to be applied and ask if you really want to apply them to all, some or none of the items or albums. You can type y for "yes", n for "no", or s for "select". If you choose the latter, the command will prompt you for each individual matching item or album.

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.

Automatically add new music to your library

As a command-line tool, beets is perfect for automated operation via a cron job or the like. To use it this way, you might want to use these options in your *config file*:

```
import:
    incremental: yes
    quiet: yes
    log: /path/to/log.txt
```

The *incremental* option will skip importing any directories that have been imported in the past. *quiet* avoids asking you any questions (since this will be run automatically, no input is possible). You might also want to use the *quiet_fallback* options to configure what should happen when no near-perfect match is found – this option depends on your level of paranoia. Finally, *log* will make beets record its decisions so you can come back later and see what you need to handle manually.

The last step is to set up cron or some other automation system to run `beet import /path/to/incoming/music`.

Useful reports

Since beets has a quite powerful query tool, this list contains some useful and powerful queries to run on your library.

- See a list of all albums which have files which are 128 bit rate:

```
beet list bitrate:128000
```

- See a list of all albums with the tracks listed in order of bit rate:

```
beet ls -f '$bitrate $artist - $title' bitrate+
```

- See a list of albums and their formats:

```
beet ls -f '$albumartist $album $format' | sort | uniq
```

Note that `beet ls --album -f '... $format'` doesn't do what you want, because `format` is an item-level field, not an album-level one. If an album's tracks exist in multiple formats, the album will appear in the list once for each format.

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. To extract *7z* files, install the py7zr package.

Optional command flags:

- By default, the command copies files to your library directory and updates the ID3 tags on your music. In order to move the files, instead of copying, use the `-m` (move) option. 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.
- When beets applies metadata to your music, it will retain the value of any existing tags that weren't overwritten, and import them into the database. You may prefer to only use existing metadata for finding matches, and to erase it completely when new metadata is applied. You can enforce this behavior with the `--from-scratch` option, or the `from_scratch` 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.
- If you want to preview which files would be imported, use the `--pretend` option. If set, beets will just print a list of files that it would otherwise import.
- If you already have a metadata backend ID that matches the items to be imported, you can instruct beets to restrict the search to that ID instead of searching for other candidates by using the `--search-id SEARCH_ID` option. Multiple IDs can be specified by simply repeating the option several times.
- You can supply `--set field=value` to assign *field* to *value* on import. These assignments will merge with (and possibly override) the *set_fields* configuration dictionary. You can use the option multiple times on the command line, like so:

```
beet import --set genre="Alternative Rock" --set mood="emotional"
```

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`. You can also specify the sort order. (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: $albumtotal' 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 [-adf] QUERY
```

Remove music from your library.

This command uses the same *query* syntax as the `list` command. By default, it just removes entries from the library database; it doesn't touch the files on disk. To actually delete the files, use the `-d` flag. When the `-a` flag is given, the command operates on albums instead of individual tracks.

When you run the `remove` command, it prints a list of all affected items in the library and asks for your permission before removing them. You can then choose to abort (type *n*), confirm (*y*), or interactively choose some of the items (*s*). In the latter case, the command will prompt you for every matching item or album and invite you to type *y* to remove the item/album, *n* to keep it or *q* to exit and only remove the items/albums selected up to this point. This option lets you choose precisely which tracks/albums to remove without spending too much time to carefully craft a query. If you do not want to be prompted at all, use the `-f` option.

modify

```
beet modify [-MWay] [-f FORMAT] 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. Without this flag, the command will only change *track-level* data, even if all the tracks belong to the same album. If you want to change an *album-level* field, such as `year` or `albumartist`, you'll want to use the `-a` flag to avoid a confusing situation where the data for individual tracks conflicts with the data for the whole album.

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).

When you run the `modify` command, it prints a list of all affected items in the library and asks for your permission before making any changes. You can then choose to abort the change (type *n*), confirm (*y*), or interactively choose some of the items (*s*). In the latter case, the command will prompt you for every matching item or album and invite you to type *y* to apply the changes, *n* to discard them or *q* to exit and apply the selected changes. This option lets you choose precisely which data to change without spending too much time to carefully craft a query. To skip the prompts entirely, use the `-y` option.

move

```
beet move [-capt] [-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. The `-e` flag (for "export") copies files without changing the database.

To perform a "dry run", just use the `-p` (for "pretend") flag. This will show you a list of files that would be moved but won't actually change anything on disk. The `-t` option sets the timid mode which will ask again before really moving or copying the files.

update

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

Update the library (and, by default, 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`. Beets will skip files if their modification times have not changed, so any out-of-band metadata changes must also update these for `beet update` to recognise that the files have been edited.

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.

By default, all the changed metadata will be populated back to the database. If you only want certain fields to be written, specify them with the `` `-F` `` flags (which can be used multiple times). For the list of supported fields, please see `` `beet fields` ``.

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).

By default, the command calculates file sizes using their bitrate and duration. The `-e` (`--exact`) option reads the exact sizes of each file (but is slower). The exact mode also outputs the exact duration in seconds.

fields

```
beet fields
```

Show the item and album metadata fields available for use in *Queries* and *Path Formats*. The listing includes any template fields provided by plugins and any flexible attributes you've manually assigned to your items and albums.

config

```
beet config [-pdc]
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.
- By default, sensitive information like passwords is removed when dumping the configuration. The `--clear` option includes this sensitive data.
- 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. You can use it twice, as in `-vv`, to make beets even more verbose.
- `-c FILE`: read a specified YAML *configuration file*. This configuration works as an overlay: rather than replacing your normal configuration options entirely, the two are merged. Any individual options set in this config file will override the corresponding settings in your base configuration.
- `-p plugins`: specify a comma-separated list of plugins to enable. If specified, the plugin list in your configuration is ignored. The long form of this argument also allows specifying no plugins, effectively disabling all plugins: `--plugins=`.

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.

zsh

If you use zsh, take a look at the included completion script. The script should be placed in a directory that is part of your `fpath`, and *not* sourced in your `.zshrc`. Running `echo $fpath` will give you a list of valid directories.

Another approach is to use zsh's bash completion compatibility. This snippet defines some bash-specific functions to make this work without errors:

```
autoload bashcompinit
bashcompinit
_get_comp_words_by_ref() { :; }
compopt() { :; }
_filedir() { :; }
eval "$(beet completion)"
```

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*
  - *library*
  - *directory*
  - *plugins*
  - *include*
  - *pluginpath*
  - *ignore*
  - *ignore_hidden*
  - *replace*
  - *path_sep_replace*
  - *asciify_paths*
  - *art_filename*
  - *threaded*
  - *format_item*
  - *format_album*
  - *sort_item*
  - *sort_album*
  - *sort_case_insensitive*
  - *original_date*
  - *artist_credit*
  - *per_disc_numbering*
  - *aunique*

    - *terminal_encoding*
    - *clutter*
    - *max_filename_length*
    - *id3v23*
    - *va_name*
- *UI Options*
    - *color*
    - *colors*
- *Importer Options*
    - *write*
    - *copy*
    - *move*
    - *link*
    - *hardlink*
    - *reflink*
    - *resume*
    - *incremental*
    - *incremental_skip_later*
    - *from_scratch*
    - *quiet*
    - *quiet_fallback*
    - *none_rec_action*
    - *timid*
    - *log*
    - *default_action*
    - *languages*
    - *detail*
    - *group_albums*
    - *autotag*
    - *duplicate_action*
    - *bell*
    - *set_fields*
- *MusicBrainz Options*
    - *searchlimit*
    - *extra_tags*

    - *genres*
- *Autotagger Matching Options*
    - *max_rec*
    - *preferred*
    - *ignored*
    - *required*
    - *ignored_media*
    - *ignore_data_tracks*
    - *ignore_video_tracks*
- *Path Format Configuration*
- *Configuration Location*
    - *Environment Variable*
    - *Command-Line Option*
    - *Default Location*
- *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*.

include

A space-separated list of extra configuration files to include. Filenames are relative to the directory containing `config.yaml`.

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 `.*`, `*~`, `System Volume Information`, `lost+found` (i.e., beets ignores Unix-style hidden files, backup files, and directories that appears at the root of some Linux and Windows filesystems).

ignore_hidden

Either `yes` or `no`; whether to ignore hidden files when importing. On Windows, the "Hidden" property of files is used to detect whether or not a file is hidden. On OS X, the file's "IsHidden" flag is used to detect whether or not a file is hidden. On both OS X and other platforms (excluding Windows), files (and directories) starting with a dot are detected as hidden files.

replace

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

When replacements other than the defaults are used, it is possible that they will increase the length of the path. In the scenario where this leads to a conflict with the maximum filename length, the default replacements will be used to resolve the conflict and beets will display a warning.

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.

path_sep_replace

A string that replaces the path separator (for example, the forward slash / on Linux and MacOS, and the backward slash \\ on Windows) when generating filenames with beets. This option is related to *replace*, but is distict from it for technical reasons.

Warning: Changing this option is potentially dangerous. For example, setting it to the actual path separator could create directories in unexpected locations. Use caution when changing it and always try it out on a small number of files before applying it to your whole library.

Default: _.

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*.

This uses the unidecode module which is language agnostic, so some characters may be transliterated from a different language than expected. For example, Japanese kanji will usually use their Chinese readings.

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 substantially faster by overlapping work: for example, it can copy files for one album in parallel with looking up data in MusicBrainz for a different album. You may want to disable this when debugging problems with the autotagger. Defaults to `yes`.

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.

It used to be named *list_format_item*.

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.

It used to be named *list_format_album*.

sort_item

Default sort order to use when fetching items from the database. Defaults to `artist+ album+ disc+ track+`. Explicit sort orders override this default.

sort_album

Default sort order to use when fetching albums from the database. Defaults to `albumartist+ album+`. Explicit sort orders override this default.

sort_case_insensitive

Either `yes` or `no`, indicating whether the case should be ignored when sorting lexicographic fields. When set to `no`, lower-case values will be placed after upper-case values (e.g., *Bar Qux foo*), while `yes` would result in the more expected *Bar foo Qux*. Default: `yes`.

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`.

artist_credit

Either `yes` or `no`, indicating whether matched tracks and albums should use the artist credit, rather than the artist. That is, if this option is turned on, then `artist` will contain the artist as credited on the release.

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 (non-pregap) 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
```

When this option is off (the default), even "pregap" hidden tracks are numbered from one, not zero, so other track numbers may appear to be bumped up by one. When it is on, the pregap track for each disc can be numbered zero.

aunique

These options are used to generate a string that is guaranteed to be unique among all albums in the library who share the same set of keys.

The defaults look like this:

```
aunique:
    keys: albumartist album
    disambiguators: albumtype year label catalognum albumdisambig releasegroupdisambig
    bracket: '[]'
```

See *Album Disambiguation* for more details.

terminal_encoding

The text encoding, as known to Python, to use for messages printed to the standard output. It's also used to read messages from the standard input. 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".

The importer only removes recursively searched subdirectories—the top-level directory you specify on the command line is never deleted.

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.

va_name

Sets the albumartist for various-artist compilations. Defaults to `'Various Artists'` (the MusicBrainz standard). Affects other sources, such as *Discogs Plugin*, too.

UI Options

The options that allow for customization of the visual appearance of the console interface.

These options are available in this section:

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.

Note: The *color* option was previously a top-level configuration. This is still respected, but a deprecation message will be shown until your top-level *color* configuration has been nested under *ui*.

---

colors

The colors that are used throughout the user interface. These are only used if the `color` option is set to `yes`. For example, you might have a section in your configuration file that looks like this:

```
ui:
    color: yes
    colors:
        text_success: green
        text_warning: yellow
        text_error: red
        text_highlight: red
        text_highlight_minor: lightgray
        action_default: turquoise
        action: blue
```

Available colors: black, darkred, darkgreen, brown (darkyellow), darkblue, purple (darkmagenta), teal (darkcyan), lightgray, darkgray, red, green, yellow, blue, fuchsia (magenta), turquoise (cyan), white

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.

link

Either `yes` or `no`, indicating whether to use symbolic links instead of moving or copying files. (It conflicts with the `move`, `copy` and `hardlink` options.) Defaults to `no`.

This option only works on platforms that support symbolic links: i.e., Unixes. It will fail on Windows.

It's likely that you'll also want to set `write` to `no` if you use this option to preserve the metadata on the linked files.

hardlink

Either `yes` or `no`, indicating whether to use hard links instead of moving, copying, or symlinking files. (It conflicts with the `move`, `copy`, and `link` options.) Defaults to `no`.

As with symbolic links (see *link*, above), this will not work on Windows and you will want to set `write` to `no`. Otherwise, metadata on the original file will be modified.

reflink

Either `yes`, `no`, or `auto`, indicating whether to use copy-on-write file clones (a.k.a. "reflinks") instead of copying or moving files. The `auto` option uses reflinks when possible and falls back to plain copying when necessary. Defaults to `no`.

This kind of clone is only available on certain filesystems: for example, btrfs and APFS. For more details on filesystem support, see the pyreflink documentation. Note that you need to install `pyreflink`, either through `python -m pip install beets[reflink]` or `python -m pip install reflink`.

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).

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`.

incremental_skip_later

Either `yes` or `no`, controlling whether skipped directories are recorded in the incremental list. When set to `yes`, skipped directories won't be recorded, and beets will try to import them again later. When set to `no`, skipped directories will be recorded, and skipped later. Defaults to `no`.

from_scratch

Either `yes` or `no` (default), controlling whether existing metadata is discarded when a match is applied. This corresponds to the `--from_scratch` flag to `beet import`.

quiet

Either `yes` or `no` (default), controlling whether to ask for a manual decision from the user when the importer is unsure how to proceed. This corresponds to the `--quiet` 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. You can use a space-separated list of language abbreviations, like `en jp es`, to specify a preference order. 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.

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`.

duplicate_action

Either `skip`, `keep`, `remove`, `merge` or `ask`. Controls how duplicates are treated in import task. "skip" means that new item(album or track) will be skipped; "keep" means keep both old and new items; "remove" means remove old item; "merge" means merge into one album; "ask" means the user should be prompted for the action each time. The default is `ask`.

bell

Ring the terminal bell to get your attention when the importer needs your input.

Default: `no`.

set_fields

A dictionary indicating fields to set to values for newly imported music. Here's an example:

```
set_fields:
    genre: 'To Listen'
    collection: 'Unordered'
```

Other field/value pairs supplied via the `--set` option on the command-line override any settings here for fields with the same name.

Fields are set on both the album and each individual track of the album. Fields are persisted to the media files of each track.

Default: `{}` (empty).

MusicBrainz Options

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

```
musicbrainz:
    host: localhost:5000
    https: no
    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 `https` key makes the client use HTTPS instead of HTTP. This setting applies only to custom servers. The official MusicBrainz server always uses HTTPS. (Default: no.) The server must have search indices enabled (see Building search indexes).

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.

searchlimit

The number of matches returned when sending search queries to the MusicBrainz server.

Default: `5`.

extra_tags

By default, beets will use only the artist, album, and track count to query MusicBrainz. Additional tags to be queried can be supplied with the `extra_tags` setting. For example:

```
musicbrainz:
    extra_tags: [year, catalognum, country, media, label]
```

This setting should improve the autotagger results if the metadata with the given tags match the metadata returned by MusicBrainz.

Note that the only tags supported by this setting are the ones listed in the above example.

Default: `[]`

genres

Use MusicBrainz genre tags to populate (and replace if it's already set) the `genre` tag. This will make it a list of all the genres tagged for the release and the release-group on MusicBrainz, separated by "; " and sorted by the total number of votes. Default: `no`

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 below 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.

For example, setting `ignored: missing_tracks` will skip any album matches where your audio files are missing some of the tracks. The importer will not attempt to display these matches. It does not ignore the fact that the album is missing tracks, which would allow these matches to apply more easily. To do that, you'll want to adjust the penalty for missing tracks.

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.

ignored_media

A list of media (i.e., formats) in metadata databases to ignore when matching music. You can use this to ignore all media that usually contain video instead of audio, for example:

```
match:
    ignored_media: ['Data CD', 'DVD', 'DVD-Video', 'Blu-ray', 'HD-DVD',
                    'VCD', 'SVCD', 'UMD', 'VHS']
```

No formats are ignored by default.

ignore_data_tracks

By default, audio files contained in data tracks within a release are included in the album's tracklist. If you want them to be included, set it `no`.

Default: `yes`.

ignore_video_tracks

By default, video tracks within a release will be ignored. If you want them to be included (for example if you would like to track the audio-only versions of the video tracks), set it to `no`.

Default: `yes`.

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 ensures 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:

```
directory: /var/mp3
import:
    copy: yes
```

(continues on next page)

(continued from previous page)

```
    write: yes
    log: beetslog.txt
art_filename: albumart
plugins: bpd
pluginpath: ~/beets/myplugins
ui:
    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,brackets}`: 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{$added,%Y}`.
- `%first{text}`: Returns the first item, separated by `;` (a semicolon followed by a space). You can use `%first{text,count,skip}`, where `count` is the number of items (default 1) and `skip` is number to skip (default 0). You can also use `%first{text,count,skip,sep,join}` where `sep` is the separator, like `;` or `/` and join is the text to concatenate the items.
- `%ifdef{field}`, `%ifdef{field,truetext}` or `%ifdef{field,truetext,falsetext}`: Checks if an flexible attribute `field` is defined. If it exists, then return `truetext` or `field` (default). Otherwise, returns `falsetext`. The `field` should be entered without `$`. Note that this doesn't work with built-in *Available Values*, as they are always defined.

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 three arguments. The first two are whitespace-separated lists of album field names: a set of *identifiers* and a set of *disambiguators*. The third argument is a pair of characters used to surround the disambiguator.

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 releasegroupdisambig`. 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.

The default characters used as brackets are `[]`. To change this, provide a third argument to the `%aunique` function consisting of two characters: the left and right brackets. Or, to turn off bracketing entirely, leave argument blank.

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 `$`, `%`, `{`, `}`, and `,` are "special" in the path template syntax. This means that, for example, if you want a `%` character to appear in your paths, you'll need to be careful that you don't accidentally write a function call. To escape any of these characters (except `{`, and `,` outside a function argument), prefix it with a `$`. For example, `$$` becomes `$`; `$%` becomes `%`, etc. The only exceptions are:

- `${`, which is ambiguous with the variable reference syntax (like `${title}`). To insert a `{` alone, it's always sufficient to just type `{`.
- commas are used as argument separators in function calls. Inside of a function's argument, use `$,` to get a literal `,` character. Outside of any function argument, escaping is not necessary: `,` by itself will produce `,` in the output.

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_releasetrackid
- 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`. By default any unadorned word like this matches in a track's title, artist, album name, album artist, genre and comments. See below on how to search other fields.

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.

Keywords can also be joined with a Boolean "or" using a comma. For example, the command:

```
$ beet ls magnetic tomorrow , beatles yesterday
```

will match both "The House of Tomorrow" by the Magnetic Fields, as well as "Yesterday" by The Beatles. Note that the comma has to be followed by a space (e.g., `foo,bar` will be treated as a single keyword, *not* as an OR-query).

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.

Most command-line shells will try to interpret common characters in regular expressions, such as `()[]|`. To type those characters, you'll need to escape them (e.g., with backslashes or quotation marks, depending on your shell).

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
```

The `length` field also lets you use a "M:SS" format. For example, this query finds tracks that are less than four and a half minutes in length:

```
$ beet list length:..4:30
```

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'
```

You can also add an optional time value to date queries, specifying hours, minutes, and seconds.

Times are separated from dates by a space, an uppercase 'T' or a lowercase 't', for example: `2008-12-01T23:59:59`. If you specify a time, then the date must contain a year, month, and day. The minutes and seconds are optional.

Here is an example that finds all items added on 2008-12-01 at or after 22:00 but before 23:00:

```
$ beet ls 'added:2008-12-01T22'
```

To find all items added on or after 2008-12-01 at 22:45:

```
$ beet ls 'added:2008-12-01T22:45..'
```

To find all items added on 2008-12-01, at or after 22:45:20 but before 22:45:41:

```
$ beet ls 'added:2008-12-01T22:45:20..2008-12-01T22:45:40'
```

Here are example of the three ways to separate dates from times. All of these queries do the same thing:

```
$ beet ls 'added:2008-12-01T22:45:20'
$ beet ls 'added:2008-12-01t22:45:20'
$ beet ls 'added:2008-12-01 22:45:20'
```

You can also use *relative* dates. For example, `-3w` means three weeks ago, and `+4d` means four days in the future. A relative date has three parts:

- Either + or −, to indicate the past or the future. The sign is optional; if you leave this off, it defaults to the future.
- A number.
- A letter indicating the unit: `d`, `w`, `m` or `y`, meaning days, weeks, months or years. (A "month" is always 30 days and a "year" is always 365 days.)

Here's an example that finds all the albums added since last week:

```
$ beet ls -a 'added:-1w..'
```

And here's an example that lists items added in a two-week period starting four weeks ago:

```
$ beet ls 'added:-6w..-4w'
```

Query Term Negation

Query terms can also be negated, acting like a Boolean "not," by prefixing them with − or `^`. This has the effect of returning all the items that do **not** match the query term. For example, this command:

```
$ beet list ^love
```

matches all the songs in the library that do not have "love" in any of their fields.

Negation can be combined with the rest of the query mechanisms, so you can negate specific fields, regular expressions, etc. For example, this command:

```
$ beet list -a artist:dylan ^year:1980..1989 "^album::the(y)?"
```

matches all the albums with an artist containing "dylan", but excluding those released in the eighties and those that have "the" or "they" on the title.

The syntax supports both `^` and `-` as synonyms because the latter indicates flags on the command line. To use a minus sign in a command-line query, use a double dash `--` to separate the options from the query:

```
$ beet list -a -- artist:dylan -year:1980..1990 "-album::the(y)?"
```

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 if that path exists, so this command is equivalent as long as `/my/music/directory` exist:

```
$ 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.

Path queries are case sensitive if the queried path is on a case-sensitive filesystem.

Sort Order

Queries can specify a sort order. Use the name of the *field* you want to sort on, followed by a `+` or `-` sign to indicate ascending or descending sort. For example, this command:

```
$ beet list -a year+
```

will list all albums in chronological order. You can also specify several sort orders, which will be used in the same order as they appear in your query:

```
$ beet list -a genre+ year+
```

This command will sort all albums by genre and, in each genre, in chronological order.

The `artist` and `albumartist` keys are special: they attempt to use their corresponding `artist_sort` and `albumartist_sort` fields for sorting transparently (but fall back to the ordinary fields when those are empty).

Lexicographic sorts are case insensitive by default, resulting in the following sort order: `Bar foo Qux`. This behavior can be changed with the *sort_case_insensitive* configuration option. Case sensitive sort will result in lower-case values being placed after upper-case values, e.g., `Bar Qux foo`.

Note that when sorting by fields that are not present on all items (such as flexible fields, or those defined by plugins) in *ascending* order, the items that lack that particular field will be listed at the *beginning* of the list.

You can set the default sorting behavior with the *sort_item* and *sort_album* configuration options.

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 convert 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`.

Each plugin has its own set of options that can be defined in a section bearing its name:

```
plugins: inline convert web

convert:
    auto: true
```

Some plugins have special dependencies that you'll need to install. The documentation page for each plugin will list them in the setup instructions. For some, you can use `pip`'s "extras" feature to install the dependencies, like this:

```
pip install beets[fetchart,lyrics,lastgenre]
```

1.3.2 Using Metadata Source Plugins

Some plugins provide sources for metadata in addition to MusicBrainz. These plugins share the following configuration option:

- **source_weight**: Penalty applied to matches during import. Set to 0.0 to disable. Default: `0.5`.

For example, to equally consider matches from Discogs and MusicBrainz add the following to your configuration:

```
plugins: discogs

discogs:
   source_weight: 0.0
```

AcousticBrainz Submit Plugin

The `absubmit` plugin lets you submit acoustic analysis results to the AcousticBrainz server.

Installation

The `absubmit` plugin requires the streaming_extractor_music program to run. Its source can be found on GitHub, and while it is possible to compile the extractor from source, AcousticBrainz would prefer if you used their binary (see the AcousticBrainz FAQ).

The `absubmit` plugin also requires requests, which you can install using pip by typing:

```
pip install requests
```

After installing both the extractor binary and requests you can enable the plugin `absubmit` in your configuration (see *Using Plugins*).

Submitting Data

To run the analysis program and upload its results, type:

```
beet absubmit [-f] [-d] [QUERY]
```

By default, the command will only look for AcousticBrainz data when the tracks don't already have it; the `-f` or `--force` switch makes it refetch data even when it already exists. You can use the `-d` or `--dry` swtich to check which files will be analyzed, before you start a longer period of processing.

The plugin works on music with a MusicBrainz track ID attached. The plugin will also skip music that the analysis tool doesn't support. streaming_extractor_music currently supports files with the extensions `mp3`, `ogg`, `oga`, `flac`, `mp4`, `m4a`, `m4r`, `m4b`, `m4p`, `aac`, `wma`, `asf`, `mpc`, `wv`, `spx`, `tta`, `3g2`, `aif`, `aiff` and `ape`.

Configuration

To configure the plugin, make a `absubmit:` section in your configuration file. The available options are:

- **auto**: Analyze every file on import. Otherwise, you need to use the `beet absubmit` command explicitly. Default: `no`
- **extractor**: The absolute path to the streaming_extractor_music binary. Default: search for the program in your `$PATH`
- **force**: Analyze items and submit of AcousticBrainz data even for tracks that already have it. Default: `no`.
- **pretend**: Do not analyze and submit of AcousticBrainz data but print out the items which would be processed. Default: `no`.

AcousticBrainz Plugin

The `acousticbrainz` plugin gets acoustic-analysis information from the AcousticBrainz project.

Enable the `acousticbrainz` plugin in your configuration (see *Using Plugins*) and run it by typing:

```
$ beet acousticbrainz [-f] [QUERY]
```

By default, the command will only look for AcousticBrainz data when the tracks doesn't already have it; the `-f` or `--force` switch makes it re-download data even when it already exists. If you specify a query, only matching tracks will be processed; otherwise, the command processes every track in your library.

For all tracks with a MusicBrainz recording ID, the plugin currently sets these fields:

- average_loudness
- bpm
- chords_changes_rate
- chords_key
- chords_number_rate
- chords_scale
- danceable
- gender
- genre_rosamerica
- initial_key (This is a built-in beets field, which can also be provided by *Key Finder Plugin*.)
- key_strength
- mood_acoustic
- mood_aggressive
- mood_electronic
- mood_happy
- mood_party
- mood_relaxed
- mood_sad
- moods_mirex
- rhythm
- timbre
- tonal
- voice_instrumental

Automatic Tagging

To automatically tag files using AcousticBrainz data during import, just enable the `acousticbrainz` plugin (see *Using Plugins*). When importing new files, beets will query the AcousticBrainz API using MBID and set the appropriate metadata.

Configuration

To configure the plugin, make a `acousticbrainz:` section in your configuration file. There are three options:

- **auto**: Enable AcousticBrainz during `beet import`. Default: `yes`.
- **force**: Download AcousticBrainz data even for tracks that already have it. Default: `no`.
- **tags**: Which tags from the list above to set on your files. Default: [] (all)

AlbumTypes Plugin

The `albumtypes` plugin adds the ability to format and output album types, such as "Album", "EP", "Single", etc. For the list of available album types, see the MusicBrainz documentation.

To use the `albumtypes` plugin, enable it in your configuration (see *Using Plugins*). The plugin defines a new field `$atypes`, which you can use in your path formats or elsewhere.

Configuration

To configure the plugin, make a `albumtypes:` section in your configuration file. The available options are:

- **types**: An ordered list of album type to format mappings. The order of the mappings determines their order in the output. If a mapping is missing or blank, it will not be in the output.
- **ignore_va**: A list of types that should not be output for Various Artists albums. Useful for not adding redundant information - various artist albums are often compilations.
- **bracket**: Defines the brackets to enclose each album type in the output.

The default configuration looks like this:

```
albumtypes:
    types:
        - ep: 'EP'
        - single: 'Single'
        - soundtrack: 'OST'
        - live: 'Live'
        - compilation: 'Anthology'
        - remix: 'Remix'
    ignore_va: compilation
    bracket: '[]'
```

Examples

With path formats configured like:

```
paths:
    default: $albumartist/[$year]$atypes $album/...
    albumtype:soundtrack Various Artists/$album [$year]$atypes/...
    comp: Various Artists/$album [$year]$atypes/...
```

The default plugin configuration generates paths that look like this, for example:

```
Aphex Twin/[1993][EP][Remix] On Remixes
Pink Floyd/[1995][Live] p·u·l·s·e
Various Artists/20th Century Lullabies [1999]
Various Artists/Ocean's Eleven [2001][OST]
```

AURA Plugin

This plugin is a server implementation of the AURA specification using the Flask framework. AURA is still a work in progress and doesn't yet have a stable version, but this server should be kept up to date. You are advised to read the *Issues* section.

Install

The `aura` plugin depends on Flask, which can be installed using `python -m pip install flask`. Then you can enable the `aura` plugin in your configuration (see *Using Plugins*).

It is likely that you will need to enable *Cross-Origin Resource Sharing (CORS)*, which introduces an additional dependency: flask-cors. This can be installed with `python -m pip install flask-cors`.

If Pillow is installed (`python -m pip install Pillow`) then the optional `width` and `height` attributes are included in image resource objects.

Usage

Use `beet aura` to start the AURA server. By default Flask's built-in server is used, which will give a warning about using it in a production environment. It is safe to ignore this warning if the server will have only a few users.

Alternatively, you can use `beet aura -d` to start the server in development mode, which will reload the server every time the AURA plugin file is changed.

You can specify the hostname and port number used by the server in your *configuration file*. For more detail see the *Configuration* section below.

If you would prefer to use a different WSGI server, such as gunicorn or uWSGI, then see *Using an External WSGI Server*.

AURA is designed to separate the client and server functionality. This plugin provides the server but not the client, so unless you like looking at JSON you will need a separate client. Currently the only client is AURA Web Client.

By default the API is served under http://127.0.0.1:8337/aura/. For example information about the track with an id of 3 can be obtained at http://127.0.0.1:8337/aura/tracks/3.

Note the absence of a trailing slash: http://127.0.0.1:8337/aura/tracks/3/ returns a `404 Not Found` error.

Configuration

To configure the plugin, make an `aura:` section in your configuration file. The available options are:

- **host**: The server hostname. Set this to `0.0.0.0` to bind to all interfaces. Default: `127.0.0.1`.
- **port**: The server port. Default: `8337`.
- **cors**: A YAML list of origins to allow CORS requests from (see *Cross-Origin Resource Sharing (CORS)*, below). Default: disabled.
- **cors_supports_credentials**: Allow authenticated requests when using CORS. Default: disabled.
- **page_limit**: The number of items responses should be truncated to if the client does not specify. Default `500`.

Cross-Origin Resource Sharing (CORS)

CORS allows browser clients to make requests to the AURA server. You should set the `cors` configuration option to a YAML list of allowed origins.

For example:

```
aura:
    cors:
        - http://www.example.com
        - https://aura.example.org
```

Alternatively you can set it to `'*'` to enable access from all origins. Note that there are security implications if you set the origin to `'*'`, so please research this before using it. Note the use of quote marks when allowing all origins. Quote marks are also required when the origin is `null`, for example when using `file:///`.

If the server is behind a proxy that uses credentials, you might want to set the `cors_supports_credentials` configuration option to true to let in-browser clients log in. Note that this option has not been tested, so it may not work.

Using an External WSGI Server

If you would like to use a different WSGI server (not Flask's built-in one), then you can! The `beetsplug.aura` module provides a WSGI callable called `create_app()` which can be used by many WSGI servers.

For example to run the AURA server using gunicorn use `gunicorn 'beetsplug.aura:create_app()'`, or for uWSGI use `uwsgi --http :8337 --module 'beetsplug.aura:create_app()'`. Note that these commands just show how to use the AURA app and you would probably use something a bit different in a production environment. Read the relevant server's documentation to figure out what you need.

Reverse Proxy Support

The plugin should work behind a reverse proxy without further configuration, however this has not been tested extensively. For details of what headers must be rewritten and a sample NGINX configuration see Flask proxy setups.

It is (reportedly) possible to run the application under a URL prefix (for example so you could have `/foo/aura/server` rather than `/aura/server`), but you'll have to work it out for yourself :-)

If using NGINX, do **not** add a trailing slash (`/`) to the URL where the application is running, otherwise you will get a 404. However if you are using Apache then you **should** add a trailing slash.

Issues

As of writing there are some differences between the specification and this implementation:

- Compound filters are not specified in AURA, but this server interprets multiple `filter` parameters as AND. See issue #19 for discussion.
- The `bitrate` parameter used for content negotiation is not supported. Adding support for this is doable, but the way Flask handles acceptable MIME types means it's a lot easier not to bother with it. This means an error could be returned even if no transcoding was required.

It is possible that some attributes required by AURA could be absent from the server's response if beets does not have a saved value for them. However, this has not happened so far.

Beets fields (including flexible fields) that do not have an AURA equivalent are not provided in any resource's attributes section, however these fields may be used for filtering.

The `mimetype` and `framecount` attributes for track resources are not supported. The first is due to beets storing the file type (e.g. `MP3`), so it is hard to filter by MIME type. The second is because there is no corresponding beets field.

Artists are defined by the `artist` field on beets Items, which means some albums have no `artists` relationship. Albums only have related artists when their beets `albumartist` field is the same as the `artist` field on at least one of it's constituent tracks.

The only art tracked by beets is a single cover image, so only albums have related images at the moment. This could be expanded to looking in the same directory for other images, and relating tracks to their album's image.

There are likely to be some performance issues, especially with larger libraries. Sorting, pagination and inclusion (most notably of images) are probably the main offenders. On a related note, the program attempts to import Pillow every time it constructs an image resource object, which is not good.

The beets library is accessed using a so called private function (with a single leading underscore) `beets.ui.__init__._open_library()`. This shouldn't cause any issues but it is probably not best practice.

Bad Files Plugin

The `badfiles` plugin adds a `beet bad` command to check for missing and corrupt files.

Configuring

First, enable the `badfiles` plugin (see *Using Plugins*). The default configuration defines the following default checkers, which you may need to install yourself:

- mp3val for MP3 files
- FLAC command-line tools for FLAC files

You can also add custom commands for a specific extension, like this:

```
badfiles:
    check_on_import: yes
    commands:
        ogg: myoggchecker --opt1 --opt2
        flac: flac --test --warnings-as-errors --silent
```

Custom commands will be run once for each file of the specified type, with the path to the file as the last argument. Commands must return a status code greater than zero for a file to be considered corrupt.

You can run the checkers when importing files by using the *check_on_import* option. When on, checkers will be run against every imported file and warnings and errors will be presented when selecting a tagging option.

Using

Type `beet bad` with a query according to beets' usual query syntax. For instance, this will run a check on all songs containing the word "wolf":

```
beet bad wolf
```

This one will run checks on a specific album:

```
beet bad album_id:1234
```

Here is an example where the FLAC decoder signals a corrupt file:

```
beet bad title::^$
/tank/Music/__/00.flac: command exited with status 1
  00.flac: *** Got error code 2:FLAC__STREAM_DECODER_ERROR_STATUS_FRAME_CRC_MISMATCH
  00.flac: ERROR while decoding data
              state = FLAC__STREAM_DECODER_READ_FRAME
```

Note that the default `mp3val` checker is a bit verbose and can output a lot of "stream error" messages, even for files that play perfectly well. Generally, if more than one stream error happens, or if a stream error happens in the middle of a file, this is a bad sign.

By default, only errors for the bad files will be shown. In order for the results for all of the checked files to be seen, including the uncorrupted ones, use the `-v` or `--verbose` option.

Bare-ASCII Search Plugin

The `bareasc` plugin provides a prefixed query that searches your library using simple ASCII character matching, with accented characters folded to their base ASCII character. This can be useful if you want to find a track with accented characters in the title or artist, particularly if you are not confident you have the accents correct. It is also not unknown for the accents to not be correct in the database entry or wrong in the CD information.

First, enable the plugin named `bareasc` (see *Using Plugins*). You'll then be able to use the `#` prefix to use bare-ASCII matching:

```
$ beet ls '#dvorak'
István Kertész - REQUIEM - Dvořàk: Requiem, op.89 - Confutatis maledictis
```

Command

In addition to the query prefix, the plugin provides a utility `bareasc` command. This command is **exactly** the same as the `beet list` command except that the output is passed through the bare-ASCII transformation before being printed. This allows you to easily check what the library data looks like in bare ASCII, which can be useful if you are trying to work out why a query is not matching.

Using the same example track as above:

```
$ beet bareasc 'Dvořàk'
Istvan Kertesz - REQUIEM - Dvorak: Requiem, op.89 - Confutatis maledictis
```

Note: the `bareasc` command does *not* automatically use bare-ASCII queries. If you want a bare-ASCII query you still need to specify the `#` prefix.

Notes

If the query string is all in lower case, the comparison ignores case as well as accents.

The default `bareasc` prefix (`#`) is used as a comment character in some shells so may need to be protected (for example in quotes) when typed into the command line.

The bare ASCII transliteration is quite simple. It may not give the expected output for all languages. For example, German u-umlaut `ü` is transformed into ASCII `u`, not into `ue`.

The bare ASCII transformation also changes Unicode punctuation like double quotes, apostrophes and even some hyphens. It is often best to leave out punctuation in the queries. Note that the punctuation changes are often not even visible with normal terminal fonts. You can always use the `bareasc` command to print the transformed entries and use a command like `diff` to compare with the output from the `list` command.

Configuration

To configure the plugin, make a `bareasc:` section in your configuration file. The only available option is:

- **prefix**: The character used to designate bare-ASCII queries. Default: `#`, which may need to be escaped in some shells.

Credits

The hard work in this plugin is done in Sean Burke's Unidecode library. Thanks are due to Sean and to all the people who created the Python version and the beets extensible query architecture.

Beatport Plugin

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 use the `beatport` plugin, first enable it in your configuration (see *Using Plugins*). Then, install the requests and requests_oauthlib libraries (which we need for querying and authorizing with the Beatport API) by typing:

```
pip install requests requests_oauthlib
```

You will also need to register for a Beatport account. The first time you run the *import* command after enabling the plugin, it will ask you to authorize with Beatport by visiting the site in a browser. On the site you will be asked to enter your username and password to authorize beets to query the Beatport API. You will then be displayed with a single line of text that you should paste as a whole into your terminal. This will store the authentication data for subsequent runs and you will not be required to repeat the above steps.

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. You can also search for an id like so:

beet import path/to/music/library –search-id id

Configuration

This plugin can be configured like other metadata source plugins as described in *Using Metadata Source Plugins*.

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 Homebrew. Run `brew install gstreamer gst-plugins-base pygobject3`.
- On Linux, you need to install GStreamer 1.0 and the GObject bindings for python. Under Ubuntu, they are called `python-gi` and `gstreamer1.0`.

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.

Usage

To use the `bpd` plugin, first enable it in your configuration (see *Using Plugins*). 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: Rigelian

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!

Configuration

To configure the plugin, make a `bpd:` section in your configuration file. The available options are:

- **host**: Default: Bind to all interfaces.
- **port**: Default: 6600
- **password**: Default: No password.
- **volume**: Initial volume, as a percentage. Default: 100
- **control_port**: Port for the internal control socket. Default: 6601

Here's an example:

```
bpd:
    host: 127.0.0.1
    port: 6600
    password: seekrit
    volume: 100
```

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.)

BPD plays music using GStreamer's `playbin` player, which has a simple API but doesn't support many advanced playback features.

Differences from the real MPD

BPD currently supports version 0.16 of the MPD protocol, but several of the commands and features are "pretend" implementations or have slightly different behaviour to their MPD equivalents. BPD aims to look enough like MPD that it can interact with the ecosystem of clients, but doesn't try to be a fully-fledged MPD replacement in terms of its playback capabilities.

These are some of the known differences between BPD and MPD:

- BPD doesn'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.
- Stored playlists aren't supported (BPD understands the commands though).
- 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 synchronously, whereas MPD does this in the background.
- Advanced playback features like cross-fade, ReplayGain and MixRamp are not supported due to BPD's simple audio player backend.
- Advanced query syntax is not currently supported.
- Clients can't use the `tagtypes` mask to hide fields.
- BPD's `random` mode is not deterministic and doesn't support priorities.
- Mounts and streams are not supported. BPD can only play files from disk.
- Stickers are not supported (although this is basically a flexattr in beets nomenclature so this is feasible to add).
- There is only a single password, and is enabled it grants access to all features rather than having permissions-based granularity.
- Partitions and alternative outputs are not supported; BPD can only play one song at a time.
- Client channels are not implemented.

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

To use the `bpm` plugin, first enable it in your configuration (see *Using 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 "-")
```

If *import.write* is `yes`, the song's tags are written to disk.

Configuration

To configure the plugin, make a `bpm:` section in your configuration file. The available options are:

- **max_strokes**: The maximum number of strokes to accept when tapping out the BPM. Default: 3.
- **overwrite**: Overwrite the track's existing BPM. Default: `yes`.

Credit

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

BPSync Plugin

This plugin provides the `bpsync` command, which lets you fetch metadata from Beatport for albums and tracks that already have Beatport IDs. This plugin works similarly to *MBSync Plugin*.

If you have downloaded music from Beatport, this can speed up the initial import if you just import "as-is" and then use `bpsync` to get up-to-date tags that are written to the files according to your beets configuration.

Usage

Enable the `bpsync` plugin in your configuration (see *Using Plugins*) and then run `beet bpsync 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.

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 divide overwhelmingly large folders into smaller subfolders by grouping albums or artists alphabetically (e.g. *A-F*, *G-M*, *N-Z*).

To use the `bucket` plugin, first enable it in your configuration (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 occurring 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.

The `%bucket` template function guesses whether to use alpha- or year-style buckets depending on the text it receives. It can guess wrong if, for example, an artist or album happens to begin with four digits. Provide `alpha` as the second argument to the template to avoid this automatic detection: for example, use `%bucket{$artist,alpha}`.

Configuration

To configure the plugin, make a `bucket:` section in your configuration file. The available options are:

- **bucket_alpha**: Ranges to use for all substitutions occurring on textual fields. Default: none.
- **bucket_alpha_regex**: A `range: regex` mapping (one per line) where `range` is one of the *bucket_alpha* ranges and `value` is a regex that overrides original range definition. Default: none.
- **bucket_year**: Ranges to use for all substitutions occurring on the `$year` field. Default: none.
- **extrapolate**: Enable this if you want to group your files into multiple year ranges without enumerating them all. This option will generate year bucket names by reproducing characteristics of declared buckets. Default: `no`

Here's an example:

```
bucket:
   bucket_year: ['2000-05']
   extrapolate: true
   bucket_alpha: ['A - D', 'E - L', 'M - R', 'S - Z']
   bucket_alpha_regex:
     'A - D': ^[0-9a-dA-D...äÄ]
```

This configuration creates five-year ranges for any input year. The *A - D* bucket now matches also all artists starting with ä or Ä and 0 to 9 and ... (ellipsis). The other alpha buckets work as ranges.

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 with PyGObject, FFmpeg, or MAD with pymad. How you install these will depend on your distribution. For example, on Ubuntu, run `apt-get install gstreamer1.0 python-gi`. On Arch Linux, you want `pacman -S gstreamer python2-gobject`. If you use GStreamer, be sure to install its codec plugins also (`gst-plugins-good`, etc.).

  Note that if you install beets in a virtualenv, you'll need it to have `--system-site-packages` enabled for Python to see the GStreamer bindings.
- On Windows, builds are provided by GStreamer

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 `gstreamer1.0-plugins-good`, `gstreamer1.0-plugins-bad`, and `gstreamer1.0-plugins-ugly`.

Usage

Once you have all the dependencies sorted out, enable the `chroma` plugin in your configuration (see *Using Plugins*) to benefit from fingerprinting the next time you run `beet import`. (The plugin doesn't produce any obvious output by default. If you want to confirm that it's enabled, you can try running in verbose mode once with `beet -v 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.

Configuration

There is one configuration option in the `chroma:` section, `auto`, which controls whether to fingerprint files during the import process. To disable fingerprint-based autotagging, set it to `no`, 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.

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

To use the `convert` plugin, first enable it in your configuration (see *Using 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`. The command will transcode all the files matching the query 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. Files that have been previously converted—and thus already exist in the destination directory—will be skipped.

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*.

Unless the `-y` (`--yes`) flag is set, the command will list all the items to be converted and ask for your confirmation.

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.

By default, files that do not need to be transcoded will be copied to their destination. Passing the `-l` (`--link`) flag creates symbolic links instead, passing `-H` (`--hardlink`) creates hard links. Note that album art embedding is disabled for files that are linked. Refer to the `link` and `hardlink` options below.

Configuration

To configure the plugin, make a `convert:` section in your configuration file. The available options are:

- **auto**: Import transcoded versions of your files automatically during imports. With this option enabled, the importer will transcode all (in the default configuration) non-MP3 files over the maximum bitrate before adding them to your library. Default: `no`.
- **tmpdir**: The directory where temporary files will be stored during import. Default: none (system default),
- **copy_album_art**: Copy album art when copying or transcoding albums matched using the `-a` option. Default: `no`.
- **album_art_maxwidth**: Downscale album art if it's too big. The resize operation reduces image width to at most `maxwidth` pixels while preserving the aspect ratio.
- **dest**: The directory where the files will be converted (or copied) to. Default: none.
- **embed**: Embed album art in converted items. Default: `yes`.
- **id3v23**: Can be used to override the global `id3v23` option. Default: `inherit`.
- **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. Default: none.
- **no_convert**: Does not transcode items matching provided query string (see *Queries*). (i.e. `format:AAC`, `format:WMA` or `path::\.(m4a|wma)$`)
- **never_convert_lossy_files**: Cross-conversions between lossy codecs—such as mp3, ogg vorbis, etc.—makes little sense as they will decrease quality even further. If set to `yes`, lossy files are always copied. Default: `no`.
- **paths**: The directory structure and naming scheme for the converted files. Uses the same format as the top-level `paths` section (see *Path Format Configuration*). Default: Reuse your top-level path format settings.
- **quiet**: Prevent the plugin from announcing every file it processes. Default: `false`.
- **threads**: The number of threads to use for parallel encoding. By default, the plugin will detect the number of processors available and use them all.
- **link**: By default, files that do not need to be transcoded will be copied to their destination. This option creates symbolic links instead. Note that options such as `embed` that modify the output files after the transcoding step will cause the original files to be modified as well if `link` is enabled. For this reason, album-art embedding is disabled for files that are linked. Default: `false`.
- **hardlink**: This options works similar to `link`, but it creates hard links instead of symlinks. This option overrides `link`. Only works when converting to a directory on the same filesystem as the library. Default: `false`.

- **delete_originals**: Transcoded files will be copied or moved to their destination, depending on the import configuration. By default, the original files are not modified by the plugin. This option deletes the original files after the transcoding step has completed. Default: `false`.

You can also configure the format to use for transcoding (see the next section):

- **format**: The name of the format to transcode to when none is specified on the command line. Default: `mp3`.
- **formats**: A set of formats and associated command lines for transcoding each.

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 optionally the file extension. `extension` is the filename extension to be used for newly transcoded files. If only the command is given as a string or the extension is not provided, the file extension defaults to the format's name. `command` is the command 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
```

Gapless MP3 encoding

While FFmpeg cannot produce "gapless" MP3s by itself, you can create them by using LAME directly. Use a shell script like this to pipe the output of FFmpeg into the LAME tool:

```
#!/bin/sh
ffmpeg -i "$1" -f wav - | lame -V 2 --noreplaygain - "$2"
```

Then configure the `convert` plugin to use the script:

```
convert:
    command: /path/to/script.sh $source $dest
    extension: mp3
```

This strategy configures FFmpeg to produce a WAV file with an accurate length header for LAME to use. Using `--noreplaygain` disables gain analysis; you can use the *ReplayGain Plugin* to do this analysis. See the LAME documentation and the HydrogenAudio wiki for other LAME configuration options and a thorough discussion of MP3 encoding.

Deezer Plugin

The `deezer` plugin provides metadata matches for the importer using the Deezer Album and Track APIs.

Basic Usage

First, enable the `deezer` plugin (see *Using Plugins*).

You can enter the URL for an album or song on Deezer at the `enter Id` prompt during import:

```
Enter search, enter Id, aBort, eDit, edit Candidates, plaY? i
Enter release ID: https://www.deezer.com/en/album/572261
```

Configuration

This plugin can be configured like other metadata source plugins as described in *Using Metadata Source Plugins*.

Discogs Plugin

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

Installation

To use the `discogs` plugin, first enable it in your configuration (see *Using Plugins*). Then, install the python3-discogs-client library by typing:

```
pip install python3-discogs-client
```

You will also need to register for a Discogs account, and provide authentication credentials via a personal access token or an OAuth2 authorization.

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.

OAuth Authorization

The first time you run the *import* command after enabling the plugin, it will ask you to authorize with Discogs by visiting the site in a browser. Subsequent runs will not require re-authorization.

Authentication via Personal Access Token

As an alternative to OAuth, you can get a token from Discogs and add it to your configuration. To get a personal access token (called a "user token" in the python3-discogs-client documentation), login to Discogs, and visit the Developer settings page. Press the `Generate new token` button, and place the generated token in your configuration, as the `user_token` config option in the `discogs` section.

Configuration

This plugin can be configured like other metadata source plugins as described in *Using Metadata Source Plugins*.

There is one additional option in the `discogs:` section, `index_tracks`. Index tracks (see the Discogs guidelines), along with headers, mark divisions between distinct works on the same release or within works. When `index_tracks` is enabled:

```
discogs:
    index_tracks: yes
```

beets will incorporate the names of the divisions containing each track into the imported track's title. For example, importing this album would result in track names like:

```
Messiah, Part I: No.1: Sinfony
Messiah, Part II: No.22: Chorus- Behold The Lamb Of God
Athalia, Act I, Scene I: Sinfonia
```

whereas with `index_tracks` disabled you'd get:

```
No.1: Sinfony
No.22: Chorus- Behold The Lamb Of God
Sinfonia
```

This option is useful when importing classical music.

Troubleshooting

Several issues have been encountered with the Discogs API. If you have one, please start by searching for a similar issue on the repo.

Here are two things you can try:

- Try deleting the token file (`~/.config/beets/discogs_token.json` by default) to force reauthorization.
- Make sure that your system clock is accurate. The Discogs servers can reject your request if your clock is too out of sync.

Duplicates Plugin

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

Usage

To use the `duplicates` plugin, first enable it in your configuration (see *Using Plugins*).

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 via command-line switches

```
-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
-s, --strict          report duplicates only if all attributes are set
-k, --key            report duplicates based on keys (can be used multiple times)
-M, --merge           merge duplicate items
-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
```

Configuration

To configure the plugin, make a `duplicates:` section in your configuration file. The available options mirror the command-line options:

- **album**: List duplicate albums instead of tracks. Default: `no`.
- **checksum**: Use an arbitrary command to compute a checksum of items. This overrides the `keys` option the first time it is run; however, because it caches the resulting checksum as `flexattrs` in the database, you can use `--key=name_of_the_checksumming_program --key=any_other_keys` (or set the `keys` configuration option) the second time around. Default: `ffmpeg -i {file} -f crc -`.
- **copy**: A destination base directory into which to copy matched items. Default: none (disabled).
- **count**: Print a count of duplicate tracks or albums in the format `$albumartist - $album - $title: $count` (for tracks) or `$albumartist - $album: $count` (for albums). Default: `no`.
- **delete**: Removes matched items from the library and from the disk. Default: `no`
- **format**: 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. Default: *format_item*
- **full**: List every track or album that has duplicates, not just the duplicates themselves. Default: `no`
- **keys**: Define 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. Default: `[mb_trackid, mb_albumid]`
- **merge**: Merge duplicate items by consolidating tracks and-or metadata where possible.
- **move**: A destination base directory into which it will move matched items. Default: none (disabled).
- **path**: Output the path instead of metadata when listing duplicates. Default: `no`.

- **strict**: Do not report duplicate matches if some of the attributes are not defined (ie. null or empty). Default: `no`
- **tag**: A `key=value` pair. The plugin will add a new `key` attribute with `value` value as a flexattr to the database for duplicate items. Default: `no`.
- **tiebreak**: Dictionary of lists of attributes keyed by `items` or `albums` to use when choosing duplicates. By default, the tie-breaking procedure favors the most complete metadata attribute set. If you would like to consider the lower bitrates as duplicates, for example, set `tiebreak: items: [bitrate]`. Default: `{}`.

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 -k albumartist -k 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 -k albumartist -k albumartist:nickelback
```

Tag duplicate items with some flag:

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

Ignore items with undefined keys:

```
beet duplicates --strict
```

Merge and delete duplicate albums with different missing tracks:

```
beet duplicates --album --merge --delete
```

Edit Plugin

The `edit` plugin lets you modify music metadata using your favorite text editor.

Enable the `edit` plugin in your configuration (see *Using Plugins*) and then type:

```
beet edit QUERY
```

Your text editor (i.e., the command in your `$EDITOR` environment variable) will open with a list of tracks to edit. Make your changes and exit your text editor to apply them to your music.

Command-Line Options

The `edit` command has these command-line options:

- `-a` or `--album`: Edit albums instead of individual items.
- `-f FIELD` or `--field FIELD`: Specify an additional field to edit (in addition to the defaults set in the configuration).
- `--all`: Edit *all* available fields.

Interactive Usage

The `edit` plugin can also be invoked during an import session. If enabled, it adds two new options to the user prompt:

```
[A]pply, More candidates, Skip, Use as-is, as Tracks, Group albums, Enter search,␣
↪enter Id, aBort, eDit, edit Candidates?
```

- `eDit`: use this option for using the original items' metadata as the starting point for your edits.
- `edit Candidates`: use this option for using a candidate's metadata as the starting point for your edits.

Please note that currently the interactive usage of the plugin will only allow you to change the item-level fields. In case you need to edit the album-level fields, the recommended approach is to invoke the plugin via the command line in album mode (`beet edit -a QUERY`) after the import.

Also, please be aware that the `edit Candidates` choice can only be used with the matches found during the initial search (and currently not supporting the candidates found via the `Enter search` or `enter Id` choices). You might find the `--search-id SEARCH_ID` *import* option useful for those cases where you already have a specific candidate ID that you want to edit.

Configuration

To configure the plugin, make an `edit:` section in your configuration file. The available options are:

- **itemfields**: A space-separated list of item fields to include in the editor by default. Default: `track title artist album`
- **albumfields**: The same when editing albums (with the `-a` option). Default: `album albumartist`

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).

Embedding Art Automatically

To automatically embed discovered album art into imported files, just enable the `embedart` 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 has its cover art set.

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

Image Similarity

When importing a lot of files with the `auto` option, one may be reluctant to overwrite existing embedded art for all of them.

You can tell beets to avoid embedding images that are too different from the existing ones. This works by computing the perceptual hashes (PHASH) of the two images and checking that the difference between the two does not exceed a threshold. You can set the threshold with the `compare_threshold` option.

A threshold of 0 (the default) disables similarity checking and always embeds new images. Set the threshold to another number—we recommend between 10 and 100—to adjust the sensitivity of the comparison. The smaller the threshold number, the more similar the images must be.

This feature requires ImageMagick.

Configuration

To configure the plugin, make an `embedart:` section in your configuration file. The available options are:

- **auto**: Enable automatic album art embedding. Default: `yes`.
- **compare_threshold**: How similar candidate art must be to existing art to be written to the file (see *Image Similarity*). Default: 0 (disabled).
- **ifempty**: Avoid embedding album art for files that already have art embedded. Default: `no`.
- **maxwidth**: A maximum width to downscale images before embedding them (the original image file is not altered). The resize operation reduces image width to at most `maxwidth` pixels. The height is recomputed so that the aspect ratio is preserved. See also *Image Resizing* for further caveats about image resizing. Default: 0 (disabled).
- **quality**: The JPEG quality level to use when compressing images (when `maxwidth` is set). This should be either a number from 1 to 100 or 0 to use the default quality. 65–75 is usually a good starting point. The default behavior depends on the imaging tool used for scaling: ImageMagick tries to estimate the input image quality and uses 92 if it cannot be determined, and PIL defaults to 75. Default: 0 (disabled)

- **remove_art_file**: Automatically remove the album art file for the album after it has been embedded. This option is best used alongside the *FetchArt* plugin to download art with the purpose of directly embedding it into the file's metadata without an "intermediate" album art file. Default: `no`.

Note: `compare_threshold` option requires ImageMagick, and `maxwidth` requires either ImageMagick or Pillow.

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. The command prompts for confirmation before making the change unless you specify the `-y` (`--yes`) option.
- `beet extractart [-a] [-n FILE] QUERY`: extracts the images for all albums matching the query. The images are placed inside the album folder. You can specify the destination file name using the `-n` option, but leave off the extension: it will be chosen automatically. The destination filename is specified using the `art_filename` configuration option. It defaults to `cover` if it's not specified via `-o` nor the config. Using `-a`, the extracted image files are automatically associated with the corresponding album.
- `beet extractart -o FILE QUERY`: extracts the image from an item matching the query and stores it in a file. You have to specify the destination file using the `-o` option, but leave off the extension: it will be chosen automatically.
- `beet clearart QUERY`: removes all embedded images from all items matching the query. The command prompts for confirmation before making the change unless you specify the `-y` (`--yes`) option.

EmbyUpdate Plugin

`embyupdate` is a plugin that lets you automatically update Emby's library whenever you change your beets library.

To use `embyupdate` plugin, enable it in your configuration (see *Using Plugins*). Then, you'll want to configure the specifics of your Emby server. You can do that using an `emby:` section in your `config.yaml`, which looks like this:

```
emby:
    host: localhost
    port: 8096
    username: user
    apikey: apikey
```

To use the `embyupdate` plugin you need to install the requests library with:

```
pip install requests
```

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

Configuration

The available options under the `emby:` section are:

- **host**: The Emby server host. You also can include `http://` or `https://`. Default: `localhost`

- **port**: The Emby server port. Default: 8096
- **username**: A username of a Emby user that is allowed to refresh the library.
- **apikey**: An Emby API key for the user.
- **password**: The password for the user. (This is only necessary if no API key is provided.)

You can choose to authenticate either with `apikey` or `password`, but only one of those two is required.

Export Plugin

The `export` plugin lets you get data from the items and export the content as JSON, CSV, or XML.

Enable the `export` plugin (see *Using Plugins* for help). Then, type `beet export` followed by a *query* to get the data from your library. For example, run this:

```
$ beet export beatles
```

to print a JSON file containing information about your Beatles tracks.

Command-Line Options

The `export` command has these command-line options:

- `--include-keys` or `-i`: Choose the properties to include in the output data. The argument is a comma-separated list of simple glob patterns where `*` matches any string. For example:

  ```
  $ beet export -i 'title,mb*' beatles
  ```

  will include the `title` property and all properties starting with `mb`. You can add the `-i` option multiple times to the command line.
- `--library` or `-l`: Show data from the library database instead of the files' tags.
- `--album` or `-a`: Show data from albums instead of tracks (implies `--library`).
- `--output` or `-o`: Path for an output file. If not informed, will print the data in the console.
- `--append`: Appends the data to the file instead of writing.
- `--format` or `-f`: Specifies the format the data will be exported as. If not informed, JSON will be used by default. The format options include csv, json, jsonlines and xml.

Configuration

To configure the plugin, make a `export:` section in your configuration file. For JSON export, these options are available under the `json` and `jsonlines` keys:

- **ensure_ascii**: Escape non-ASCII characters with `\uXXXX` entities.
- **indent**: The number of spaces for indentation.
- **separators**: A `[item_separator, dict_separator]` tuple.
- **sort_keys**: Sorts the keys in JSON dictionaries.

Those options match the options from the Python json module. Similarly, these options are available for the CSV format under the `csv` key:

- **delimiter**: Used as the separating character between fields. The default value is a comma (,).
- **dialect**: The kind of CSV file to produce. The default is *excel*.

These options match the options from the Python csv module.

The default options look like this:

```
export:
    json:
        formatting:
            ensure_ascii: false
            indent: 4
            separators: [',' , ': ']
            sort_keys: true
    csv:
        formatting:
            delimiter: ','
            dialect: excel
```

FetchArt Plugin

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

To use the `fetchart` 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 tries to get 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 embed the art into the files' tags, use the *EmbedArt Plugin*. (You'll want to have both plugins enabled.)

Configuration

To configure the plugin, make a `fetchart:` section in your configuration file. The available options are:

- **auto**: Enable automatic album art fetching during import. Default: `yes`.
- **cautious**: Pick only trusted album art by ignoring filenames that do not contain one of the keywords in `cover_names`. Default: `no`.
- **cover_names**: Prioritize images containing words in this list. Default: `cover front art album folder`.
- **minwidth**: Only images with a width bigger or equal to `minwidth` are considered as valid album art candidates. Default: 0.
- **maxwidth**: A maximum image width to downscale fetched images if they are too big. The resize operation reduces image width to at most `maxwidth` pixels. The height is recomputed so that the aspect ratio is preserved.

- **quality**: The JPEG quality level to use when compressing images (when `maxwidth` is set). This should be either a number from 1 to 100 or 0 to use the default quality. 65–75 is usually a good starting point. The default behavior depends on the imaging tool used for scaling: ImageMagick tries to estimate the input image quality and uses 92 if it cannot be determined, and PIL defaults to 75. Default: 0 (disabled)
- **max_filesize**: The maximum size of a target piece of cover art in bytes. When using an ImageMagick backend this sets `-define jpeg:extent=max_filesize`. Using PIL this will reduce JPG quality by up to 50% to attempt to reach the target filesize. Neither method is *guaranteed* to reach the target size, however in most cases it should succeed. Default: 0 (disabled)
- **enforce_ratio**: Only images with a width:height ratio of 1:1 are considered as valid album art candidates if set to `yes`. It is also possible to specify a certain deviation to the exact ratio to still be considered valid. This can be done either in pixels (`enforce_ratio: 10px`) or as a percentage of the longer edge (`enforce_ratio: 0.5%`). Default: `no`.
- **sources**: List of sources to search for images. An asterisk * expands to all available sources. Default: `filesystem coverart itunes amazon albumart`, i.e., everything but `wikipedia`, `google`, `fanarttv` and `lastfm`. Enable those sources for more matches at the cost of some speed. They are searched in the given order, thus in the default config, no remote (Web) art source are queried if local art is found in the filesystem. To use a local image as fallback, move it to the end of the list. For even more fine-grained control over the search order, see the section on *Album Art Sources* below.
- **google_key**: Your Google API key (to enable the Google Custom Search backend). Default: None.
- **google_engine**: The custom search engine to use. Default: The beets custom search engine, which searches the entire web.
- **fanarttv_key**: The personal API key for requesting art from fanart.tv. See below.
- **lastfm_key**: The personal API key for requesting art from Last.fm. See below.
- **store_source**: If enabled, fetchart stores the artwork's source in a flexible tag named `art_source`. See below for the rationale behind this. Default: `no`.
- **high_resolution**: If enabled, fetchart retrieves artwork in the highest resolution it can find (warning: image files can sometimes reach >20MB). Default: `no`.
- **deinterlace**: If enabled, Pillow or ImageMagick backends are instructed to store cover art as non-progressive JPEG. You might need this if you use DAPs that don't support progressive images. Default: `no`.
- **cover_format**: If enabled, forced the cover image into the specified format. Most often, this will be either `JPEG` or `PNG`[1]. Also respects `deinterlace`. Default: None (leave unchanged).

Note: `maxwidth` and `enforce_ratio` options require either ImageMagick or Pillow.

Note: Previously, there was a `remote_priority` option to specify when to look for art on the filesystem. This is still respected, but a deprecation message will be shown until you replace this configuration with the new `filesystem` value in the `sources` array.

Here's an example that makes plugin select only images that contain `front` or `back` keywords in their filenames and prioritizes the iTunes source over others:

```
fetchart:
    cautious: true
    cover_names: front back
    sources: itunes *
```

[1] Other image formats are available, though the full list depends on your system and what backend you are using. If you're using the ImageMagick backend, you can use `magick identify -list format` to get a full list of all supported formats, and you can use the Python function PIL.features.pilinfo() to print a list of all supported formats in Pillow (`python3 -c 'import PIL.features as f; f.pilinfo()'`).

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.

Display Only Missing Album Art

Use the `fetchart` command with the `-q` switch in order to display only missing art:

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

By default the command will display all albums matching the `query`. When the `-q` or `--quiet` switch is given, only albums for which artwork has been fetched, or for which artwork could not be found will be printed.

Image Resizing

Beets can resize images using Pillow, ImageMagick, or a server-side resizing proxy. If either Pillow 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 ImageMagick, 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 Pillow instead.

Album Art Sources

By default, this plugin searches for art in the local filesystem as well as on the Cover Art Archive, the iTunes Store, Amazon, and AlbumArt.org, in that order. You can reorder the sources or remove some to speed up the process using the `sources` configuration option.

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.

For some of the art sources, the backend service can match artwork by various criteria. If you want finer control over the search order in such cases, you can use this alternative syntax for the `sources` option:

```
fetchart:
    sources:
        - filesystem
        - coverart: release
        - itunes
        - coverart: releasegroup
        - '*'
```

where listing a source without matching criteria will default to trying all available strategies. Entries of the forms `coverart: release releasegroup` and `coverart: *` are also valid. Currently, only the `coverart`

source supports multiple criteria: namely, `release` and `releasegroup`, which refer to the respective MusicBrainz IDs.

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 custom search

To use the google image search backend you need to register for a Google API key. Set the `google_key` configuration option to your key, then add `google` to the list of sources in your configuration.

Optionally, you can define a custom search engine. Get your search engine's token and use it for your `google_engine` configuration option. The default engine searches the entire web for cover art.

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

Fanart.tv

Although not strictly necessary right now, you might think about registering a personal fanart.tv API key. Set the `fanarttv_key` configuration option to your key, then add `fanarttv` to the list of sources in your configuration.

More detailed information can be found on their blog. Specifically, the personal key will give you earlier access to new art.

Last.fm

To use the Last.fm backend, you need to register for a Last.fm API key. Set the `lastfm_key` configuration option to your API key, then add `lastfm` to the list of sources in your configutation.

Storing the Artwork's Source

Storing the current artwork's source might be used to narrow down `fetchart` commands. For example, if some albums have artwork placed manually in their directories that should not be replaced by a forced album art fetch, you could do

```
beet fetchart -f ^art_source:filesystem
```

The values written to `art_source` are the same names used in the `sources` configuration value.

FileFilter Plugin

The `filefilter` plugin allows you to skip files during import using regular expressions.

To use the `filefilter` plugin, enable it in your configuration (see *Using Plugins*).

Configuration

To configure the plugin, make a `filefilter:` section in your configuration file. The available options are:

- **path**: A regular expression to filter files based on their path and name. Default: `.*` (import everything)

- **album_path** and **singleton_path**: You may specify different regular expressions used for imports of albums and singletons. This way, you can automatically skip singletons when importing albums if the names (and paths) of the files are distinguishable via a regex. The regexes defined here take precedence over the global `path` option.

Here's an example:

```
filefilter:
    path: .*\d\d[^/]+$
          # will only import files which names start with two digits
    album_path: .*\d\d[^/]+$
    singleton_path: .*/(?!\d\d)[^/]+$
```

Fish Plugin

The `fish` plugin adds a `beet fish` command that creates a Fish shell tab-completion file named `beet.fish` in `~/.config/fish/completions`. This enables tab-completion of `beet` commands for the Fish shell.

Configuration

Enable the `fish` plugin (see *Using Plugins*) on a system running the Fish shell.

Usage

Type `beet fish` to generate the `beet.fish` completions file at: `~/.config/fish/completions/`. If you later install or disable plugins, run `beet fish` again to update the completions based on the enabled plugins.

For users not accustomed to tab completion... After you type `beet` followed by a space in your shell prompt and then the `TAB` key, you should see a list of the beets commands (and their abbreviated versions) that can be invoked in your current environment. Similarly, typing `beet -<TAB>` will show you all the option flags available to you, which also applies to subcommands such as `beet import -<TAB>`. If you type `beet ls` followed by a space and then the and the `TAB` key, you will see a list of all the album/track fields that can be used in beets queries. For example, typing `beet ls ge<TAB>` will complete to `genre:` and leave you ready to type the rest of your query.

Options

In addition to beets commands, plugin commands, and option flags, the generated completions also include by default all the album/track fields. If you only want the former and do not want the album/track fields included in the generated completions, use `beet fish -f` to only generate completions for beets/plugin commands and option flags.

If you want generated completions to also contain album/track field *values* for the items in your library, you can use the `-e` or `--extravalues` option. For example: `beet fish -e genre` or `beet fish -e genre -e albumartist` In the latter case, subsequently typing `beet list genre: <TAB>` will display a list of all the genres in your library and `beet list albumartist: <TAB>` will show a list of the album artists in your library. Keep in mind that all of these values will be put into the generated completions file, so use this option with care when specified fields contain a large number of values. Libraries with, for example, very large numbers of genres/artists may result in higher memory utilization, completion latency, et cetera. This option is not meant to replace database queries altogether.

Freedesktop Plugin

The `freedesktop` plugin created .directory files in your album folders. This plugin is now deprecated and replaced by the *Thumbnails Plugin* with the `dolphin` option enabled.

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 `fromfilename` plugin, enable it in your configuration (see *Using Plugins*).

FtInTitle Plugin

The `ftintitle` plugin automatically moves "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 `ftintitle` plugin, enable it in your configuration (see *Using Plugins*).

Configuration

To configure the plugin, make a `ftintitle:` section in your configuration file. The available options are:

- **auto**: Enable metadata rewriting during import. Default: `yes`.
- **drop**: Remove featured artists entirely instead of adding them to the title field. Default: `no`.
- **format**: Defines the format for the featuring X part of the new title field. In this format the `{0}` is used to define where the featured artists are placed. Default: `feat. {0}`

Running Manually

From the command line, type:

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

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

Use the `-d` flag to remove featured artists (equivalent of the `drop` config option).

Fuzzy Search Plugin

The `fuzzy` plugin provides a prefixed query that searches your 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 *Using Plugins*). You'll then be able to use the `~` prefix to use fuzzy matching:

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

Configuration

To configure the plugin, make a `fuzzy:` section in your configuration file. The available options are:

- **threshold**: The "sensitivity" of the fuzzy match. A value of 1.0 will show only perfect matches and a value of 0.0 will match everything. Default: 0.7.
- **prefix**: The character used to designate fuzzy queries. Default: ~, which may need to be escaped in some shells.

Gmusic Plugin

The `gmusic` plugin interfaced beets to Google Play Music. It has been removed after the shutdown of this service.

Hook Plugin

Internally, beets uses *events* to tell plugins when something happens. For example, one event fires when the importer finishes processes a song, and another triggers just before the `beet` command exits. The `hook` plugin lets you run commands in response to these events.

Configuration

To configure the plugin, make a `hook` section in your configuration file. The available options are:

- **hooks**: A list of events and the commands to run (see *Configuring Each Hook*). Default: Empty.

Configuring Each Hook

Each element under `hooks` should have these keys:

- **event**: The name of the event that will trigger this hook. See the *plugin events* documentation for a list of possible values.
- **command**: The command to run when this hook executes.

Command Substitution

Commands can access the parameters of events using Python string formatting. Use `{name}` in your command and the plugin will substitute it with the named value. The name can also refer to a field, as in `{album.path}`.

You can find a list of all available events and their arguments in the *plugin events* documentation.

Example Configuration

```
hook:
  hooks:
    # Output on exit:
    #   beets just exited!
    #   have a nice day!
    - event: cli_exit
      command: echo "beets just exited!"
    - event: cli_exit
```

(continues on next page)

(continued from previous page)

```
    command: echo "have a nice day!"

  # Output on item import:
  #   importing "<file_name_here>"
  # Where <file_name_here> is the item being imported
  - event: item_imported
    command: echo "importing \"{item.path}\""

  # Output on write:
  #   writing to "<file_name_here>"
  # Where <file_name_here> is the file being written to
  - event: write
    command: echo "writing to {path}"
```

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 `ihate` plugin, enable it in your configuration (see *Using Plugins*).

Configuration

To configure the plugin, make an `ihate:` section in your configuration file. The available options are:

- **skip**: Never import items and albums that match a query in this list. Default: `[]` (empty list).
- **warn**: Print a warning message for matches in this list of queries. Default: `[]`.

Here's an example:

```
ihate:
    warn:
        - artist:rnb
        - genre:soul
        # Only warn about tribute albums in rock genre.
        - genre:rock album:tribute
    skip:
        - genre::russian\srock
        - genre:polka
        - artist:manowar
        - album:christmas
```

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

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.

To use the `importadded` plugin, enable it in your configuration (see *Using Plugins*).

Usage

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` are set to the oldest mtime of the files in the album before they were imported. The mtime of album directories is ignored.

This plugin can optionally be configured to also preserve mtimes at import using the `preserve_mtimes` option.

When `preserve_write_mtimes` option is set, this plugin preserves mtimes after each write to files using the `item.added` attribute.

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.

Configuration

To configure the plugin, make an `importadded:` section in your configuration file. There are two options available:

- **preserve_mtimes**: After importing files, re-set their mtimes to their original value. Default: `no`.
- **preserve_write_mtimes**: After writing files, re-set their mtimes to their original value. Default: `no`.

Reimport

This plugin will skip reimported singleton items and reimported albums and all of their items.

ImportFeeds Plugin

This plugin helps you keep track of newly imported music in your library.

To use the `importfeeds` plugin, enable it in your configuration (see *Using Plugins*).

Configuration

To configure the plugin, make an `importfeeds:` section in your configuration file. The available options are:

- **absolute_path**: Use absolute paths instead of relative paths. Some applications may need this to work properly. Default: `no`.
- **dir**: The output directory. Default: Your beets library directory.
- **formats**: Select the kind of output. Use one or more of:
  - **m3u**: Catalog the imports in a centralized playlist.

  - **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.
  - **echo**: Do not write a playlist file at all, but echo a list of new file paths to the terminal.

  Default: None.

- **m3u_name**: Playlist name used by the `m3u` format. Default: `imported.m3u`.
- **relative_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. Default: None.

Here's an example configuration for this plugin:

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

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 `info` plugin in your configuration (see *Using Plugins*) 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.

You can also enter a *query* to inspect music from your library:

```
$ beet info beatles
```

If you just want to see specific properties you can use the `--include-keys` option to filter them. The argument is a comma-separated list of field names. For example:

```
$ beet info -i 'title,mb_artistid' beatles
```

Will only show the `title` and `mb_artistid` properties. You can add the `-i` option multiple times to the command line.

Additional command-line options include:

- `--library` or `-l`: Show data from the library database instead of the files' tags.
- `--album` or `-a`: Show data from albums instead of tracks (implies `--library`).
- `--summarize` or `-s`: Merge all the information from multiple files into a single list of values. If the tags differ across the files, print `[various]`.
- `--format` or `-f`: Specify a specific format with which to print every item. This uses the same template syntax as beets' *path formats*.
- `--keys-only` or `-k`: Show the name of the tags without the values.

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 the `inline` plugin, enable it in your configuration (see *Using Plugins*). 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)
```

IPFS Plugin

The `ipfs` plugin makes it easy to share your library and music with friends. The plugin uses ipfs for storing the library and file content.

Installation

This plugin requires go-ipfs to be running as a daemon and that the associated `ipfs` command is on the user's `$PATH`.

Once you have the client installed, enable the `ipfs` plugin in your configuration (see *Using Plugins*).

Usage

This plugin can store and retrieve music individually, or it can share entire library databases.

Adding

To add albums to ipfs, making them shareable, use the `-a` or `--add` flag. If used without arguments it will add all albums in the local library. When added, all items and albums will get an "ipfs" field in the database containing the hash of that specific file/folder. Newly imported albums will be added automatically to ipfs by default (see below).

Retrieving

You can give the ipfs hash for some music to a friend. They can get that album from ipfs, and import it into beets, using the `-g` or `--get` flag. If the argument passed to the `-g` flag isn't an ipfs hash, it will be used as a query instead, getting all albums matching the query.

Sharing Libraries

Using the `-p` or `--publish` flag, a copy of the local library will be published to ipfs. Only albums/items with ipfs records in the database will published, and local paths will be stripped from the library. A hash of the library will be returned to the user.

A friend can then import this remote library by using the `-i` or `--import` flag. To tag an imported library with a specific name by passing a name as the second argument to `-i,` after the hash. The content of all remote libraries will be combined into an additional library as long as the content doesn't already exist in the joined library.

When remote libraries has been imported you can search them by using the `-l` or `--list` flag. The hash of albums matching the query will be returned, and this can then be used with `-g` to fetch and import the album to the local library.

Ipfs can be mounted as a FUSE file system. This means that music in a remote library can be streamed directly, without importing them to the local library first. If the `/ipfs` folder is mounted then matching queries will be sent to the *Play Plugin* using the `-m` or `--play` flag.

Configuration

The ipfs plugin will automatically add imported albums to ipfs and add those hashes to the database. This can be turned off by setting the `auto` option in the `ipfs:` section of the config to `no`.

If the setting `nocopy` is true (defaults false) then the plugin will pass the `--nocopy` option when adding things to ipfs. If the filestore option of ipfs is enabled this will mean files are neither removed from beets nor copied somewhere else.

Key Finder Plugin

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

To use the `keyfinder` plugin, enable it in your configuration (see *Using Plugins*).

Configuration

To configure the plugin, make a `keyfinder:` section in your configuration file. The available options are:

- **auto**: Analyze every file on import. Otherwise, you need to use the `beet keyfinder` command explicitly. Default: `yes`
- **bin**: The name of the program use for key analysis. You can use either KeyFinder or keyfinder-cli. If you installed the KeyFinder GUI on a Mac, for example, you want something like `/Applications/KeyFinder.app/Contents/MacOS/KeyFinder`. If using keyfinder-cli, the binary must be named `keyfinder-cli`. Default: `KeyFinder` (i.e., search for the program in your `$PATH`)..
- **overwrite**: Calculate a key even for files that already have an *initial_key* value. Default: `no`.

KodiUpdate Plugin

The `kodiupdate` plugin lets you automatically update Kodi's music library whenever you change your beets library.

To use `kodiupdate` plugin, enable it in your configuration (see *Using Plugins*). Then, you'll want to configure the specifics of your Kodi host. You can do that using a `kodi:` section in your `config.yaml`, which looks like this:

```
kodi:
    host: localhost
    port: 8080
    user: kodi
    pwd: kodi
```

To use the `kodiupdate` plugin you need to install the requests library with:

```
pip install requests
```

You'll also need to enable JSON-RPC in Kodi in order the use the plugin. In Kodi's interface, navigate to System/Settings/Network/Services and choose "Allow control of Kodi via HTTP."

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

Configuration

The available options under the `kodi:` section are:

- **host**: The Kodi host name. Default: `localhost`

- **port**: The Kodi host port. Default: 8080
- **user**: The Kodi host user. Default: `kodi`
- **pwd**: The Kodi host password. Default: `kodi`

LastGenre Plugin

The `lastgenre` plugin fetches *tags* from Last.fm and assigns them as genres to your albums and items.

Installation

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

```
pip install pylast
```

After you have pylast installed, enable the `lastgenre` plugin in your configuration (see *Using Plugins*).

Usage

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 or forgo a 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.

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.

The most common way to use this would be with a custom whitelist containing only a desired subset of genres. Consider for a example this minimal whitelist:

```
rock
heavy metal
pop
```

together with the default genre tree. Then an item that has its genre specified as *viking metal* would actually be tagged as *heavy metal* because neither *viking metal* nor its parent *black metal* are in the whitelist. It always tries to use the most specific genre that's available in the whitelist.

The relevant subtree path in the default tree looks like this:

```
- rock:
    - heavy metal:
        - black metal:
            - viking metal
```

Considering that, it's not very useful to use the default whitelist (which contains about any genre contained in the tree) with canonicalization because nothing would ever be matched to a more generic node since all the specific subgenres are in the whitelist to begin with.

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. 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.

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.

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.

Specific vs. Popular Genres

By default, the plugin sorts genres by popularity. However, you can use the `prefer_specific` option to override this behavior and instead sort genres by specificity, as determined by your whitelist and canonicalization tree.

For instance, say you have both `folk` and `americana` in your whitelist and canonicalization tree and `americana` is a leaf within `folk`. If Last.fm returns both of those tags, lastgenre is going to use the most popular, which is often the most generic (in this case `folk`). By setting `prefer_specific` to true, lastgenre would use `americana` instead.

Configuration

To configure the plugin, make a `lastgenre:` section in your configuration file. The available options are:

- **auto**: Fetch genres automatically during import. Default: `yes`.
- **canonical**: Use a canonicalization tree. Setting this to `yes` will use a built-in tree. You can also set it to a path, like the `whitelist` config value, to use your own tree. Default: `no` (disabled).
- **count**: Number of genres to fetch. Default: 1
- **fallback**: A string if to use a fallback genre when no genre is found. You can use the empty string `''` to reset the genre. Default: None.
- **force**: By default, beets will always fetch new genres, even if the files already have one. To instead leave genres in place in when they pass the whitelist, set the `force` option to `no`. Default: `yes`.
- **min_weight**: Minimum popularity factor below which genres are discarded. Default: 10.
- **prefer_specific**: Sort genres by the most to least specific, rather than most to least popular. Default: `no`.
- **source**: Which entity to look up in Last.fm. Can be either `artist`, `album` or `track`. Default: `album`.

- **separator**: A separator for multiple genres. Default: `', '`.
- **whitelist**: The filename of a custom genre list, `yes` to use the internal whitelist, or `no` to consider all genres valid. Default: `yes`.
- **title_case**: Convert the new tags to TitleCase before saving. Default: `yes`.

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 or items matching a certain query.

By default, `beet lastgenre` matches albums. To match individual tracks or singletons, use the `-A` switch: `beet lastgenre -A [QUERY]`.

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

LastImport Plugin

The `lastimport` plugin downloads play-count data from your Last.fm library into beets' database. You can later create *smart playlists* by querying `play_count` and do other fun stuff with this field.

Installation

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

```
pip install pylast
```

After you have pylast installed, enable the `lastimport` plugin in your configuration (see *Using Plugins*).

Next, add your Last.fm username to your beets configuration file:

```
lastfm:
    user: beetsfanatic
```

Importing Play Counts

Simply run `beet lastimport` and wait for the plugin to request tracks from Last.fm and match them to your beets library. (You will be notified of tracks in your Last.fm profile that do not match any songs in your library.)

Then, your matched tracks will be populated with the `play_count` field, which you can use in any query or template. For example:

```
$ beet ls -f '$title: $play_count' play_count:5..
Eple (Melody A.M.): 60
```

To see more information (namely, the specific play counts for matched tracks), use the `-v` option.

Configuration

Aside from the required `lastfm.user` field, this plugin has some specific options under the `lastimport:` section:

- **per_page**: The number of tracks to request from the API at once. Default: 500.
- **retry_limit**: How many times should we re-send requests to Last.fm on failure? Default: 3.

By default, the plugin will use beets's own Last.fm API key. You can also override it with your own API key:

```
lastfm:
    api_key: your_api_key
```

Load Extension Plugin

Beets uses an SQLite database to store and query library information, which has support for extensions to extend its functionality. The `loadext` plugin lets you enable these SQLite extensions within beets.

One of the primary uses of this within beets is with the "ICU" extension, which adds support for case insensitive querying of non-ASCII characters.

Configuration

To configure the plugin, make a `loadext` section in your configuration file. The section must consist of a list of paths to extensions to load, which looks like this:

```
loadext:
  - libicu
```

If a relative path is specified, it is resolved relative to the beets configuration directory.

If no file extension is specified, the default dynamic library extension for the current platform will be used.

Building the ICU extension

This section is for **advanced** users only, and is not an in-depth guide on building the extension.

To compile the ICU extension, you will need a few dependencies:

- gcc
- icu-devtools
- libicu
- libicu-dev
- libsqlite3-dev

Here's roughly how to download, build and install the extension (although the specifics may vary from system to system):

```
$ wget https://sqlite.org/2019/sqlite-src-3280000.zip
$ unzip sqlite-src-3280000.zip
$ cd sqlite-src-3280000/ext/icu
$ gcc -shared -fPIC icu.c `icu-config --ldflags` -o libicu.so
$ cp libicu.so ~/.config/beets
```

Lyrics Plugin

The `lyrics` plugin fetches and stores song lyrics from databases on the Web. Namely, the current version of the plugin uses Musixmatch, Genius.com, Tekstowo.pl, and, optionally, the Google custom search API.

Fetch Lyrics During Import

To automatically fetch lyrics for songs you import, enable the `lyrics` plugin in your configuration (see *Using Plugins*). Then, install the requests library by typing:

```
pip install requests
```

The plugin uses requests to download lyrics.

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.

Configuration

To configure the plugin, make a `lyrics:` section in your configuration file. The available options are:

- **auto**: Fetch lyrics automatically during import. Default: `yes`.
- **bing_client_secret**: Your Bing Translation application password (to *Activate On-the-Fly Translation*)
- **bing_lang_from**: By default all lyrics with a language other than `bing_lang_to` are translated. Use a list of lang codes to restrict the set of source languages to translate. Default: `[]`
- **bing_lang_to**: Language to translate lyrics into. Default: None.
- **fallback**: By default, the file will be left unchanged when no lyrics are found. Use the empty string `''` to reset the lyrics in such a case. Default: None.
- **force**: By default, beets won't fetch lyrics if the files already have ones. To instead always fetch lyrics, set the `force` option to `yes`. Default: `no`.
- **google_API_key**: Your Google API key (to enable the Google Custom Search backend). Default: None.
- **google_engine_ID**: The custom search engine to use. Default: The beets custom search engine, which gathers an updated list of sources known to be scrapeable.
- **sources**: List of sources to search for lyrics. An asterisk `*` expands to all available sources. Default: `google musixmatch genius tekstowo`, i.e., all the available sources. The `google` source will be automatically deactivated if no `google_API_key` is setup. The `google`, `genius`, and `tekstowo` sources will only be enabled if BeautifulSoup is installed.

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

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

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 Mag-

netic 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. Inversely, the `-l` option restricts operations to lyrics that are locally available, which show lyrics faster without using the network at all.

Rendering Lyrics into Other Formats

The `-r directory` option renders all lyrics as reStructuredText (ReST) documents in `directory` (by default, the current directory). That directory, in turn, can be parsed by tools like Sphinx to generate HTML, ePUB, or PDF documents.

A minimal `conf.py` and `index.rst` files are created the first time the command is run. They are not overwritten on subsequent runs, so you can safely modify these files to customize the output.

Sphinx supports various builders, but here are a few suggestions.

- Build an HTML version:

```
sphinx-build -b html . _build/html
```

- Build an ePUB3 formatted file, usable on ebook readers:

```
sphinx-build -b epub3 . _build/epub
```

- Build a PDF file, which incidentally also builds a LaTeX file:

```
sphinx-build -b latex %s _build/latex && make -C _build/latex all-pdf
```

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. Then add `google` to the list of sources in your configuration (or use default list, which includes it as long as you have an API key). If you use default `google_engine_ID`, we recommend limiting the sources to `musixmatch google` as the other sources are already included in the Google results.

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.

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

Activate Genius and Tekstowo.pl Lyrics

Using the Genius or Tekstowo.pl backends requires BeautifulSoup, which you can install using pip by typing:

```
pip install beautifulsoup4
```

These backends are enabled by default.

Activate On-the-Fly Translation

Using the Bing Translation API requires langdetect, which you can install using pip by typing:

```
pip install langdetect
```

You also need to register for a Microsoft Azure Marketplace free account and to the Microsoft Translator API. Follow the four steps process, specifically at step 3 enter `beets` as *Client ID* and copy/paste the generated *Client secret* into your `bing_client_secret` configuration, alongside `bing_lang_to` target *language code*.

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 in your configuration (see *Using 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.

The command has one command-line option:

- To remove albums from the collection which are no longer present in the beets database, use the `-r` (`--remove`) flag.

Configuration

To configure the plugin, make a `mbcollection:` section in your configuration file. There is one option available:

- **auto**: Automatically amend your MusicBrainz collection whenever you import a new album. Default: `no`.
- **collection**: The MBID of which MusicBrainz collection to update. Default: `None`.
- **remove**: Remove albums from collections which are no longer present in the beets database. Default: `no`.

MusicBrainz Submit Plugin

The `mbsubmit` plugin provides an extra prompt choice during an import session that prints the tracks of the current album in a format that is parseable by MusicBrainz's track parser.

Usage

Enable the `mbsubmit` plugin in your configuration (see *Using Plugins*) and select the `Print tracks` choice which is by default displayed when no strong recommendations are found for the album:

```
No matching release found for 3 tracks.
For help, see: https://beets.readthedocs.org/en/latest/faq.html#nomatch
[U]se as-is, as Tracks, Group albums, Skip, Enter search, enter Id, aBort,
Print tracks? p
01. An Obscure Track - An Obscure Artist (3:37)
02. Another Obscure Track - An Obscure Artist (2:05)
03. The Third Track - Another Obscure Artist (3:02)

No matching release found for 3 tracks.
For help, see: https://beets.readthedocs.org/en/latest/faq.html#nomatch
[U]se as-is, as Tracks, Group albums, Skip, Enter search, enter Id, aBort,
Print tracks?
```

As MusicBrainz currently does not support submitting albums programmatically, the recommended workflow is to copy the output of the `Print tracks` choice and paste it into the parser that can be found by clicking on the "Track Parser" button on MusicBrainz "Tracklist" tab.

Configuration

To configure the plugin, make a `mbsubmit:` section in your configuration file. The following options are available:

- **format**: The format used for printing the tracks, defined using the same template syntax as beets' *path formats*. Default: `$track. $title - $artist ($length)`.
- **threshold**: The minimum strength of the autotagger recommendation that will cause the `Print tracks` choice to be displayed on the prompt. Default: `medium` (causing the choice to be displayed for all albums that have a recommendation of medium strength or lower). Valid values: `none`, `low`, `medium`, `strong`.

Please note that some values of the `threshold` configuration option might require other `beets` command line switches to be enabled in order to work as intended. In particular, setting a threshold of `strong` will only display the prompt if `timid` mode is enabled. You can find more information about how the recommendation system works at *Autotagger Matching Options*.

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 `mbsync` plugin in your configuration (see *Using Plugins*) 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.
- To customize the output of unrecognized items, use the `-f` (`--format`) option. The default output is `format_item` or `format_album` for items and albums, respectively.

MetaSync Plugin

This plugin provides the `metasync` command, which lets you fetch certain metadata from other sources: for example, your favorite audio player.

Currently, the plugin supports synchronizing with the Amarok music player, and with iTunes. It can fetch the rating, score, first-played date, last-played date, play count, and track uid from Amarok.

Installation

Enable the `metasync` plugin in your configuration (see *Using Plugins*).

To synchronize with Amarok, you'll need the dbus-python library. There are packages for most major Linux distributions.

Configuration

To configure the plugin, make a `metasync:` section in your configuration file. The available options are:

- **source**: A list of comma-separated sources to fetch metadata from. Set this to "amarok" or "itunes" to enable synchronization with that player. Default: empty

The follow subsections describe additional configure required for some players.

itunes

The path to your iTunes library **xml** file has to be configured, e.g.:

```
metasync:
    source: itunes
    itunes:
        library: ~/Music/iTunes Library.xml
```

Please note the indentation.

Usage

Run `beet metasync QUERY` to fetch metadata from the configured list of sources.

The command has a few command-line options:

- To preview the changes that would be made without applying them, use the `-p` (`--pretend`) flag.

- To specify temporary sources to fetch metadata from, use the `-s` (`--source`) flag with a comma-separated list of a sources.

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.

Usage

Add the `missing` plugin to your configuration (see *Using Plugins*). By default, the `beet missing` command lists the names of tracks that your library is missing from each album. It can also list the names of albums that your library is missing from each artist. 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, using command-line switches:

```
-f FORMAT, --format=FORMAT
                      print with custom FORMAT
-c, --count           count missing tracks per album
-t, --total           count total of missing tracks or albums
-a, --album           show missing albums for artist instead of tracks
```

. . . or by editing corresponding options.

Note that `-c` is ignored when used with `-a`.

Configuration

To configure the plugin, make a `missing:` section in your configuration file. The available options are:

- **count**: Print a count of missing tracks per album, with `format` defaulting to `$albumartist - $album: $missing`. Default: `no`.
- **format**: 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. Default: *format_item*.
- **total**: Print a single count of missing tracks in all albums. Default: `no`.

Here's an example

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

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 albums in your collection:

```
beet missing -a
```

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'
```

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`.

To gather these statistics it runs as an MPD client and watches the current state of MPD. This means that `mpdstats` needs to be running continuously for it to work.

Installing Dependencies

This plugin requires the python-mpd2 library in order to talk to the MPD server.

Install the library from pip, like so:

```
$ pip install python-mpd2
```

Add the `mpdstats` plugin to your configuration (see *Using Plugins*).

Usage

Use the `mpdstats` command to fire it up:

```
$ beet mpdstats
```

Configuration

To configure the plugin, make an `mpd:` section in your configuration file. The available options are:

- **host**: The MPD server hostname. Default: The `$MPD_HOST` environment variable if set, falling back to `localhost` otherwise.
- **port**: The MPD server port. Default: The `$MPD_PORT` environment variable if set, falling back to 6600 otherwise.
- **password**: The MPD server password. Default: None.
- **music_directory**: If your MPD library is at a different location from the beets library (e.g., because one is mounted on a NFS share), specify the path here.
- **strip_path**: If your MPD library contains local path, specify the part to remove here. Combining this with **music_directory** you can mangle MPD path to match the beets library one. Default: The beets library directory.
- **rating**: Enable rating updates. Default: `yes`.
- **rating_mix**: Tune the way rating is calculated (see below). Default: 0.75.

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.

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 `mpdupdate` plugin, enable it in your configuration (see *Using Plugins*). 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.)

Configuration

The available options under the `mpd:` section are:

- **host**: The MPD server name. Default: The `$MPD_HOST` environment variable if set, falling back to `localhost` otherwise.
- **port**: The MPD server port. Default: The `$MPD_PORT` environment variable if set, falling back to 6600 otherwise.
- **password**: The MPD server password. Default: None.

ParentWork Plugin

The `parentwork` plugin fetches the work title, parent work title and parent work composer from MusicBrainz.

In the MusicBrainz database, a recording can be associated with a work. A work can itself be associated with another work, for example one being part of the other (what we call the *direct parent*). This plugin looks the work id from the library and then looks up the direct parent, then the direct parent of the direct parent and so on until it reaches the top. The work at the top is what we call the *parent work*.

This plugin is especially designed for classical music. For classical music, just fetching the work title as in MusicBrainz is not satisfying, because MusicBrainz has separate works for, for example, all the movements of a symphony. This plugin aims to solve this problem by also fetching the parent work, which would be the whole symphony in this example.

The plugin can detect changes in `mb_workid` so it knows when to re-fetch other metadata, such as `parentwork`. To do this, when it runs, it stores a copy of `mb_workid` in the bookkeeping field `parentwork_workid_current`. At any later run of `beet parentwork` it will check if the tags `mb_workid` and `parentwork_workid_current` are still identical. If it is not the case, it means the work has changed and all the tags need to be fetched again.

This plugin adds seven tags:

- **parentwork**: The title of the parent work.
- **mb_parentworkid**: The MusicBrainz id of the parent work.
- **parentwork_disambig**: The disambiguation of the parent work title.
- **parent_composer**: The composer of the parent work.
- **parent_composer_sort**: The sort name of the parent work composer.
- **work_date**: The composition date of the work, or the first parent work that has a composition date. Format: yyyy-mm-dd.
- **parentwork_workid_current**: The MusicBrainz id of the work as it was when the parentwork was retrieved. This tag exists only for internal bookkeeping, to keep track of recordings whose works have changed.
- **parentwork_date**: The composition date of the parent work.

To use the `parentwork` plugin, enable it in your configuration (see *Using Plugins*).

Configuration

To configure the plugin, make a `parentwork:` section in your configuration file. The available options are:

- **force**: As a default, `parentwork` only fetches work info for recordings that do not already have a `parentwork` tag or where `mb_workid` differs from `parentwork_workid_current`. If `force` is enabled, it fetches it for all recordings. Default: `no`
- **auto**: If enabled, automatically fetches works at import. It takes quite some time, because beets is restricted to one MusicBrainz query per second. Default: `no`

Permissions Plugin

The `permissions` plugin allows you to set file permissions for imported music files and its directories.

To use the `permissions` plugin, enable it in your configuration (see *Using Plugins*). Permissions will be adjusted automatically on import.

Configuration

To configure the plugin, make an `permissions:` section in your configuration file. The `file` config value therein uses **octal modes** to specify the desired permissions. The default flags for files are octal 644 and 755 for directories.

Here's an example:

```
permissions:
    file: 644
    dir: 755
```

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 or paths on the command line.

Command Line Usage

To use the `play` 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
```

While playing you'll be able to interact with the player if it is a command-line oriented, and you'll get its output in real time.

Interactive Usage

The `play` plugin can also be invoked during an import. If enabled, the plugin adds a `plaY` option to the prompt, so pressing `y` will execute the configured command and play the items currently being imported.

Once the configured command exits, you will be returned to the import decision prompt. If your player is configured to run in the background (in a client/server setup), the music will play until you choose to stop it, and the import operation continues immediately.

Configuration

To configure the plugin, make a `play:` section in your configuration file. The available options are:

- **command**: The command used to open the playlist. Default: `open` on OS X, `xdg-open` on other Unixes and `start` on Windows. Insert `$args` to use the `--args` feature.
- **relative_to**: If set, emit paths relative to this directory. Default: None.
- **use_folders**: When using the `-a` option, the m3u will contain the paths to each track on the matched albums. Enable this option to store paths to folders instead. Default: `no`.
- **raw**: Instead of creating a temporary m3u playlist and then opening it, simply call the command with the paths returned by the query as arguments. Default: `no`.
- **warning_threshold**: Set the minimum number of files to play which will trigger a warning to be emitted. If set to `no`, warning are never issued. Default: 100.
- **bom**: Set whether or not a UTF-8 Byte Order Mark should be emitted into the m3u file. If you're using foobar2000 or Winamp, this is needed. Default: `no`.

Optional Arguments

The `--args` (or `-A`) flag to the `play` command lets you specify additional arguments for your player command. Options are inserted after the configured `command` string and before the playlist filename.

For example, if you have the plugin configured like this:

```
play:
    command: mplayer -quiet
```

and you occasionally want to shuffle the songs you play, you can type:

```
$ beet play --args -shuffle
```

to get beets to execute this command:

```
mplayer -quiet -shuffle /path/to/playlist.m3u
```

instead of the default.

If you need to insert arguments somewhere other than the end of the `command` string, use `$args` to indicate where to insert them. For example:

```
play:
    command: mpv $args --playlist
```

indicates that you need to insert extra arguments before specifying the playlist.

The `--yes` (or `-y`) flag to the `play` command will skip the warning message if you choose to play more items than the **warning_threshold** value usually allows.

Note on the Leakage of the Generated Playlists

Because the command that will open the generated `.m3u` files can be arbitrarily configured by the user, beets won't try to delete those files. For this reason, using this plugin will leave one or several playlist(s) in the directory selected to create temporary files (Most likely `/tmp/` on Unix-like systems. See tempfile.tempdir in the Python docs.). Leaking those playlists until they are externally wiped could be an issue for privacy or storage reasons. If this is the case for you, you might want to use the `raw` config option described above.

Playlist Plugin

`playlist` is a plugin to use playlists in m3u format.

To use it, enable the `playlist` plugin in your configuration (see *Using Plugins*). Then configure your playlists like this:

```
playlist:
    auto: no
    relative_to: ~/Music
    playlist_dir: ~/.mpd/playlists
    forward_slash: no
```

It is possible to query the library based on a playlist by specifying its absolute path:

```
$ beet ls playlist:/path/to/someplaylist.m3u
```

The plugin also supports referencing playlists by name. The playlist is then searched in the playlist_dir and the ".m3u" extension is appended to the name:

```
$ beet ls playlist:anotherplaylist
```

The plugin can also update playlists in the playlist directory automatically every time an item is moved or deleted. This can be controlled by the `auto` configuration option.

Configuration

To configure the plugin, make a `playlist:` section in your configuration file. In addition to the `playlists` described above, the other configuration options are:

- **auto**: If this is set to `yes`, then anytime an item in the library is moved or removed, the plugin will update all playlists in the `playlist_dir` directory that contain that item to reflect the change. Default: `no`
- **playlist_dir**: Where to read playlist files from. Default: The current working directory (i.e., `'.'`).
- **relative_to**: Interpret paths in the playlist files relative to a base directory. Instead of setting it to a fixed path, it is also possible to set it to `playlist` to use the playlist's parent directory or to `library` to use the library directory. Default: `library`
- **forward_slash**: Forces forward slashes in the generated playlist files. If you intend to use this plugin to generate playlists for MPD on Windows, set this to yes. Default: Use system separator.

PlexUpdate Plugin

`plexupdate` is a very simple plugin for beets that lets you automatically update Plex's music library whenever you change your beets library.

To use `plexupdate` plugin, enable it in your configuration (see *Using Plugins*). Then, you'll probably want to configure the specifics of your Plex server. You can do that using an `plex:` section in your `config.yaml`, which looks like this:

```
plex:
    host: localhost
    port: 32400
    token: token
```

The `token` key is optional: you'll need to use it when in a Plex Home (see Plex's own documentation about tokens).

To use the `plexupdate` plugin you need to install the requests library with:

```
pip install requests
```

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

Configuration

The available options under the `plex:` section are:

- **host**: The Plex server name. Default: `localhost`.
- **port**: The Plex server port. Default: 32400.
- **token**: The Plex Home token. Default: Empty.
- **library_name**: The name of the Plex library to update. Default: `Music`
- **secure**: Use secure connections to the Plex server. Default: `False`
- **ignore_cert_errors**: Ignore TLS certificate errors when using secure connections. Default: `False`

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 *Using 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`.

As an alternative, you can use `-t MINUTES` to choose a set of music with a given play time. To select tracks that total one hour, for example, type `beet random -t60`.

ReplayGain Plugin

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

Installation

This plugin can use one of many backends to compute the ReplayGain values: GStreamer, mp3gain (and its cousin, aacgain), Python Audio Tools or ffmpeg. ffmpeg and mp3gain can be easier to install. mp3gain supports less audio formats than the other backend.

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).

To speed up analysis with some of the avalaible backends, this plugin processes tracks or albums (when using the `-a` option) in parallel. By default, a single thread is used per logical core of your CPU.

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

```
replaygain:
    backend: gstreamer
```

The GStreamer backend does not support parallel analysis.

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
```

Python Audio Tools

This backend uses the Python Audio Tools package to compute ReplayGain for a range of different file formats. The package is not available via PyPI; it must be installed manually (only versions preceding 3.x are compatible).

On OS X, most of the dependencies can be installed with Homebrew:

```
brew install mpg123 mp3gain vorbisgain faad2 libvorbis
```

The Python Audio Tools backend does not support parallel analysis.

ffmpeg

This backend uses ffmpeg to calculate EBU R128 gain values. To use it, install the ffmpeg command-line tool and select the `ffmpeg` backend in your config file.

Configuration

To configure the plugin, make a `replaygain:` section in your configuration file. The available options are:

- **auto**: Enable ReplayGain analysis during import. Default: `yes`.
- **threads**: The number of parallel threads to run the analysis in. Overridden by `--threads` at the command line. Default: # of logical CPU cores
- **parallel_on_import**: Whether to enable parallel analysis during import. As of now this ReplayGain data is not written to files properly, so this option is disabled by default. If you wish to enable it, remember to run `beet write` after importing to actually write to the imported files. Default: `no`
- **backend**: The analysis backend; either `gstreamer`, `command`, `audiotools` or `ffmpeg`. Default: `command`.
- **overwrite**: Re-analyze files that already have ReplayGain tags. Default: `no`.

- **targetlevel**: A number of decibels for the target loudness level for files using `REPLAYGAIN_` tags. Default: `89`.
- **r128_targetlevel**: The target loudness level in decibels (i.e. `<loudness in LUFS> + 107`) for files using `R128_` tags. Default: 84 (Use `83` for ATSC A/85, `84` for EBU R128 or `89` for ReplayGain 2.0.)
- **r128**: A space separated list of formats that will use `R128_` tags with integer values instead of the common `REPLAYGAIN_` tags with floating point values. Requires the "ffmpeg" backend. Default: `Opus`.
- **per_disc**: Calculate album ReplayGain on disc level instead of album level. Default: `no`

These options only work with the "command" backend:

- **command**: The path to the `mp3gain` or `aacgain` executable (if beets cannot find it by itself). For example: `/Applications/MacMP3Gain.app/Contents/Resources/aacgain`. Default: Search in your `$PATH`.
- **noclip**: Reduce the amount of ReplayGain adjustment to whatever amount would keep clipping from occurring. Default: `yes`.

This option only works with the "ffmpeg" backend:

- **peak**: Either `true` (the default) or `sample`. `true` is more accurate but slower.

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 [-Waf] [QUERY]
```

The `-a` flag analyzes whole albums instead of individual tracks. Provide a query (see *Queries*) to indicate which items or albums to analyze. Files that already have ReplayGain values are skipped unless `-f` is supplied. Use `-w` (write tags) or `-W` (don't write tags) to control whether ReplayGain tags are written into the music files, or stored in the beets database only (the default is to use *the importer's configuration*).

To execute with a different number of threads, call `beet replaygain --threads N`:

```
$ beet replaygain --threads N [-Waf] [QUERY]
```

with N any integer. To disable parallelism, use `--threads 0`.

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
```

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 `rewrite` plugin (see *Using Plugins*). 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.

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 `scrub` plugin (see *Using 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.

Configuration

To configure the plugin, make a `scrub:` section in your configuration file. There is one option:

- **auto**: Enable metadata stripping during import. Default: `yes`.

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 `smartplaylist` plugin in your configuration (see *Using Plugins*). Then configure your smart playlists like the following example:

```
smartplaylist:
    relative_to: ~/Music
    playlist_dir: ~/.mpd/playlists
    forward_slash: no
    playlists:
        - name: all.m3u
          query: ''

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

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"']
```

Note that since beets query syntax is in effect, you can also use sorting directives:

```
- name: 'Chronological Beatles'
  query: 'artist:Beatles year+'
- name: 'Mixed Rock'
  query: ['artist:Beatles year+', 'artist:"Led Zeppelin" bitrate+']
```

The former case behaves as expected, however please note that in the latter the sorts will be merged: `year+ bitrate+` will apply to both the Beatles and Led Zeppelin. If that bothers you, please get in touch.

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, each playlist is automatically regenerated at the end of the session if an item or album it matches changed in the library database. To force regeneration, you can invoke it manually from the command line:

```
$ beet splupdate
```

This will regenerate all smart playlists. You can also specify which ones you want to regenerate:

```
$ beet splupdate BeatlesUniverse.m3u MyTravelPlaylist
```

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.

Configuration

To configure the plugin, make a `smartplaylist:` section in your configuration file. In addition to the `playlists` described above, the other configuration options are:

- **auto**: Regenerate the playlist after every database change. Default: `yes`.
- **playlist_dir**: Where to put the generated playlist files. Default: The current working directory (i.e., `'.'`).
- **relative_to**: Generate paths in the playlist files relative to a base directory. If you intend to use this plugin to generate playlists for MPD, point this to your MPD music directory. Default: Use absolute paths.
- **forward_slash**: Forces forward slashes in the generated playlist files. If you intend to use this plugin to generate playlists for MPD on Windows, set this to yes. Default: Use system separator.
- **prefix**: Prepend this string to every path in the playlist file. For example, you could use the URL for a server where the music is stored. Default: empty string.
- **urlencoded**: URL-encode all paths. Default: `no`.

SonosUpdate Plugin

The `sonosupdate` plugin lets you automatically update Sonos's music library whenever you change your beets library.

To use `sonosupdate` plugin, enable it in your configuration (see *Using Plugins*).

To use the `sonosupdate` plugin you need to install the soco library with:

```
pip install soco
```

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

Spotify Plugin

The `spotify` plugin generates Spotify playlists from tracks in your library with the `beet spotify` command using the Spotify Search API.

Also, the plugin can use the Spotify Album and Track APIs to provide metadata matches for the importer.

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.
- You want to autotag music with metadata from the Spotify API.

Basic Usage

First, enable the `spotify` 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...
https://open.spotify.com/track/19w0OHr8SiZzRhjpnjctJ4
https://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.

You can enter the URL for an album or song on Spotify at the `enter Id` prompt during import:

```
Enter search, enter Id, aBort, eDit, edit Candidates, playY? i
Enter release ID: https://open.spotify.com/album/2rFYTHFBLQN3AYlrymBPPA
```

Configuration

This plugin can be configured like other metadata source plugins as described in *Using Metadata Source Plugins*. In addition, the following configuration options are provided.

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

- **mode**: One of the following:
  - `list`: 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.

  Default: `list`.
- **region_filter**: A two-character country abbreviation, to limit results to that market. Default: None.
- **show_failures**: List each lookup that does not return a Spotify ID (and therefore cannot be added to a playlist). Default: `no`.
- **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. The options are `popularity` and `first` (to just choose the first match returned). Default: `popularity`.
- **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. Default: None.

Here's an example:

```
spotify:
    source_weight: 0.7
    mode: open
    region_filter: US
    show_failures: on
    tiebreak: first

    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"
        }
    ]
```

Subsonic Playlist Plugin

The `subsonicplaylist` plugin allows to import playlists from a subsonic server. This is done by retrieving the track info from the subsonic server, searching for them in the beets library, and adding the playlist names to the *subsonic_playlist* tag of the found items. The content of the tag has the format:

subsonic_playlist: ";first playlist;second playlist;"

To get all items in a playlist use the query *;playlist name;*.

Command Line Usage

To use the `subsonicplaylist` plugin, enable it in your configuration (see *Using Plugins*). Then use it by invoking the `subsonicplaylist` command. Next, configure the plugin to connect to your Subsonic server, like this:

```
subsonicplaylist:
    base_url: http://subsonic.example.com
    username: someUser
    password: somePassword
```

After this you can import your playlists by invoking the *subsonicplaylist* command.

By default only the tags of the items found for playlists will be updated. This means that, if one imported a playlist, then delete one song from it and imported the playlist again, the deleted song will still have the playlist set in its *subsonic_playlist* tag. To solve this problem one can use the *-d/–delete* flag. This resets all *subsonic_playlist* tag before importing playlists.

Here's an example configuration with all the available options and their default values:

```
subsonicplaylist:
    base_url: "https://your.subsonic.server"
    delete: no
    playlist_ids: []
    playlist_names: []
    username: ''
    password: ''
```

The *base_url*, *username*, and *password* options are required.

SubsonicUpdate Plugin

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

To use `subsonicupdate` plugin, enable it in your configuration (see *Using Plugins*). Then, you'll probably want to configure the specifics of your Subsonic server. You can do that using a `subsonic:` section in your `config.yaml`, which looks like this:

```
subsonic:
    url: https://example.com:443/subsonic
    user: username
    pass: password
    auth: token
```

With that all in place, beets will send a Rest API to your Subsonic server every time you import new music. Due to a current limitation of the API, all libraries visible to that user will be scanned.

This plugin requires Subsonic with an active Premium license (or active trial).

Configuration

The available options under the `subsonic:` section are:

- **url**: The Subsonic server resource. Default: `http://localhost:4040`
- **user**: The Subsonic user. Default: `admin`
- **pass**: The Subsonic user password. (This may either be a clear-text password or hex-encoded with the prefix `enc:`.) Default: `admin`
- **auth**: The authentication method. Possible choices are `token` or `password`. `token` authentication is preferred to avoid sending cleartext password.

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 the `the` plugin, enable it (see *Plugins*) and then use a template function called `%the` 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.

Configuration

To configure the plugin, make a `the:` section in your configuration file. The available options are:

- **a**: Handle "A/An" moves. Default: `yes`.
- **the**: handle "The" moves. Default: `yes`.
- **patterns**: Custom regexp patterns, space-separated. 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. Default: `[]`.
- **strip**: Remove the article altogether instead of moving it to the end. Default: `no`.
- **format**: A Python format string for the output. Use `{0}` to indicate the part without the article and `{1}` for the article. Spaces are already trimmed from ends of both parts. Default: `'{0}, {1}'`.

Thumbnails Plugin

The `thumbnails` plugin creates thumbnails for your album folders with the album cover. This works on freedesktop.org-compliant file managers such as Nautilus or Thunar, and is therefore POSIX-only.

To use the `thumbnails` plugin, enable it (see *Plugins*) as well as the *FetchArt Plugin*. You'll need 2 additional python packages: pyxdg and pathlib.

`thumbnails` needs to resize the covers, and therefore requires either ImageMagick or Pillow.

Configuration

To configure the plugin, make a `thumbnails` section in your configuration file. The available options are

- **auto**: Whether the thumbnail should be automatically set on import. Default: `yes`.
- **force**: Generate the thumbnail even when there's one that seems fine (more recent than the cover art). Default: `no`.
- **dolphin**: Generate dolphin-compatible thumbnails. Dolphin (KDE file explorer) does not respect freedesktop.org's standard on thumbnails. This functionality replaces the *Freedesktop Plugin* Default: `no`

Usage

The `thumbnails` command provided by this plugin creates a thumbnail for albums that match a query (see *Queries*).

Types Plugin

The `types` plugin lets you declare types for attributes you use in your library. For example, you can declare that a `rating` field is numeric so that you can query it with ranges—which isn't possible when the field is considered a string (the default).

Enable the `types` plugin as described in *Plugins* and then add a `types` section to your *configuration file*. The configuration section should map field name to one of `int`, `float`, `bool`, or `date`.

Here's an example:

```
types:
    rating: int
```

Now you can assign numeric ratings to tracks and albums and use *range queries* to filter them.:

```
beet modify "My favorite track" rating=5
beet ls rating:4..5

beet modify --album "My favorite album" rating=5
beet ls --album rating:4..5
```

Unimported Plugin

The `unimported` plugin allows one to list all files in the library folder which are not listed in the beets library database, including art files.

Command Line Usage

To use the `unimported` plugin, enable it in your configuration (see *Using Plugins*). Then use it by invoking the `beet unimported` command. The command will list all files in the library folder which are not imported. You can exclude file extensions or entire subdirectories using the configuration file:

```
unimported:
    ignore_extensions: jpg png
    ignore_subdirectories: NonMusic data temp
```

The default configuration lists all unimported files, ignoring no extensions.

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.

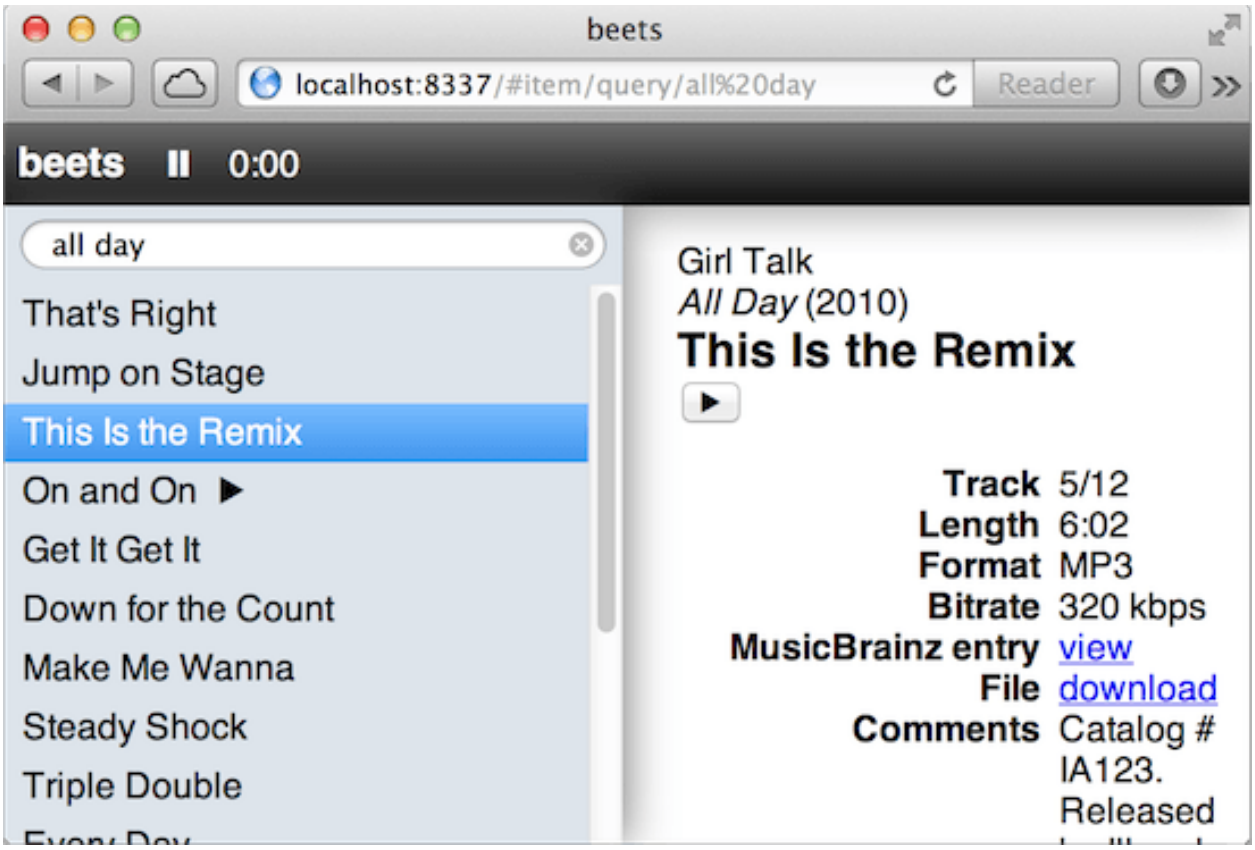
Install

The Web interface depends on Flask. To get it, just run `pip install flask`. Then enable the `web` plugin in your configuration (see *Using Plugins*).

If you need CORS (it's disabled by default—see *Cross-Origin Resource Sharing (CORS)*, below), then you also need flask-cors. Just type `pip install flask-cors`.

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]`. Or the configuration options below.

Usage

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

Configuration

To configure the plugin, make a `web:` section in your configuration file. The available options are:

- **host**: The server hostname. Set this to 0.0.0.0 to bind to all interfaces. Default: Bind to 127.0.0.1.
- **port**: The server port. Default: 8337.
- **cors**: The CORS allowed origin (see *Cross-Origin Resource Sharing (CORS)*, below). Default: CORS is disabled.
- **cors_supports_credentials**: Support credentials when using CORS (see *Cross-Origin Resource Sharing (CORS)*, below). Default: CORS_SUPPORTS_CREDENTIALS is disabled.
- **reverse_proxy**: If true, enable reverse proxy support (see *Reverse Proxy Support*, below). Default: false.
- **include_paths**: If true, includes paths in item objects. Default: false.
- **readonly**: If true, DELETE and PATCH operations are not allowed. Only GET is permitted. Default: true.

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

Cross-Origin Resource Sharing (CORS)

The `web` plugin's API can be used as a backend for an in-browser client. By default, browsers will only allow access from clients running on the same server as the API. (You will get an arcane error about `XMLHttpRequest` otherwise.) A technology called CORS lets you relax this restriction.

If you want to use an in-browser client hosted elsewhere (or running from a different server on your machine), first install the flask-cors plugin by typing `pip install flask-cors`. Then set the `cors` configuration option to the "origin" (protocol, host, and optional port number) where the client is served. Or set it to `'*'` to enable access from all origins. Note that there are security implications if you set the origin to `'*'`, so please research this before using it.

If the `web` server is behind a proxy that uses credentials, you might want to set the `cors_supports_credentials` configuration option to true to let in-browser clients log in.

For example:

```
web:
    host: 0.0.0.0
    cors: 'http://example.com'
```

Reverse Proxy Support

When the server is running behind a reverse proxy, you can tell the plugin to respect forwarded headers. Specifically, this can help when you host the plugin at a base URL other than the root `/` or when you use the proxy to handle secure connections. Enable the `reverse_proxy` configuration option if you do this.

Technically, this option lets the proxy provide `X-Script-Name` and `X-Scheme` HTTP headers to control the plugin's the `SCRIPT_NAME` and its `wsgi.url_scheme` parameter.

Here's a sample Nginx configuration that serves the web plugin under the /beets directory:

```
location /beets {
    proxy_pass http://127.0.0.1:8080;
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Scheme $scheme;
    proxy_set_header X-Script-Name /beets;
}
```

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.

DELETE /item/6

Removes the item with id *6* from the beets library. If the *?delete* query string is included, the matching file will be deleted from disk.

Only allowed if `readonly` configuration option is set to `no`.

PATCH /item/6

Updates the item with id *6* and write the changes to the music file. The body should be a JSON object containing the changes to the object.

Returns the updated JSON representation.

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

Only allowed if `readonly` configuration option is set to `no`.

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.

This endpoint also supports *DELETE* and *PATCH* methods as above, to operate on all items of the list.

GET /item/path/...

Look for an item at the given absolute path on the server. If it corresponds to a track, return the track in the same format as `/item/*`.

If the server runs UNIX, you'll need to include an extra leading slash: `http://localhost:8337/item/path//Users/beets/Music/Foo/Bar/Baz.mp3`

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" }
  ]
}
```

Path elements are joined as parts of a query. For example, `/item/query/foo/bar` will be converted to the query `foo,bar`. To specify literal path separators in a query, use a backslash instead of a slash.

This endpoint also supports *DELETE* and *PATCH* methods as above, to operate on all items returned by the query.

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/art`
- `DELETE /album/5`
- `GET /album/5,7`
- `DELETE /album/5,7`
- `GET /album/query/querystring`
- `DELETE /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`. You can also add the '?expand' flag to get the individual items of an album.

`DELETE` is only allowed if `readonly` configuration option is set to `no`.

GET /stats

Responds with the number of tracks and albums in the database.

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

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."

The plugin can work in one of two modes:

- `fields`: A blacklist, where you choose the tags you want to remove (used by default).
- `keep_fields`: A whitelist, where you instead specify the tags you want to keep.

To use the `zero` plugin, enable the plugin in your configuration (see *Using Plugins*).

Configuration

Make a `zero:` section in your configuration file. You can specify the fields to nullify and the conditions for nullifying them:

- Set `auto` to `yes` to null fields automatically on import. Default: `yes`.
- Set `fields` to a whitespace-separated list of fields to remove. You can get the list of all available fields by running `beet fields`. In addition, the `images` field allows you to remove any images embedded in the media file.
- Set `keep_fields` to *invert* the logic of the plugin. Only these fields will be kept; other fields will be removed. Remember to set only `fields` or `keep_fields`—not both!
- To conditionally filter a field, use `field: [regexp, regexp]` to specify regular expressions.
- By default this plugin only affects files' tags; the beets database is left unchanged. To update the tags in the database, set the `update_database` option to true.

For example:

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

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".

Manually Triggering Zero

You can also type `beet zero [QUERY]` to manually invoke the plugin on music in your library.

Preserving Album Art

If you use the `keep_fields` option, the plugin will remove embedded album art from files' tags unless you tell it not to. To keep the album art, include the special field `images` in the list. For example:

```
zero:
    keep_fields: title artist album year track genre images
```

1.3.3 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.
- *Spotify Plugin*: Search for releases in the Spotify database.
- *Deezer Plugin*: Search for releases in the Deezer database.
- *FromFilename Plugin*: Guess metadata for untagged tracks from their filenames.

1.3.4 Metadata

- *AcousticBrainz Submit Plugin*: Analyse audio with the streaming_extractor_music program and submit the metadata to the AcousticBrainz server
- *AcousticBrainz Plugin*: Fetch various AcousticBrainz metadata
- *BPM Plugin*: Measure tempo using keystrokes.
- *BPSync Plugin*: Fetch updated metadata from Beatport.
- *Edit Plugin*: Edit metadata from a text editor.
- *EmbedArt Plugin*: Embed album art images into files' metadata.
- *FetchArt Plugin*: Fetch album cover art from various sources.
- *FtInTitle Plugin*: Move "featured" artists from the artist field to the title field.
- *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.
- *LastGenre Plugin*: Fetch genres based on Last.fm tags.
- *LastImport Plugin*: Collect play counts from Last.fm.
- *Lyrics Plugin*: Automatically fetch song lyrics.
- *MBSync Plugin*: Fetch updated metadata from MusicBrainz.
- *MetaSync Plugin*: Fetch metadata from local or remote sources
- *MPDStats Plugin*: Connect to MPD and update the beets library with play statistics (last_played, play_count, skip_count, rating).

- *ParentWork Plugin*: Fetch work titles and works they are part of.
- *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.

1.3.5 Path Formats

- *AlbumTypes Plugin*: Format album type in path formats.
- *Bucket Plugin*: Group your files into bucket directories that cover different field values ranges.
- *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).

1.3.6 Interoperability

- *AURA Plugin*: A server implementation of the AURA specification.
- *Bad Files Plugin*: Check audio file integrity.
- *EmbyUpdate Plugin*: Automatically notifies Emby whenever the beets library changes.
- *Fish Plugin*: Adds Fish shell tab autocompletion to `beet` commands.
- *ImportFeeds Plugin*: Keep track of imported files via `.m3u` playlist file(s) or symlinks.
- *IPFS Plugin*: Import libraries from friends and get albums from them via ipfs.
- *KodiUpdate Plugin*: Automatically notifies Kodi whenever the beets library changes.
- *MPDUpdate Plugin*: Automatically notifies MPD whenever the beets library changes.
- *Play Plugin*: Play beets queries in your music player.
- *Playlist Plugin*: Use M3U playlists to query the beets library.
- *PlexUpdate Plugin*: Automatically notifies Plex whenever the beets library changes.
- *Smart Playlist Plugin*: Generate smart playlists based on beets queries.
- *SonosUpdate Plugin*: Automatically notifies Sonos whenever the beets library changes.
- *Thumbnails Plugin*: Get thumbnails with the cover art on your album folders.
- *SubsonicUpdate Plugin*: Automatically notifies Subsonic whenever the beets library changes.

1.3.7 Miscellaneous

- *Bare-ASCII Search Plugin*: Search albums and tracks with bare ASCII string matching.
- *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.
- *Duplicates Plugin*: List duplicate tracks or albums.
- *Export Plugin*: Export data from queries to a format.

- *FileFilter Plugin*: Automatically skip files during the import process based on regular expressions.
- *Fuzzy Search Plugin*: Search albums and tracks with fuzzy string matching.
- *Hook Plugin*: Run a command when an event is emitted by beets.
- *IHate Plugin*: Automatically skip albums and tracks during the import process.
- *Info Plugin*: Print music files' tags to the console.
- *Load Extension Plugin*: Load SQLite extensions.
- *MusicBrainz Collection Plugin*: Maintain your MusicBrainz collection list.
- *MusicBrainz Submit Plugin*: Print an album's tracks in a MusicBrainz-friendly format.
- *Missing Plugin*: List missing tracks.
- mstream: A music streaming server + webapp that can be used alongside beets.
- *Random Plugin*: Randomly choose albums and tracks from your library.
- *Spotify Plugin*: Create Spotify playlists from the Beets library.
- *Types Plugin*: Declare types for flexible attributes.
- *Web Plugin*: An experimental Web-based GUI for beets.

1.3.8 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:

- beets-alternatives manages external files.
- beet-amazon adds Amazon.com as a tagger data source.
- beets-artistcountry fetches the artist's country of origin from MusicBrainz.
- beets-autofix automates repetitive tasks to keep your library in order.
- beets-barcode lets you scan or enter barcodes for physical media to search for their metadata.
- beetcamp enables **bandcamp.com** autotagger with a fairly extensive amount of metadata.
- beets-bpmanalyser analyses songs and calculates their tempo (BPM).
- beets-check automatically checksums your files to detect corruption.
- A cmus plugin integrates with the cmus console music player.
- beets-copyartifacts helps bring non-music files along during import.
- beets-describe gives you the full picture of a single attribute of your library items.
- drop2beets automatically imports singles as soon as they are dropped in a folder (using Linux's `inotify`). You can also set a sub-folders hierarchy to set flexible attributes by the way.
- dsedivec has two plugins: `edit` and `moveall`.

- beets-follow lets you check for new albums from artists you like.
- beetFs is a FUSE filesystem for browsing the music in your beets library. (Might be out of date.)
- beets-goingrunning generates playlists to go with your running sessions.
- beets-ibroadcast uploads tracks to the iBroadcast cloud service.
- beets-importreplace lets you perform regex replacements on incoming metadata.
- beets-mosaic generates a montage of a mosaic from cover art.
- beets-noimport adds and removes directories from the incremental import skip list.
- beets-originquery augments MusicBrainz queries with locally-sourced data to improve autotagger results.
- beets-popularity fetches popularity values from Spotify.
- beets-setlister generate playlists from the setlists of a given artist.
- beet-summarize can compute lots of counts and statistics about your music library.
- beets-usertag lets you use keywords to tag and organize your music.
- whatlastgenre fetches genres from various music sites.
- beets-xtractor extracts low- and high-level musical information from your songs.
- beets-ydl downloads audio from youtube-dl sources and import into beets.
- beets-yearfixer attempts to fix all missing `original_year` and `year` fields.

1.4 FAQ

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

- *How do I...*
  - *... rename my files according to a new path format configuration?*
  - *... find all the albums I imported "as-is"?*
  - *... create "Disc N" directories for multi-disc albums?*
  - *... import a multi-disc album?*
  - *... enter a MusicBrainz ID?*
  - *... upgrade to the latest version of beets?*
  - *... run the latest source version of beets?*
  - *... report a bug in beets?*
  - *... find the configuration file (config.yaml)?*
  - *... avoid using special characters in my filenames?*
  - *... point beets at a new music directory?*
- *Why does beets...*
  - *... complain that it can't find a match?*

- *. . . appear to be missing some plugins?*
- *. . . ignore control-C during an import?*
- *. . . not change my ID3 tags?*
- *. . . complain that a file is "unreadable"?*
- *. . . seem to "hang" after an import finishes?*
- *. . . put a bunch of underscores in my filenames?*
- *. . . say "command not found"?*

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:

- `https://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. Choose one of these methods:
   - Use `pip` to install the latest snapshot tarball. Type: `pip install https://github.com/beetbox/beets/tarball/master`
   - Grab the source using git. First, clone the repository: `git clone https://github.com/beetbox/beets.git`. Then, `cd beets` and `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 git+https://github.com/beetbox/beets#egg=beets` to clone beets and install it, allowing you to modify the source in-place to try out changes.
   - Combine the previous two approaches, cloning the source yourself, and then installing in editable mode: `git clone https://github.com/beetbox/beets.git` then `pip install -e beets`. This approach lets you decide where the source is stored, with any changes immediately reflected in your environment.

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*.

. . . point beets at a new music directory?

If you want to move your music from one directory to another, the best way is to let beets do it for you. First, edit your configuration and set the `directory` setting to the new place. Then, type `beet move` to have beets move all your files.

If you've already moved your music *outside* of beets, you have a few options:

- Move the music back (with an ordinary `mv`) and then use the above steps.
- Delete your database and re-create it from the new paths using `beet import -AWC`.
- Resort to manually modifying the SQLite database (not recommended).

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 relevant question*).

To see the beets documentation for your version (and avoid confusion with new features in trunk), select your version from the menu in the sidebar.

. . . 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 `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 Contributing

Contents

- *Contributing*
  - *Thank you!*
  - *Types of Contributions*
    - *Non-Programming*
    - *Programming*
  - *Your First Contribution*
  - *How to Submit Your Work*
  - *The Code*
  - *Coding Conventions*
    - *General*
    - *Style*
    - *Handling Paths*
    - *Editor Settings*
  - *Testing*

* *Running the Tests*
* *Writing Tests*

1.5.1 Thank you!

First off, thank you for considering contributing to beets! It's people like you that make beets continue to succeed.

These guidelines describe how you can help most effectively. By following these guidelines, you can make life easier for the development team as it indicates you respect the maintainers' time; in return, the maintainers will reciprocate by helping to address your issue, review changes, and finalize pull requests.

1.5.2 Types of Contributions

We love to get contributions from our community—you! There are many ways to contribute, whether you're a programmer or not.

Non-Programming

- Promote beets! Help get the word out by telling your friends, writing a blog post, or discussing it on a forum you frequent.
- Improve the documentation. It's incredibly easy to contribute here: just find a page you want to modify and hit the "Edit on GitHub" button in the upper-right. You can automatically send us a pull request for your changes.
- GUI design. For the time being, beets is a command-line-only affair. But that's mostly because we don't have any great ideas for what a good GUI should look like. If you have those great ideas, please get in touch.
- Benchmarks. We'd like to have a consistent way of measuring speed improvements in beets' tagger and other functionality as well as a way of comparing beets' performance to other tools. You can help by compiling a library of freely-licensed music files (preferably with incorrect metadata) for testing and measurement.
- Think you have a nice config or cool use-case for beets? We'd love to hear about it! Submit a post to our our forums under the "Show and Tell" category for a chance to get featured in the docs.
- Consider helping out in our forums by responding to support requests or driving some new discussions.

Programming

- As a programmer (even if you're just a beginner!), you have a ton of opportunities to get your feet wet with beets.
- For developing plugins, or hacking away at beets, there's some good information in the "For Developers" section of the docs.

Getting the Source

The easiest way to get started with the latest beets source is to use pip to install an "editable" package. This can be done with one command:

```
$ pip install -e git+https://github.com/beetbox/beets.git#egg=beets
```

Or, equivalently:

```
$ git clone https://github.com/beetbox/beets.git
$ cd beets
$ pip install -e .
```

If you already have a released version of beets installed, you may need to remove it first by typing `pip uninstall beets`. The pip command above will put the beets source in a `src/beets` directory and install the `beet` CLI script to a standard location on your system. You may want to use the `--src` option to specify the parent directory where the source will be checked out and the `--user` option such that the package will be installed to your home directory (compare with the output of `pip install --help`).

Code Contribution Ideas

- We maintain a set of issues marked as "bite-sized". These are issues that would serve as a good introduction to the codebase. Claim one and start exploring!
- Like testing? Our test coverage is somewhat low. You can help out by finding low-coverage modules or checking out other testing-related issues.
- There are several ways to improve the tests in general (see *Testing* and some places to think about performance optimization (see Optimization).
- Not all of our code is up to our coding conventions. In particular, the library API documentation are currently quite sparse. You can help by adding to the docstrings in the code and to the documentation pages themselves. beets follows PEP-257 for docstrings and in some places, we also sometimes use ReST autodoc syntax for Sphinx to, for example, refer to a class name.

1.5.3 Your First Contribution

If this is your first time contributing to an open source project, welcome! If you are confused at all about how to contribute or what to contribute, take a look at this great tutorial, or stop by our forums if you have any questions.

We maintain a list of issues we reserved for those new to open source labeled "first timers only". Since the goal of these issues is to get users comfortable with contributing to an open source project, please do not hesitate to ask any questions.

1.5.4 How to Submit Your Work

Do you have a great bug fix, new feature, or documentation expansion you'd like to contribute? Follow these steps to create a GitHub pull request and your code will ship in no time.

1. Fork the beets repository and clone it (see above) to create a workspace.
2. Make your changes.
3. Add tests. If you've fixed a bug, write a test to ensure that you've actually fixed it. If there's a new feature or plugin, please contribute tests that show that your code does what it says.
4. Add documentation. If you've added a new command flag, for example, find the appropriate page under `docs/` where it needs to be listed.
5. Add a changelog entry to `docs/changelog.rst` near the top of the document.
6. Run the tests and style checker. The easiest way to run the tests is to use tox. For more information on running tests, see *Testing*.
7. Push to your fork and open a pull request! We'll be in touch shortly.

8. If you add commits to a pull request, please add a comment or re-request a review after you push them since GitHub doesn't automatically notify us when commits are added.

Remember, code contributions have four parts: the code, the tests, the documentation, and the changelog entry. Thank you for contributing!

1.5.5 The Code

The documentation has a section on the library API that serves as an introduction to beets' design.

1.5.6 Coding Conventions

General

There are a few coding conventions we use in beets:

- Whenever you access the library database, do so through the provided Library methods or via a Transaction object. Never call `lib.conn.*` directly. For example, do this:

```
with g.lib.transaction() as tx:
      rows = tx.query('SELECT DISTINCT "{0}" FROM "{1}" ORDER BY "{2}"'
                      .format(field, model._table, sort_field))
```

  To fetch Item objects from the database, use lib.items(...) and supply a query as an argument. Resist the urge to write raw SQL for your query. If you must use lower-level queries into the database, do this:

```
with lib.transaction() as tx:
    rows = tx.query('SELECT ...')
```

  Transaction objects help control concurrent access to the database and assist in debugging conflicting accesses.

- Always use the future imports `print_function`, `division`, and `absolute_import`, but *not* `unicode_literals`. These help keep your code modern and will help in the eventual move to Python 3.
- `str.format()` should be used instead of the `%` operator
- Never `print` informational messages; use the logging module instead. In particular, we have our own logging shim, so you'll see `from beets import logging` in most files.
  - Always log Unicode strings (e.g., `log.debug(u"hello world")`).
  - The loggers use str.format-style logging instead of `%`-style, so you can type `log.debug(u"{0}", obj)` to do your formatting.
- Exception handlers must use `except A as B:` instead of `except A, B:`.

Style

We follow PEP 8 and google's docstring format.

You can use `tox -e lint` to check your code for any style errors.

Handling Paths

A great deal of convention deals with the handling of **paths**. Paths are stored internally—in the database, for instance—as byte strings (i.e., `bytes` instead of `str` in Python 3). This is because POSIX operating systems' path names are only reliably usable as byte strings—operating systems typically recommend but do not require that filenames use a given encoding, so violations of any reported encoding are inevitable. On Windows, the strings are always encoded with UTF-8; on Unix, the encoding is controlled by the filesystem. Here are some guidelines to follow:

- If you have a Unicode path or you're not sure whether something is Unicode or not, pass it through `bytestring_path` function in the `beets.util` module to convert it to bytes.
- Pass every path name trough the `syspath` function (also in `beets.util`) before sending it to any *operating system* file operation (`open`, for example). This is necessary to use long filenames (which, maddeningly, must be Unicode) on Windows. This allows us to consistently store bytes in the database but use the native encoding rule on both POSIX and Windows.
- Similarly, the `displayable_path` utility function converts bytestring paths to a Unicode string for displaying to the user. Every time you want to print out a string to the terminal or log it with the `logging` module, feed it through this function.

Editor Settings

Personally, I work on beets with vim. Here are some `.vimrc` lines that might help with PEP 8-compliant Python coding:

```
filetype indent on
autocmd FileType python setlocal shiftwidth=4 tabstop=4 softtabstop=4 expandtab␣
↪shiftround autoindent
```

Consider installing this alternative Python indentation plugin. I also like neomake with its flake8 checker.

1.5.7 Testing

Running the Tests

To run the tests for multiple Python versions, compile the docs, and check style, use tox. Just type `tox` or use something like `tox -e py27` to test a specific configuration. detox makes this go faster.

You can disable a hand-selected set of "slow" tests by setting the environment variable SKIP_SLOW_TESTS before running them.

Other ways to run the tests:

- `python testall.py` (ditto)
- `python -m unittest discover -p 'test_*'` (ditto)
- pytest

You can also see the latest test results on Linux and on Windows.

Note, if you are on Windows and are seeing errors running tox, it may be related to this issue, in which case you may have to install tox v3.8.3 e.g. `python -m pip install tox=3.8.3`

Coverage

`tox -e cov` will add coverage info for tests: Coverage is pretty low still – see the current status on Codecov.

Red Flags

The pytest-random plugin makes it easy to randomize the order of tests. `py.test test --random` will occasionally turn up failing tests that reveal ordering dependencies—which are bad news!

Test Dependencies

The tests have a few more dependencies than beets itself. (The additional dependencies consist of testing utilities and dependencies of non-default plugins exercised by the test suite.) The dependencies are listed under 'test' in `extras_require` in setup.py. To install the test dependencies, run `python -m pip install .[test]`. Or, just run a test suite with `tox` which will install them automatically.

Writing Tests

Writing tests is done by adding or modifying files in folder test. Take a look at https://github.com/beetbox/beets/blob/master/test/test_template.py#L224 to get a basic view on how tests are written. We currently allow writing tests with either unittest or pytest.

Any tests that involve sending out network traffic e.g. an external API call, should be skipped normally and run under our weekly integration test suite. These tests can be useful in detecting external changes that would affect `beets`. In order to do this, simply add the following snippet before the applicable test case:

```
@unittest.skipUnless(
    os.environ.get('INTEGRATION_TEST', '0') == '1',
    'integration testing not enabled')
```

If you do this, it is also advised to create a similar test that 'mocks' the network call and can be run under normal circumstances by our CI and others. See unittest.mock for more info.

- **AVOID** using the `start()` and `stop()` methods of `mock.patch`, as they require manual cleanup. Use the annotation or context manager forms instead.

1.6 For Developers

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

See also the documentation for MediaFile, the library used by beets to read and write metadata tags in media files.

1.6.1 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 a `CommonOptionsParser` instance. Just use it like you would a normal `OptionParser` in an independent script. Note that it offers several methods to add common options: `--album`, `--path` and `--format`. This feature is versatile and extensively documented, try `pydoc beets.ui.CommonOptionsParser` for more information.

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 `BeetsPlugin.register_listener`. Here's an example:

```
from beets.plugins import BeetsPlugin

def loaded():
    print 'Plugin loaded!'

class SomePlugin(BeetsPlugin):
  def __init__(self):
    super(SomePlugin, self).__init__()
    self.register_listener('pluginload', loaded)
```

Note that if you want to access an attribute of your plugin (e.g. `config` or `log`) you'll have to define a method and not a function. Here is the usual registration process in this case:

```
from beets.plugins import BeetsPlugin

class SomePlugin(BeetsPlugin):
  def __init__(self):
    super(SomePlugin, self).__init__()
    self.register_listener('pluginload', self.loaded)

  def loaded(self):
    self._log.info('Plugin loaded!')
```

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`
- *album_removed*: called with an `Album` object every time an album is removed from the library (even when its file is not deleted from disk).
- *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_linked*: called with an `Item` object whenever a symlink is created for a file. Parameters: `item`, `source` path, `destination` path
- *item_hardlinked*: called with an `Item` object whenever a hardlink is created for a file. Parameters: `item`, `source` path, `destination` path

- *item_reflinked*: called with an `Item` object whenever a reflink is created for a file. 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, a `path`, and a `tags` dictionary just before a file's metadata is written to disk (i.e., just before the file on disk is opened). Event handlers may change the `tags` dictionary to customize the tags that are written to the media file. Event handlers may also 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_created*: called immediately after an import task is initialized. Plugins can use this to, for example, change imported files of a task before anything else happens. It's also possible to replace the task with another task by returning a list of tasks. This list can contain zero or more *ImportTask*'s. Returning an empty list will stop the task. Parameters: ''*task*' (an *ImportTask*) and `session` (an *ImportSession*).
- *import_task_start*: called when before an import task begins processing. Parameters: `task` and `session`.
- *import_task_apply*: called after metadata changes have been applied in an import task. This is called on the same thread as the UI, so use this sparingly and only for tasks that can be done quickly. For most plugins, an import pipeline stage is a better choice (see *Add Import Pipeline Stages*). Parameters: `task` and `session`.
- *import_task_before_choice*: called after candidate search for an import task before any decision is made about how/if to import or tag. Can be used to present information about the task or initiate interaction with the user before importing occurs. Return an importer action to take a specific action. Only one handler may return a non-None result. 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. Parameters: `lib` and `model`.
- *cli_exit*: called just before the `beet` command-line program exits. Parameter: `lib`.
- *import_begin*: called just before a `beet import` session starts up. Parameter: `session`.
- *trackinfo_received*: called after metadata for a track item has been fetched from a data source, such as MusicBrainz. You can modify the tags that the rest of the pipeline sees on a `beet import` operation or during later adjustments, such as `mbsync`. Slow handlers of the event can impact the operation, since the event is fired for any fetched possible match *before* the user (or the autotagger machinery) gets to see the match. Parameter: `info`.
- *albuminfo_received*: like *trackinfo_received*, the event indicates new metadata for album items. The parameter is an `AlbumInfo` object instead of a `TrackInfo`. Parameter: `info`.
- *before_choose_candidate*: called before the user is prompted for a decision during a `beet import` interactive session. Plugins can use this event for *appending choices to the prompt* by returning a list of `PromptChoices`. Parameters: `task` and `session`.
- *mb_track_extract*: called after the metadata is obtained from MusicBrainz. The parameter is a `dict` containing the tags retrieved from MusicBrainz for a track. Plugins must return a new (potentially empty) `dict` with

additional `field: value` pairs, which the autotagger will apply to the item, as flexible attributes if `field` is not a hardcoded field. Fields already present on the track are overwritten. Parameter: `data`

- *mb_album_extract*: Like *mb_track_extract*, but for album tags. Overwrites tags set at the track level, if they have the same `field`. Parameter: `data`

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

Extend the Autotagger

Plugins 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 Confuse 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.

If your plugin provides configuration values for sensitive data (e.g., passwords, API keys, ...), you should add these to the config so they can be redacted automatically when users dump their config. This can be done by setting each value's *redact* flag, like so:

```
self.config['password'].redact = True
```

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 expands 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. 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 `ImportSession` 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, session, task):
        print('Importing something!')
```

It is also possible to request your function to run early in the pipeline by adding the function to the plugin's `early_import_stages` field instead:

```
self.early_import_stages = [self.stage]
```

Extend the Query Syntax

You can add new kinds of queries to beets' *query syntax*. There are two ways to add custom queries: using a prefix and using a name. Prefix-based query extension can apply to *any* field, while named queries are not associated with any field. For example, beets already supports regular expression queries, which are indicated by a colon prefix—plugins can do the same.

For either kind of query extension, define a subclass of the `Query` type from the `beets.dbcore.query` module. Then:

- To define a prefix-based query, define a `queries` method in your plugin class. Return from this method a dictionary mapping prefix strings to query classes.
- To define a named query, defined dictionaries named either `item_queries` or `album_queries`. These should map names to query types. So if you use `{ "foo": FooQuery }`, then the query `foo:bar` will construct a query like `FooQuery("bar")`.

For prefix-based queries, you will want to extend `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
        }
```

Flexible Field Types

If your plugin uses flexible fields to store numbers or other non-string values, you can specify the types of those fields. A rating plugin, for example, might want to declare that the `rating` field should have an integer type:

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

class RatingPlugin(BeetsPlugin):
    item_types = {'rating': types.INTEGER}

    @property
    def album_types(self):
        return {'rating': types.INTEGER}
```

A plugin may define two attributes: *item_types* and *album_types*. Each of those attributes is a dictionary mapping a flexible field name to a type instance. You can find the built-in types in the *beets.dbcore.types* and *beets.library* modules or implement your own type by inheriting from the *Type* class.

Specifying types has several advantages:

- Code that accesses the field like `item['my_field']` gets the right type (instead of just a string).
- You can use advanced queries (like *ranges*) from the command line.
- User input for flexible fields may be validated and converted.

Logging

Each plugin object has a `_log` attribute, which is a `Logger` from the standard Python logging module. The logger is set up to PEP 3101, str.format-style string formatting. So you can write logging calls like this:

```
self._log.debug(u'Processing {0.title} by {0.artist}', item)
```

When beets is in verbose mode, plugin messages are prefixed with the plugin name to make them easier to see.

Which messages will be logged depends on the logging level and the action performed:

- Inside import stages and event handlers, the default is `WARNING` messages and above.
- Everywhere else, the default is `INFO` or above.

The verbosity can be increased with `--verbose` (`-v`) flags: each flags lowers the level by a notch. That means that, with a single `-v` flag, event handlers won't have their `DEBUG` messages displayed, but command functions (for example) will. With `-vv` on the command line, `DEBUG` messages will be displayed everywhere.

This addresses a common pattern where plugins need to use the same code for a command and an import stage, but the command needs to print more messages than the import stage. (For example, you'll want to log "found lyrics for this song" when you're run explicitly as a command, but you don't want to noisily interrupt the importer interface when running automatically.)

Append Prompt Choices

Plugins can also append choices to the prompt presented to the user during an import session.

To do so, add a listener for the `before_choose_candidate` event, and return a list of `PromptChoices` that represent the additional choices that your plugin shall expose to the user:

```
from beets.plugins import BeetsPlugin
from beets.ui.commands import PromptChoice

class ExamplePlugin(BeetsPlugin):
    def __init__(self):
        super(ExamplePlugin, self).__init__()
        self.register_listener('before_choose_candidate',
                               self.before_choose_candidate_event)

    def before_choose_candidate_event(self, session, task):
        return [PromptChoice('p', 'Print foo', self.foo),
                PromptChoice('d', 'Do bar', self.bar)]

    def foo(self, session, task):
        print('User has chosen "Print foo"!')

    def bar(self, session, task):
        print('User has chosen "Do bar"!')
```

The previous example modifies the standard prompt:

```
# selection (default 1), Skip, Use as-is, as Tracks, Group albums,
Enter search, enter Id, aBort?
```

by appending two additional options (`Print foo` and `Do bar`):

```
# selection (default 1), Skip, Use as-is, as Tracks, Group albums,
Enter search, enter Id, aBort, Print foo, Do bar?
```

If the user selects a choice, the `callback` attribute of the corresponding `PromptChoice` will be called. It is the responsibility of the plugin to check for the status of the import session and decide the choices to be appended: for example, if a particular choice should only be presented if the album has no candidates, the relevant checks against `task.candidates` should be performed inside the plugin's `before_choose_candidate_event` accordingly.

Please make sure that the short letter for each of the choices provided by the plugin is not already in use: the importer will emit a warning and discard all but one of the choices using the same letter, giving priority to the core importer prompt choices. As a reference, the following characters are used by the choices on the core importer prompt, and hence should not be used: `a`, `s`, `u`, `t`, `g`, `e`, `i`, `b`.

Additionally, the callback function can optionally specify the next action to be performed by returning a `importer.action` value. It may also return a `autotag.Proposal` value to update the set of current proposals to be considered.

1.6.2 Library Database API

This page describes the internal API of beets' core database features. It doesn't exhaustively document the API, but is aimed at giving an overview of the architecture to orient anyone who wants to dive into the code.

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

The *Library* is typically instantiated as a singleton. A single invocation of beets usually has only one *Library*. It's powered by `dbcore.Database` under the hood, which handles the SQLite abstraction, something like a very minimal ORM. The library is also responsible for handling queries to retrieve stored objects.

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

A database of music containing songs and albums.

__init__(*path='library.blb'*, *directory='~/Music'*, *path_formats=(('default', '$artist/$album/$track $title'),)*, *replacements=None*)

Initialize self. See help(type(self)) for accurate signature.

You can add new items or albums to the library:

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.

And there are methods for querying the database:

items(*query=None*, *sort=None*)
Get *Item* objects matching the query.

albums(*query=None*, *sort=None*)
Get *Album* objects matching the 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*.

Any modifications must go through a `Transaction` which you get can using this method:

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

Model Classes

The two model entities in beets libraries, *Item* and *Album*, share a base class, *LibModel*, that provides common functionality. That class itself specialises `dbcore.Model` which provides an ORM-like abstraction.

To get or change the metadata of a model (an item or album), either access its attributes (e.g., `print(album.year)` or `album.year = 2012`) or use the `dict`-like interface (e.g. `item['artist']`).

Model base

Models use dirty-flags to track when the object's metadata goes out of sync with the database. The dirty dictionary maps field names to booleans indicating whether the field has been written since the object was last synchronized (via load or store) with the database.

class beets.library.**LibModel**(*db=None*, ***values*)
Shared concrete functionality for Items and Albums.

classmethod all_keys()
Get a list of available keys for objects of this type. Includes fixed and computed fields.

__init__(*db=None*, ***values*)
Create a new object with an optional Database association and initial field values.

_types = {}

_fields = {}

There are CRUD-like methods for interacting with the database:

store(*fields=None*)
Save the object's metadata into the library database. :param fields: the fields to be stored. If not specified, all fields will be.

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

If check_revision is true, the database is only queried loaded when a transaction has been committed since the item was last loaded.

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

add(*lib=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.

The base class `dbcore.Model` has a `dict`-like interface, so normal the normal mapping API is supported:

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.

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

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

get(*key*, *default=None*, *raise_=False*)
Get the value for a field, or *default*. Alternatively, raise a KeyError if the field is not available.

Item

Each *Item* object represents a song or track. (We use the more generic term item because, one day, beets might support non-music media.) An item can either be purely abstract, in which case it's just a bag of metadata fields, or it can have an associated file (indicated by `item.path`).

In terms of the underlying SQLite database, items are backed by a single table called items with one column per metadata fields. The metadata fields currently in use are listed in `library.py` in `Item._fields`.

To read and write a file's tags, we use the MediaFile library. To make changes to either the database or the tags on a file, you update an item's fields (e.g., `item.title = "Let It Be"`) and then call `item.write()`.

Items also track their modification times (mtimes) to help detect when they become out of sync with on-disk metadata, mainly to speed up the *update* (which needs to check whether the database is in sync with the filesystem). This feature turns out to be sort of complicated.

For any *Item*, there are two mtimes: the on-disk mtime (maintained by the OS) and the database mtime (maintained by beets). Correspondingly, there is on-disk metadata (ID3 tags, for example) and DB metadata. The goal with the mtime is to ensure that the on-disk and DB mtimes match when the on-disk and DB metadata are in sync; this lets beets do a quick mtime check and avoid rereading files in some circumstances.

Specifically, beets attempts to maintain the following invariant:

> If the on-disk metadata differs from the DB metadata, then the on-disk mtime must be greater than the DB mtime.

As a result, it is always valid for the DB mtime to be zero (assuming that real disk mtimes are always positive). However, whenever possible, beets tries to set `db_mtime = disk_mtime` at points where it knows the metadata is synchronized. When it is possible that the metadata is out of sync, beets can then just set `db_mtime = 0` to return to a consistent state.

This leads to the following implementation policy:

- On every write of disk metadata (`Item.write()`), the DB mtime is updated to match the post-write disk mtime.
- Same for metadata reads (`Item.read()`).
- On every modification to DB metadata (`item.field = ...`), the DB mtime is reset to zero.

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

__init__(*db=None*, ***values*)
Create a new object with an optional Database association and initial field values.

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.

destination(*fragment=False*, *basedir=None*, *platform=None*, *path_formats=None*, *replacements=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.

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

The methods `read()` and `write()` are complementary: one reads a file's tags and updates the item's metadata fields accordingly while the other takes the item's fields and writes them to the file's tags.

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.

write(*path=None*, *tags=None*, *id3v23=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.

path is the path of the mediafile to write the data to. It defaults to the item's path.

tags is a dictionary of additional metadata the should be written to the file. (These tags need not be in *_media_fields*.)

id3v23 will override the global *id3v23* config option if it is set to something other than *None*.

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

try_write(**args*, ***kwargs*)
Calls *write()* but catches and logs *FileOperationError* exceptions.

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

try_sync(*write*, *move*, *with_album=True*)
Synchronize the item with the database and, possibly, updates its tags on disk and its path (by moving the file).

write indicates whether to write new tags into the file. Similarly, *move* controls whether the path should be updated. In the latter case, files are *only* moved when they are inside their library's directory (if any).

Similar to calling *write()*, *move()*, and `store()` (conditionally).

The *Item* class supplements the normal model interface so that they interacting with the filesystem as well:

move(*operation=<MoveOperation.MOVE: 0>*, *basedir=None*, *with_album=True*, *store=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.

Instead of moving the item it can also be copied, linked or hardlinked depending on *operation* which should be an instance of *util.MoveOperation*.

basedir overrides the library base directory for the destination.

If the item is in an album and *with_album* is *True*, the album is given an opportunity to move its art.

By default, 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. If *store* is *False* however, the item won't be stored and you'll have to manually store it after invoking this method.

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.

Album

An *Album* is a collection of Items in the database. Every item in the database has either zero or one associated albums (accessible via `item.album_id`). An item that has no associated album is called a singleton. Changing fields on an album (e.g. `album.year = 2012`) updates the album itself and also changes the same field in all associated items.

An *Album* object keeps track of album-level metadata, which is (mostly) a subset of the track-level metadata. The album-level metadata fields are listed in `Album._fields`. For those fields that are both item-level and album-level (e.g., `year` or `albumartist`), every item in an album should share the same value. Albums use an SQLite table called `albums`, in which each column is an album metadata field.

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.

__init__(*db=None*, ***values*)
Create a new object with an optional Database association and initial field values.

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

Albums extend the normal model interface to also forward changes to their items:

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

store(*fields=None*)
Update the database with the album information. The album's tracks are also updated. :param fields: The fields to be stored. If not specified, all fields will be.

try_sync(*write*, *move*)
Synchronize the album and its items with the database. Optionally, also write any new tags into the files and update their paths.

write indicates whether to write tags to the item files, and *move* controls whether files (both audio and album art) are moved.

move(*operation=<MoveOperation.MOVE: 0>*, *basedir=None*, *store=True*)
Move, copy, link or hardlink (depending on *operation*) all items to their destination. Any album art moves along with them.

basedir overrides the library base directory for the destination.

operation should be an instance of *util.MoveOperation*.

By default, the album is stored to the database, persisting any modifications to its metadata. If *store* is *False* however, the album is not stored automatically, and you'll have to manually store it after invoking this method.

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.

Albums also manage album art, image files that are associated with each album:

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.

Sends an 'art_set' event with *self* as the sole argument.

move_art(*operation=<MoveOperation.MOVE: 0>*)
Move, copy, link or hardlink (depending on *operation*) any existing album art so that it remains in the same directory as the items.

operation should be an instance of *util.MoveOperation*.

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.

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* (see this blog post for motivation). 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.

Queries

To access albums and items in a library, we use *Queries*. In beets, the `Query` abstract base class represents a criterion that matches items or albums in the database. Every subclass of `Query` must implement two methods, which implement two different ways of identifying matching items/albums.

The `clause()` method should return an SQLite `WHERE` clause that matches appropriate albums/items. This allows for efficient batch queries. Correspondingly, the `match(item)` method should take an `Item` object and return a boolean, indicating whether or not a specific item matches the criterion. This alternate implementation allows clients to determine whether items that have already been fetched from the database match the query.

There are many different types of queries. Just as an example, `FieldQuery` determines whether a certain field matches a certain value (an equality query). `AndQuery` (like its abstract superclass, `CollectionQuery`) takes a set of other query objects and bundles them together, matching only albums/items that match all constituent queries.

Beets has a human-writable plain-text query syntax that can be parsed into `Query` objects. Calling `AndQuery.from_strings` parses a list of query parts into a query object that can then be used with `Library` objects.

1.6.3 Music Importer

The importer component is responsible for the user-centric workflow that adds music to a library. This is one of the first aspects that a user experiences when using beets: it finds music in the filesystem, groups it into albums, finds corresponding metadata in MusicBrainz, asks the user for intervention, applies changes, and moves/copies files. A description of its user interface is given in *Using the Auto-Tagger*.

The workflow is implemented in the `beets.importer` module and is distinct from the core logic for matching MusicBrainz metadata (in the `beets.autotag` module). The workflow is also decoupled from the command-line interface with the hope that, eventually, other (graphical) interfaces can be bolted onto the same importer implementation.

The importer is multithreaded and follows the pipeline pattern. Each pipeline stage is a Python coroutine. The `beets.util.pipeline` module houses a generic, reusable implementation of a multithreaded pipeline.

1.6.4 Providing a CLI

The `beets.ui` module houses interactions with the user via a terminal, the *Command-Line Interface*. The main function is called when the user types beet on the command line. The CLI functionality is organized into commands, some of which are built-in and some of which are provided by plugins. The built-in commands are all implemented in the `beets.ui.commands` submodule.

1.7 Changelog

1.7.1 1.6.0 (November 27, 2021)

This release is our first experiment with time-based releases! We are aiming to publish a new release of beets every 3 months. We therefore have a healthy but not dizzyingly long list of new features and fixes.

With this release, beets now requires Python 3.6 or later (it removes support for Python 2.7, 3.4, and 3.5). There are also a few other dependency changes—if you're a maintainer of a beets package for a package manager, thank you for your ongoing efforts, and please see the list of notes below.

Major new features:

- When fetching genres from MusicBrainz, we now include genres from the release group (in addition to the release). We also prioritize genres based on the number of votes. Thanks to aereaux.
- Primary and secondary release types from MusicBrainz are now stored in a new `albumtypes` field. Thanks to edgars-supe. #2200
- An accompanying new *AlbumTypes Plugin* includes some options for formatting this new `albumtypes` field. Thanks to edgars-supe.

Other new things:

- *Permissions Plugin*: The plugin now sets cover art permissions to match the audio file permissions.
- *Unimported Plugin*: A new configuration option supports excluding specific subdirectories in library.
- *Info Plugin*: Add support for an `--album` flag.
- *Export Plugin*: Similarly add support for an `--album` flag.
- `beet move` now highlights path differences in color (when enabled).
- When moving files and a direct rename of a file is not possible (for example, when crossing filesystems), beets now copies to a temporary file in the target folder first and then moves to the destination instead of directly copying the target path. This gets us closer to always updating files atomically. Thanks to catap. #4060
- *FetchArt Plugin*: Add a new option to store cover art as non-progressive image. This is useful for DAPs that do not support progressive images. Set `deinterlace: yes` in your configuration to enable this conversion.
- *FetchArt Plugin*: Add a new option to change the file format of cover art images. This may also be useful for DAPs that only support some image formats.
- Support flexible attributes in `%aunique`. #2678 #3553
- Make `%aunique` faster, especially when using inline fields. #4145

Bug fixes:

- *Lyrics Plugin*: Fix a crash when Beautiful Soup is not installed. #4027
- *Discogs Plugin*: Support a new Discogs URL format for IDs. #4080
- *Discogs Plugin*: Remove built-in rate-limiting because the Discogs Python library we use now has its own rate-limiting. :bug: *4108*
- *Export Plugin*: Fix some duplicated output.
- *AURA Plugin*: Fix a potential security hole when serving image files. #4160

For plugin developers:

- *`beets.library.Item.destination()`* now accepts a *replacements* argument to be used in favor of the default.
- The *pluginload* event is now sent after plugin types and queries are available, not before.
- A new plugin event, *album_removed*, is called when an album is removed from the library (even when its file is not deleted from disk).

Here are some notes for packagers:

- As noted above, the minimum Python version is now 3.6.
- We fixed a flaky test, named *test_album_art* in the *test_zero.py* file, that some distributions had disabled. Disabling this test should no longer be necessary. #4037 #4038
- This version of beets no longer depends on the six library. #4030

- The *gmusic* plugin was removed since Google Play Music has been shut down. Thus, the optional dependency on *gmusicapi* does not exist anymore. #4089

1.7.2 1.5.0 (August 19, 2021)

This long overdue release of beets includes far too many exciting and useful features than could ever be satisfactorily enumerated. As a technical detail, it also introduces two new external libraries: MediaFile and Confuse used to be part of beets but are now reusable dependencies—packagers, please take note. Finally, this is the last version of beets where we intend to support Python 2.x and 3.5; future releases will soon require Python 3.6.

One non-technical change is that we moved our official `#beets` home on IRC from freenode to Libera.Chat.

Major new features:

- Fields in queries now fall back to an item's album and check its fields too. Notably, this allows querying items by an album's attribute: in other words, `beet list foo:bar` will not only find tracks with the *foo* attribute; it will also find tracks *on albums* that have the *foo* attribute. This may be particularly useful in the *Path Format Configuration*, which matches individual items to decide which path to use. Thanks to FichteFoll. #2797 #2988
- A new *reflink* config option instructs the importer to create fast, copy-on-write file clones on filesystems that support them. Thanks to rubdos.
- A new *Unimported Plugin* lets you find untracked files in your library directory.
- The *AURA Plugin* has arrived! Try out the future of remote music library access today.
- We now fetch information about works from MusicBrainz. MusicBrainz matches provide the fields `work` (the title), `mb_workid` (the MBID), and `work_disambig` (the disambiguation string). Thanks to dosoe. #2580 #3272
- A new *ParentWork Plugin* gets information about the original work, which is useful for classical music. Thanks to dosoe. #2580 #3279
- *BPD Plugin*: BPD now supports most of the features of version 0.16 of the MPD protocol. This is enough to get it talking to more complicated clients like ncmpcpp, but there are still some incompatibilities, largely due to MPD commands we don't support yet. (Let us know if you find an MPD client that doesn't get along with BPD!) #3214 #800
- A new *Deezer Plugin* can autotag tracks and albums using the Deezer database. Thanks to rhlahuja. #3355
- A new *Bare-ASCII Search Plugin* provides a new query type: "bare ASCII" queries that ignore accented characters, treating them as though they were plain ASCII characters. Use the # prefix with *list* or other commands. #3882
- *FetchArt Plugin*: The plugin can now get album art from last.fm. #3530
- *Web Plugin*: The API now supports the HTTP *DELETE* and *PATCH* methods for modifying items. They are disabled by default; set `readonly: no` in your configuration file to enable modification via the API. #3870

Other new things:

- `beet remove` now also allows interactive selection of items from the query, similar to `beet modify`.
- Enable HTTPS for MusicBrainz by default and add configuration option *https* for custom servers. See *MusicBrainz Options* for more details.
- *MPDStats Plugin*: Add a new *strip_path* option to help build the right local path from MPD information.
- *Convert Plugin*: Conversion can now parallelize conversion jobs on Python 3.
- *LastGenre Plugin*: Add a new *title_case* config option to make title-case formatting optional.
- There's a new message when running `beet config` when there's no available configuration file. #3779

- When importing a duplicate album, the prompt now says "keep all" instead of "keep both" to reflect that there may be more than two albums involved. #3569
- *Chromaprint/Acoustid Plugin*: The plugin now updates file metadata after generating fingerprints through the *submit* command.
- *LastGenre Plugin*: Added more heavy metal genres to the built-in genre filter lists.
- A new *Subsonic Playlist Plugin* can import playlists from a Subsonic server.
- *SubsonicUpdate Plugin*: The plugin now automatically chooses between token- and password-based authentication based on the server version.
- A new *extra_tags* configuration option lets you use more metadata in MusicBrainz queries to further narrow the search.
- A new *Fish Plugin* adds Fish shell tab autocompletion to beets.
- *FetchArt Plugin* and *EmbedArt Plugin*: Added a new `quality` option that controls the quality of the image output when the image is resized.
- *Key Finder Plugin*: Added support for keyfinder-cli. Thanks to BrainDamage.
- *FetchArt Plugin*: Added a new `high_resolution` config option to allow downloading of higher resolution iTunes artwork (at the expense of file size). #3391
- *Discogs Plugin*: The plugin applies two new fields: *discogs_labelid* and *discogs_artistid*. #3413
- *Export Plugin*: Added a new `-f` (`--format`) flag, which can export your data as JSON, JSON lines, CSV, or XML. Thanks to austinmm. #3402
- *Convert Plugin*: Added a new `-l` (`--link`) flag and `link` option as well as the `-H` (`--hardlink`) flag and `hardlink` option, which symlink or hardlink files that do not need to be converted (instead of copying them). #2324
- *ReplayGain Plugin*: The plugin now supports a `per_disc` option that enables calculation of album ReplayGain on disc level instead of album level. Thanks to samuelnilsson. #293
- *ReplayGain Plugin*: The new `ffmpeg` ReplayGain backend supports `R128_` tags. #3056
- *ReplayGain Plugin*: A new `r128_targetlevel` configuration option defines the reference volume for files using `R128_` tags. `targetlevel` only configures the reference volume for `REPLAYGAIN_` files. #3065
- *Discogs Plugin*: The plugin now collects the "style" field. Thanks to thedevilisinthedetails. #2579 #3251
- *AcousticBrainz Submit Plugin*: By default, the plugin now avoids re-analyzing files that already have AcousticBrainz data. There are new `force` and `pretend` options to help control this new behavior. Thanks to SusannaMaria. #3318
- *Discogs Plugin*: The plugin now also gets genre information and a new `discogs_albumid` field from the Discogs API. Thanks to thedevilisinthedetails. #465 #3322
- *AcousticBrainz Plugin*: The plugin now fetches two more additional fields: `moods_mirex` and `timbre`. Thanks to malcops. #2860
- *Playlist Plugin* and *Smart Playlist Plugin*: A new `forward_slash` config option facilitates compatibility with MPD on Windows. Thanks to MartyLake. #3331 #3334
- The *data_source* field, which indicates which metadata source was used during an autotagging import, is now also applied as an album-level flexible attribute. #3350 #1693
- *Beatport Plugin*: The plugin now gets the musical key, BPM, and genre for each track. #2080
- A new *BPSync Plugin* can synchronize metadata changes from the Beatport database (like the existing *MBSync Plugin* for MusicBrainz).

- *Hook Plugin*: The plugin now treats non-zero exit codes as errors. #3409
- *SubsonicUpdate Plugin*: A new `url` configuration replaces the older (and now deprecated) separate `host`, `port`, and `contextpath` config options. As a consequence, the plugin can now talk to Subsonic over HTTPS. Thanks to jef. #3449
- *Discogs Plugin*: The new `index_tracks` option enables incorporation of work names and intra-work divisions into imported track titles. Thanks to cole-miller. #3459
- *Web Plugin*: The query API now interprets backslashes as path separators to support path queries. Thanks to nmeum. #3567
- `beet import` now handles tar archives with bzip2 or gzip compression. #3606
- `beet import` *also* now handles 7z archives, via the py7zr library. Thanks to arogl. #3906
- *PlexUpdate Plugin*: Added an option to use a secure connection to Plex server, and to ignore certificate validation errors if necessary. #2871
- *Convert Plugin*: A new `delete_originals` configuration option can delete the source files after conversion during import. Thanks to logan-arens. #2947
- There is a new `--plugins` (or `-p`) CLI flag to specify a list of plugins to load.
- A new *genres* option fetches genre information from MusicBrainz. This functionality depends on functionality that is currently unreleased in the python-musicbrainzngs library: see PR #266. Thanks to aereaux.
- *ReplayGain Plugin*: Analysis now happens in parallel using the `command` and `ffmpeg` backends. #3478
- *ReplayGain Plugin*: The bs1770gain backend is removed. Thanks to SamuelCook.
- Added `trackdisambig` which stores the recording disambiguation from MusicBrainz for each track. #1904
- *FetchArt Plugin*: The new `max_filesize` configuration sets a maximum target image file size.
- *Bad Files Plugin*: Checkers can now run during import with the `check_on_import` config option.
- *Export Plugin*: The plugin is now much faster when using the *–include-keys* option is used. Thanks to ssssam.
- The importer's *set_fields* option now saves all updated fields to on-disk metadata. #3925 #3927
- We now fetch ISRC identifiers from MusicBrainz. Thanks to aereaux.
- *MetaSync Plugin*: The plugin now also fetches the "Date Added" field from iTunes databases and stores it in the `itunes_dateadded` field. Thanks to sandersantema.
- *Lyrics Plugin*: Added a new Tekstowo.pl lyrics provider. Thanks to various people for the implementation and for reporting issues with the initial version. #3344 #3904 #3905 #3994
- `beet update` will now confirm that the user still wants to update if their library folder cannot be found, preventing the user from accidentally wiping out their beets database. Thanks to user: *logan-arens*. #1934

Fixes:

- Adapt to breaking changes in Python's `ast` module in Python 3.8.
- *Beatport Plugin*: Fix the assignment of the *genre* field, and rename *musical_key* to *initial_key*. #3387
- *Lyrics Plugin*: Fixed the Musixmatch backend for lyrics pages when lyrics are divided into multiple elements on the webpage, and when the lyrics are missing.
- *Web Plugin*: Allow use of the backslash character in regex queries. #3867
- *Web Plugin*: Fixed a small bug that caused the album art path to be redacted even when `include_paths` option is set. #3866

- *Discogs Plugin*: Fixed a bug with the `index_tracks` option that sometimes caused the index to be discarded. Also, remove the extra semicolon that was added when there is no index track.
- *SubsonicUpdate Plugin*: The API client was using the *POST* method rather the *GET* method. Also includes better exception handling, response parsing, and tests.
- *The Plugin*: Fixed incorrect regex for "the" that matched any 3-letter combination of the letters t, h, e. #3701
- *FetchArt Plugin*: Fixed a bug that caused the plugin to not take environment variables, such as proxy servers, into account when making requests. #3450
- *FetchArt Plugin*: Temporary files for fetched album art that fail validation are now removed.
- *Inline Plugin*: In function-style field definitions that refer to flexible attributes, values could stick around from one function invocation to the next. This meant that, when displaying a list of objects, later objects could seem to reuse values from earlier objects when they were missing a value for a given field. These values are now properly undefined. #2406
- *BPD Plugin*: Seeking by fractions of a second now works as intended, fixing crashes in MPD clients like mpDris2 on seek. The `playlistid` command now works properly in its zero-argument form. #3214
- *ReplayGain Plugin*: Fix a Python 3 incompatibility in the Python Audio Tools backend. #3305
- *ImportAdded Plugin*: Fixed a crash that occurred when the `after_write` signal was emitted. #3301
- *ReplayGain Plugin*: Fix the storage format for R128 gain tags. #3311 #3314
- *Discogs Plugin*: Fixed a crash that occurred when the master URI isn't set in the API response. #2965 #3239
- *Spotify Plugin*: Fix handling of year-only release dates returned by the Spotify albums API. Thanks to rhlahuja. #3343
- Fixed a bug that caused the UI to display incorrect track numbers for tracks with index 0 when the `per_disc_numbering` option was set. #3346
- `none_rec_action` does not import automatically when `timid` is enabled. Thanks to RollingStar. #3242
- Fix a bug that caused a crash when tagging items with the beatport plugin. #3374
- `beet import` now logs which files are ignored when in debug mode. #3764
- *BPD Plugin*: Fix the transition to next track when in consume mode. Thanks to aereaux. #3437
- *Lyrics Plugin*: Fix a corner-case with Genius lowercase artist names #3446
- *ParentWork Plugin*: Don't save tracks when nothing has changed. #3492
- Added a warning when configuration files defined in the *include* directive of the configuration file fail to be imported. #3498
- Added normalization to integer values in the database, which should avoid problems where fields like `bpm` would sometimes store non-integer values. #762 #3507 #3508
- Fix a crash when querying for null values. #3516 #3517
- *Lyrics Plugin*: Tolerate a missing lyrics div in the Genius scraper. Thanks to thejli21. #3535 #3554
- *Lyrics Plugin*: Use the artist sort name to search for lyrics, which can help find matches when the artist name has special characters. Thanks to hashhar. #3340 #3558
- *ReplayGain Plugin*: Trying to calculate volume gain for an album consisting of some formats using `ReplayGain` and some using `R128` will no longer crash; instead it is skipped and and a message is logged. The log message has also been rewritten for to improve clarity. Thanks to autrimpo. #3533
- *Lyrics Plugin*: Adapt the Genius backend to changes in markup to reduce the scraping failure rate. #3535 #3594
- *Lyrics Plugin*: Fix a crash when writing ReST files for a query without results or fetched lyrics. #2805

- *FetchArt Plugin*: Attempt to fetch pre-resized thumbnails from Cover Art Archive if the `maxwidth` option matches one of the sizes supported by the Cover Art Archive API. Thanks to trolley. #3637
- *IPFS Plugin*: Fix Python 3 compatibility. Thanks to musoke. #2554
- Fix a bug that caused metadata starting with something resembling a drive letter to be incorrectly split into an extra directory after the colon. #3685
- *MPDStats Plugin*: Don't record a skip when stopping MPD, as MPD keeps the current track in the queue. Thanks to aereaux. #3722
- String-typed fields are now normalized to string values, avoiding an occasional crash when using both the *FetchArt Plugin* and the *Discogs Plugin* together. #3773 #3774
- Fix a bug causing PIL to generate poor quality JPEGs when resizing artwork. #3743
- *Key Finder Plugin*: Catch output from `keyfinder-cli` that is missing key. #2242
- *ReplayGain Plugin*: Disable parallel analysis on import by default. #3819
- *MPDStats Plugin*: Fix Python 2/3 compatibility #3798
- *Discogs Plugin*: Replace the deprecated official *discogs-client* library with the community supported python3-discogs-client library. #3608
- *Chromaprint/Acoustid Plugin*: Fixed submitting AcoustID information for tracks that already have a fingerprint. #3834
- Allow equals within the value part of the `--set` option to the `beet import` command. #2984
- Duplicates can now generate checksums. Thanks wisp3rwind for the pointer to how to solve. Thanks to arogl. #2873
- Templates that use `%ifdef` now produce the expected behavior when used in conjunction with non-string fields from the *Types Plugin*. #3852
- *Lyrics Plugin*: Fix crashes when a website could not be retrieved, affecting at least the Genius source. #3970
- *Duplicates Plugin*: Fix a crash when running the `dup` command with a query that returns no results. #3943
- *Beatport Plugin*: Fix the default assignment of the musical key. #3377
- *Lyrics Plugin*: Improved searching on the Genius backend when the artist contains special characters. #3634
- *ParentWork Plugin*: Also get the composition date of the parent work, instead of just the child work. Thanks to aereaux. #3650
- *Lyrics Plugin*: Fix a bug in the heuristic for detecting valid lyrics in the Google source. #2969
- *Thumbnails Plugin*: Fix a crash due to an incorrect string type on Python 3. #3360
- *FetchArt Plugin*: The Cover Art Archive source now iterates over all front images instead of blindly selecting the first one.
- *Lyrics Plugin*: Removed the LyricWiki source (the site shut down on 21/09/2020).
- *SubsonicUpdate Plugin*: The plugin is now functional again. A new *auth* configuration option is required in the configuration to specify the flavor of authentication to use. #4002

For plugin developers:

- MediaFile has been split into a standalone project. Where you used to do `from beets import mediafile`, now just do `import mediafile`. Beets re-exports MediaFile at the old location for backwards-compatibility, but a deprecation warning is raised if you do this since we might drop this wrapper in a future release.

- Similarly, we've replaced beets' configuration library (previously called Confit) with a standalone version called Confuse. Where you used to do `from beets.util import confit`, now just do `import confuse`. The code is almost identical apart from the name change. Again, we'll re-export at the old location (with a deprecation warning) for backwards compatibility, but we might stop doing this in a future release.
- `beets.util.command_output` now returns a named tuple containing both the standard output and the standard error data instead of just stdout alone. Client code will need to access the `stdout` attribute on the return value. Thanks to zsinskri. #3329
- There were sporadic failures in `test.test_player`. Hopefully these are fixed. If they resurface, please reopen the relevant issue. #3309 #3330
- The `beets.plugins.MetadataSourcePlugin` base class has been added to simplify development of plugins which query album, track, and search APIs to provide metadata matches for the importer. Refer to the *Spotify Plugin* and the *Deezer Plugin* for examples of using this template class. #3355
- Accessing fields on an *Item* now falls back to the album's attributes. So, for example, `item.foo` will first look for a field *foo* on *item* and, if it doesn't exist, next tries looking for a field named *foo* on the album that contains *item*. If you specifically want to access an item's attributes, use `Item.get(key, with_album=False)`. #2988
- `Item.keys` also has a `with_album` argument now, defaulting to `True`.
- A `revision` attribute has been added to `Database`. It is increased on every transaction that mutates it. #2988
- The classes `AlbumInfo` and `TrackInfo` now convey arbitrary attributes instead of a fixed, built-in set of field names (which was important to address #1547). Thanks to dosoe.
- Two new events, `mb_album_extract` and `mb_track_extract`, let plugins add new fields based on MusicBrainz data. Thanks to dosoe.

For packagers:

- Beets' library for manipulating media file metadata has now been split to a standalone project called MediaFile, released as mediafile. Beets now depends on this new package. Beets now depends on Mutagen transitively through MediaFile rather than directly, except in the case of one of beets' plugins (in particular, the *Scrub Plugin*).
- Beets' library for configuration has been split into a standalone project called Confuse, released as confuse. Beets now depends on this package. Confuse has existed separately for some time and is used by unrelated projects, but until now we've been bundling a copy within beets.
- We attempted to fix an unreliable test, so a patch to skip or repair the test may no longer be necessary.
- This version drops support for Python 3.4.
- We have removed an optional dependency on bs1770gain.

1.7.3 1.4.9 (May 30, 2019)

This small update is part of our attempt to release new versions more often! There are a few important fixes, and we're clearing the deck for a change to beets' dependencies in the next version.

The new feature is:

- You can use the NO_COLOR environment variable to disable terminal colors. #3273

There are some fixes in this release:

- Fix a regression in the last release that made the image resizer fail to detect older versions of ImageMagick. #3269

- *Gmusic Plugin*: The `oauth_file` config option now supports more flexible path values, including ~ for the home directory. #3270
- *Gmusic Plugin*: Fix a crash when using version 12.0.0 or later of the `gmusicapi` module. #3270
- Fix an incompatibility with Python 3.8's AST changes. #3278

Here's a note for packagers:

- `pathlib` is now an optional test dependency on Python 3.4+, removing the need for a Debian patch. #3275

1.7.4 1.4.8 (May 16, 2019)

This release is far too long in coming, but it's a good one. There is the usual torrent of new features and a ridiculously long line of fixes, but there are also some crucial maintenance changes. We officially support Python 3.7 and 3.8, and some performance optimizations can (anecdotally) make listing your library more than three times faster than in the previous version.

The new core features are:

- A new *aunique* configuration option allows setting default options for the *Album Disambiguation* template function.
- The `albumdisambig` field no longer includes the MusicBrainz release group disambiguation comment. A new `releasegroupdisambig` field has been added. #3024
- The *modify* command now allows resetting fixed attributes. For example, `beet modify -a artist:beatles artpath!` resets `artpath` attribute from matching albums back to the default value. #2497
- A new importer option, *ignore_data_tracks*, lets you skip audio tracks contained in data files. #3021

There are some new plugins:

- The *Playlist Plugin* can query the beets library using M3U playlists. Thanks to Holzhaus and Xenopathic. #123 #3145
- The *Load Extension Plugin* allows loading of SQLite extensions, primarily for use with the ICU SQLite extension for internationalization. #3160 #3226
- The *SubsonicUpdate Plugin* can automatically update your Subsonic library. Thanks to maffo999. #3001

And many improvements to existing plugins:

- *LastGenre Plugin*: Added option `-A` to match individual tracks and singletons. #3220 #3219
- *Play Plugin*: The plugin can now emit a UTF-8 BOM, fixing some issues with foobar2000 and Winamp. Thanks to mz2212. #2944
- ***Gmusic Plugin*:**
  - Add a new option to automatically upload to Google Play Music library on track import. Thanks to shuaiscott.
  - Add new options for Google Play Music authentication. Thanks to thetarkus. #3002
- *ReplayGain Plugin*: `albumpeak` on large collections is calculated as the average, not the maximum. #3008 #3009
- ***Chromaprint/Acoustid Plugin*:**
  - Now optionally has a bias toward looking up more relevant releases according to the *preferred* configuration options. Thanks to archer4499. #3017

  - Fingerprint values are now properly stored as strings, which prevents strange repeated output when running `beet write`. Thanks to Holzhaus. #3097 #2942

- *Convert Plugin*: The plugin now has an `id3v23` option that allows you to override the global `id3v23` option. Thanks to Holzhaus. #3104
- ***Spotify Plugin*:**
  - The plugin now uses OAuth for authentication to the Spotify API. Thanks to rhlahuja. #2694 #3123
  - The plugin now works as an import metadata provider: you can match tracks and albums using the Spotify database. Thanks to rhlahuja. #3123
- *IPFS Plugin*: The plugin now supports a `nocopy` option which passes that flag to ipfs. Thanks to wildthyme.
- *Discogs Plugin*: The plugin now has rate limiting for the Discogs API. #3081
- *MPDStats Plugin*, *MPDUpdate Plugin*: These plugins now use the `MPD_PORT` environment variable if no port is specified in the configuration file. #3223
- ***BPD Plugin*:**
  - MPD protocol commands `consume` and `single` are now supported along with updated semantics for `repeat` and `previous` and new fields for `status`. The bpd server now understands and ignores some additional commands. #3200 #800
  - MPD protocol command `idle` is now supported, allowing the MPD version to be bumped to 0.14. #3205 #800
  - MPD protocol command `decoders` is now supported. #3222
  - The plugin now uses the main beets logging system. The special-purpose `--debug` flag has been removed. Thanks to arcresu. #3196
- *MBSync Plugin*: The plugin no longer queries MusicBrainz when either the `mb_albumid` or `mb_trackid` field is invalid. See also the discussion on Google Groups Thanks to arogl.
- *Export Plugin*: The plugin now also exports `path` field if the user explicitly specifies it with `-i` parameter. This only works when exporting library fields. #3084
- *AcousticBrainz Plugin*: The plugin now declares types for all its fields, which enables easier querying and avoids a problem where very small numbers would be stored as strings. Thanks to rain0r. #2790 #3238

Some improvements have been focused on improving beets' performance:

- **Querying the library is now faster:**
  - We only convert fields that need to be displayed. Thanks to pprkut. #3089
  - We now compile templates once and reuse them instead of recompiling them to print out each matching object. Thanks to SimonPersson. #3258
  - Querying the library for items is now faster, for all queries that do not need to access album level properties. This was implemented by lazily fetching the album only when needed. Thanks to SimonPersson. #3260
- *AcousticBrainz Submit Plugin*, *Bad Files Plugin*: Analysis now works in parallel (on Python 3 only). Thanks to bemeurer. #2442 #3003
- *MPDStats Plugin*: Use the `currentsong` MPD command instead of `playlist` to get the current song, improving performance when the playlist is long. Thanks to ray66. #3207 #2752

Several improvements are related to usability:

- The disambiguation string for identifying albums in the importer now shows the catalog number. Thanks to 8h2a. #2951

- Added whitespace padding to missing tracks dialog to improve readability. Thanks to jams2. #2962
- The *move* command now lists the number of items already in-place. Thanks to RollingStar. #3117
- Modify selection can now be applied early without selecting every item. #3083
- Beets now emits more useful messages during startup if SQLite returns an error. The SQLite error message is now attached to the beets message. #3005
- Fixed a confusing typo when the *Convert Plugin* plugin copies the art covers. #3063

Many fixes have been focused on issues where beets would previously crash:

- Avoid a crash when archive extraction fails during import. #3041
- Missing album art file during an update no longer causes a fatal exception (instead, an error is logged and the missing file path is removed from the library). #3030
- When updating the database, beets no longer tries to move album art twice. #3189
- Fix an unhandled exception when pruning empty directories. #1996 #3209
- *FetchArt Plugin*: Added network connection error handling to backends so that beets won't crash if a request fails. Thanks to Holzhaus. #1579
- *Bad Files Plugin*: Avoid a crash when the underlying tool emits undecodable output. #3165
- *Beatport Plugin*: Avoid a crash when the server produces an error. #3184
- *BPD Plugin*: Fix crashes in the bpd server during exception handling. #3200
- *BPD Plugin*: Fix a crash triggered when certain clients tried to list the albums belonging to a particular artist. #3007 #3215
- *ReplayGain Plugin*: Avoid a crash when the `bs1770gain` tool emits malformed XML. #2983 #3247

There are many fixes related to compatibility with our dependencies including addressing changes interfaces:

- On Python 2, pin the jellyfish requirement to version 0.6.0 for compatibility.
- Fix compatibility with Python 3.7 and its change to a name in the re module. #2978
- Fix several uses of deprecated standard-library features on Python 3.7. Thanks to arcresu. #3197
- Fix compatibility with pre-release versions of Python 3.8. #3201 #3202
- *Web Plugin*: Fix an error when using more recent versions of Flask with CORS enabled. Thanks to rveachkc. #2979: #2980
- Avoid some deprecation warnings with certain versions of the MusicBrainz library. Thanks to zhelezov. #2826 #3092
- Restore iTunes Store album art source, and remove the dependency on python-itunes, which had gone unmaintained and was not Python-3-compatible. Thanks to ocelma for creating python-itunes in the first place. Thanks to nathdwek. #2371 #2551 #2718
- *LastGenre Plugin*, *Edit Plugin*: Avoid a deprecation warnings from the PyYAML library by switching to the safe loader. Thanks to translit and sbraz. #3192 #3225
- Fix a problem when resizing images with PIL/pillow on Python 3. Thanks to architek. #2504 #3029

And there are many other fixes:

- R128 normalization tags are now properly deleted from files when the values are missing. Thanks to autrimpo. #2757
- Display the artist credit when matching albums if the *artist_credit* configuration option is set. #2953

- With the *from_scratch* configuration option set, only writable fields are cleared. Beets now no longer ignores the format your music is saved in. #2972
- The `%aunique` template function now works correctly with the `-f/--format` option. #3043
- Fixed the ordering of items when manually selecting changes while updating tags Thanks to TaizoSimpson. #3501
- The `%title` template function now works correctly with apostrophes. Thanks to GuilhermeHideki. #3033
- *LastGenre Plugin*: It's now possible to set the `prefer_specific` option without also setting `canonical`. #2973
- *FetchArt Plugin*: The plugin now respects the `ignore` and `ignore_hidden` settings. #1632
- *Hook Plugin*: Fix byte string interpolation in hook commands. #2967 #3167
- *The Plugin*: Log a message when something has changed, not when it hasn't. Thanks to arcresu. #3195
- *LastGenre Plugin*: The `force` config option now actually works. #2704 #3054
- Resizing image files with ImageMagick now avoids problems on systems where there is a `convert` command that is *not* ImageMagick's by using the `magick` executable when it is available. Thanks to ababyduck. #2093 #3236

There is one new thing for plugin developers to know about:

- In addition to prefix-based field queries, plugins can now define *named queries* that are not associated with any specific field. For example, the new *Playlist Plugin* supports queries like `playlist:name` although there is no field named `playlist`. See *Extend the Query Syntax* for details.

And some messages for packagers:

- Note the changes to the dependencies on jellyfish and munkres.
- The optional python-itunes dependency has been removed.
- Python versions 3.7 and 3.8 are now supported.

1.7.5 1.4.7 (May 29, 2018)

This new release includes lots of new features in the importer and the metadata source backends that it uses. We've changed how the beets importer handles non-audio tracks listed in metadata sources like MusicBrainz:

- The importer now ignores non-audio tracks (namely, data and video tracks) listed in MusicBrainz. Also, a new option, *ignore_video_tracks*, lets you return to the old behavior and include these video tracks. #1210
- A new importer option, *ignored_media*, can let you skip certain media formats. #2688

There are other subtle improvements to metadata handling in the importer:

- In the MusicBrainz backend, beets now imports the `musicbrainz_releasetrackid` field. This is a first step toward #406. Thanks to Rawrmonkeys.
- A new importer configuration option, *artist_credit*, will tell beets to prefer the artist credit over the artist when autotagging. #1249

And there are even more new features:

- *ReplayGain Plugin*: The `beet replaygain` command now has `--force`, `--write` and `--nowrite` options. #2778
- A new importer configuration option, *incremental_skip_later*, lets you avoid recording skipped directories to the list of "processed" directories in *incremental* mode. This way, you can revisit them later with another import. Thanks to sekjun9878. #2773

- *FetchArt Plugin*: The configuration options now support finer-grained control via the `sources` option. You can now specify the search order for different *matching strategies* within different backends.
- *Web Plugin*: A new `cors_supports_credentials` configuration option lets in-browser clients communicate with the server even when it is protected by an authorization mechanism (a proxy with HTTP authentication enabled, for example).
- A new *SonosUpdate Plugin* plugin automatically notifies Sonos controllers to update the music library when the beets library changes. Thanks to cgtobi.
- *Discogs Plugin*: The plugin now stores master release IDs into `mb_releasegroupid`. It also "simulates" track IDs using the release ID and the track list position. Thanks to dbogdanov. #2336
- *Discogs Plugin*: Fetch the original year from master releases. #1122

There are lots and lots of fixes:

- *ReplayGain Plugin*: Fix a corner-case with the `bs1770gain` backend where ReplayGain values were assigned to the wrong files. The plugin now requires version 0.4.6 or later of the `bs1770gain` tool. #2777
- *Lyrics Plugin*: The plugin no longer crashes in the Genius source when BeautifulSoup is not found. Instead, it just logs a message and disables the source. #2911
- *Lyrics Plugin*: Handle network and API errors when communicating with Genius. #2771
- *Lyrics Plugin*: The `lyrics` command previously wrote ReST files by default, even when you didn't ask for them. This default has been fixed.
- *Lyrics Plugin*: When writing ReST files, the `lyrics` command now groups lyrics by the `albumartist` field, rather than `artist`. #2924
- Plugins can now see updated import task state, such as when rejecting the initial candidates and finding new ones via a manual search. Notably, this means that the importer prompt options that the *Edit Plugin* provides show up more reliably after doing a secondary import search. #2441 #2731
- *ImportAdded Plugin*: Fix a crash on non-autotagged imports. Thanks to m42i. #2601 #1918
- *PlexUpdate Plugin*: The Plex token is now redacted in configuration output. Thanks to Kovrinic. #2804
- Avoid a crash when importing a non-ASCII filename when using an ASCII locale on Unix under Python 3. #2793 #2803
- Fix a problem caused by time zone misalignment that could make date queries fail to match certain dates that are near the edges of a range. For example, querying for dates within a certain month would fail to match dates within hours of the end of that month. #2652
- *Convert Plugin*: The plugin now runs before other plugin-provided import stages, which addresses an issue with generating ReplayGain data incompatible between the source and target file formats. Thanks to autrimpo. #2814
- *FtInTitle Plugin*: The `drop` config option had no effect; it now does what it says it should do. #2817
- Importing a release with multiple release events now selects the event based on the order of your *preferred* countries rather than the order of release events in MusicBrainz. #2816
- *Web Plugin*: The time display in the web interface would incorrectly jump at the 30-second mark of every minute. Now, it correctly changes over at zero seconds. #2822
- *Web Plugin*: Fetching album art now works (instead of throwing an exception) under Python 3. Additionally, the server will now return a 404 response when the album ID is unknown (instead of throwing an exception and producing a 500 response). #2823
- *Web Plugin*: Fix an exception on Python 3 for filenames with non-Latin1 characters. (These characters are now converted to their ASCII equivalents.) #2815

- Partially fix bash completion for subcommand names that contain hyphens. Thanks to jhermann. #2836 #2837
- *ReplayGain Plugin*: Really fix album gain calculation using the GStreamer backend. #2846
- Avoid an error when doing a "no-op" move on non-existent files (i.e., moving a file onto itself). #2863
- *Discogs Plugin*: Fix the `medium` and `medium_index` values, which were occasionally incorrect for releases with two-sided mediums such as vinyl. Also fix the `medium_total` value, which now contains total number of tracks on the medium to which a track belongs, not the total number of different mediums present on the release. Thanks to dbogdanov. #2887
- The importer now supports audio files contained in data tracks when they are listed in MusicBrainz: the corresponding audio tracks are now merged into the main track list. Thanks to jdetrey. #1638
- *Key Finder Plugin*: Avoid a crash when trying to process unmatched tracks. #2537
- *MBSync Plugin*: Support MusicBrainz recording ID changes, relying on release track IDs instead. Thanks to jdetrey. #1234
- *MBSync Plugin*: We can now successfully update albums even when the first track has a missing MusicBrainz recording ID. #2920

There are a couple of changes for developers:

- Plugins can now run their import stages *early*, before other plugins. Use the `early_import_stages` list instead of plain `import_stages` to request this behavior. #2814
- We again properly send `albuminfo_received` and `trackinfo_received` in all cases, most notably when using the `mbsync` plugin. This was a regression since version 1.4.1. #2921

1.7.6 1.4.6 (December 21, 2017)

The highlight of this release is "album merging," an oft-requested option in the importer to add new tracks to an existing album you already have in your library. This way, you no longer need to resort to removing the partial album from your library, combining the files manually, and importing again.

Here are the larger new features in this release:

- When the importer finds duplicate albums, you can now merge all the tracks—old and new—together and try importing them as a single, combined album. Thanks to udiboy1209. #112 #2725
- *Lyrics Plugin*: The plugin can now produce reStructuredText files for beautiful, readable books of lyrics. Thanks to anarcat. #2628
- A new *from_scratch* configuration option makes the importer remove old metadata before applying new metadata. This new feature complements the *zero* and *scrub* plugins but is slightly different: beets clears out all the old tags it knows about and only keeps the new data it gets from the remote metadata source. Thanks to tummychow. #934 #2755

There are also somewhat littler, but still great, new features:

- *Convert Plugin*: A new `no_convert` option lets you skip transcoding items matching a query. Instead, the files are just copied as-is. Thanks to Stunner. #2732 #2751
- *FetchArt Plugin*: A new quiet switch that only prints out messages when album art is missing. Thanks to euri10. #2683
- *MusicBrainz Collection Plugin*: You can configure a custom MusicBrainz collection via the new `collection` configuration option. #2685
- *MusicBrainz Collection Plugin*: The collection update command can now remove albums from collections that are longer in the beets library.

- *FetchArt Plugin*: The `clearart` command now asks for confirmation before touching your files. Thanks to konman2. #2708 #2427
- *MPDStats Plugin*: The plugin now correctly updates song statistics when MPD switches from a song to a stream and when it plays the same song multiple times consecutively. #2707
- *AcousticBrainz Plugin*: The plugin can now be configured to write only a specific list of tags. Thanks to woparry.

There are lots and lots of bug fixes:

- *Hook Plugin*: Fixed a problem where accessing non-string properties of `item` or `album` (e.g., `item.track`) would cause a crash. Thanks to broddo. #2740
- *Play Plugin*: When `relative_to` is set, the plugin correctly emits relative paths even when querying for albums rather than tracks. Thanks to j000. #2702
- We suppress a spurious Python warning about a `BrokenPipeError` being ignored. This was an issue when using beets in simple shell scripts. Thanks to Azphreal. #2622 #2631
- *ReplayGain Plugin*: Fix a regression in the previous release related to the new R128 tags. #2615 #2623
- *Lyrics Plugin*: The MusixMatch backend now detects and warns when the server has blocked the client. Thanks to anarcat. #2634 #2632
- *ImportFeeds Plugin*: Fix an error on Python 3 in certain configurations. Thanks to djl. #2467 #2658
- *Edit Plugin*: Fix a bug when editing items during a re-import with the `-L` flag. Previously, diffs against against unrelated items could be shown or beets could crash. #2659
- *KodiUpdate Plugin*: Fix the server URL and add better error reporting. #2662
- Fixed a problem where "no-op" modifications would reset files' mtimes, resulting in unnecessary writes. This most prominently affected the *Edit Plugin* when saving the text file without making changes to some music. #2667
- *Chromaprint/Acoustid Plugin*: Fix a crash when running the `submit` command on Python 3 on Windows with non-ASCII filenames. #2671
- *AcousticBrainz Submit Plugin*: Fix an occasional crash on Python 3 when the AB analysis tool produced non-ASCII metadata. #2673
- *Duplicates Plugin*: Use the default tiebreak for items or albums when the configuration only specifies a tiebreak for the other kind of entity. Thanks to cgevans. #2758
- *Duplicates Plugin*: Fix the `--key` command line option, which was ignored.
- *ReplayGain Plugin*: Fix album ReplayGain calculation with the GStreamer backend. #2636
- *Scrub Plugin*: Handle errors when manipulating files using newer versions of Mutagen. #2716
- *FetchArt Plugin*: The plugin no longer gets skipped during import when the "Edit Candidates" option is used from the *Edit Plugin*. #2734
- Fix a crash when numeric metadata fields contain just a minus or plus sign with no following numbers. Thanks to eigengrau. #2741
- *FromFilename Plugin*: Recognize file names that contain *only* a track number, such as *01.mp3*. Also, the plugin now allows underscores as a separator between fields. Thanks to Vrihub. #2738 #2759
- Fixed an issue where images would be resized according to their longest edge, instead of their width, when using the `maxwidth` config option in the *FetchArt Plugin* and *EmbedArt Plugin*. Thanks to sekjun9878. #2729

There are some changes for developers:

- "Fixed fields" in Album and Item objects are now more strict about translating missing values into type-specific null-like values. This should help in cases where a string field is unexpectedly *None* sometimes instead of just showing up as an empty string. #2605
- Refactored the move functions the *beets.library* module and the *manipulate_files* function in *beets.importer* to use a single parameter describing the file operation instead of multiple Boolean flags. There is a new numerated type describing how to move, copy, or link files. #2682

1.7.7 1.4.5 (June 20, 2017)

Version 1.4.5 adds some oft-requested features. When you're importing files, you can now manually set fields on the new music. Date queries have gotten much more powerful: you can write precise queries down to the second, and we now have *relative* queries like `-1w`, which means *one week ago*.

Here are the new features:

- You can now set fields to certain values during *import*, using either a `--set field=value` command-line flag or a new *set_fields* configuration option under the *importer* section. Thanks to bartkl. #1881 #2581
- *Date queries* can now include times, so you can filter your music down to the second. Thanks to discopatrick. #2506 #2528
- *Date queries* can also be *relative*. You can say `added:-1w..` to match music added in the last week, for example. Thanks to euri10. #2598
- A new *Gmusic Plugin* lets you interact with your Google Play Music library. Thanks to tigranl. #2553 #2586
- *ReplayGain Plugin*: We now keep R128 data in separate tags from classic ReplayGain data for formats that need it (namely, Ogg Opus). A new *r128* configuration option enables this behavior for specific formats. Thanks to autrimpo. #2557 #2560
- The *move* command gained a new `--export` flag, which copies files to an external location without changing their paths in the library database. Thanks to SpirosChadoulos. #435 #2510

There are also some bug fixes:

- *LastGenre Plugin*: Fix a crash when using the *prefer_specific* and *canonical* options together. Thanks to yacoob. #2459 #2583
- *Web Plugin*: Fix a crash on Windows under Python 2 when serving non-ASCII filenames. Thanks to robot3498712. #2592 #2593
- *MetaSync Plugin*: Fix a crash in the Amarok backend when filenames contain quotes. Thanks to aranc23. #2595 #2596
- More informative error messages are displayed when the file format is not recognized. #2599

1.7.8 1.4.4 (June 10, 2017)

This release built up a longer-than-normal list of nifty new features. We now support DSF audio files and the importer can hard-link your files, for example.

Here's a full list of new features:

- Added support for DSF files, once a future version of Mutagen is released that supports them. Thanks to docbobo. #459 #2379
- A new *hardlink* config option instructs the importer to create hard links on filesystems that support them. Thanks to jacobwgillespie. #2445
- A new *KodiUpdate Plugin* lets you keep your Kodi library in sync with beets. Thanks to Pauligrinder. #2411

- A new *bell* configuration option under the `import` section enables a terminal bell when input is required. Thanks to SpirosChadoulos. #2366 #2495
- A new field, `composer_sort`, is now supported and fetched from MusicBrainz. Thanks to dosoe. #2519 #2529
- The MusicBrainz backend and *Discogs Plugin* now both provide a new attribute called `track_alt` that stores more nuanced, possibly non-numeric track index data. For example, some vinyl or tape media will report the side of the record using a letter instead of a number in that field. #1831 #2363
- *Web Plugin*: Added a new endpoint, `/item/path/foo`, which will return the item info for the file at the given path, or 404.
- *Web Plugin*: Added a new config option, `include_paths`, which will cause paths to be included in item API responses if set to true.
- The `%aunique` template function for *Album Disambiguation* now takes a third argument that specifies which brackets to use around the disambiguator value. The argument can be any two characters that represent the left and right brackets. It defaults to *[]* and can also be blank to turn off bracketing. #2397 #2399
- Added a `--move` or `-m` option to the importer so that the files can be moved to the library instead of being copied or added "in place." #2252 #2429
- *Bad Files Plugin*: Added a `--verbose` or `-v` option. Results are now displayed only for corrupted files by default and for all the files when the verbose option is set. #1654 #2434
- *EmbedArt Plugin*: The explicit `embedart` command now asks for confirmation before embedding art into music files. Thanks to Stunner. #1999
- You can now run beets by typing *python -m beets*. #2453
- *Smart Playlist Plugin*: Different playlist specifications that generate identically-named playlist files no longer conflict; instead, the resulting lists of tracks are concatenated. #2468
- *Missing Plugin*: A new mode lets you see missing albums from artists you have in your library. Thanks to qlyoung. #2481
- *Web Plugin* : Add new *reverse_proxy* config option to allow serving the web plugins under a reverse proxy.
- Importing a release with multiple release events now selects the event based on your *preferred* countries. #2501
- *Play Plugin*: A new `-y` or `--yes` parameter lets you skip the warning message if you enqueue more items than the warning threshold usually allows.
- Fix a bug where commands which forked subprocesses would sometimes prevent further inputs. This bug mainly affected *Convert Plugin*. Thanks to jansol. #2488 #2524

There are also quite a few fixes:

- In the *replace* configuration option, we now replace a leading hyphen (-) with an underscore. #549 #2509
- *AcousticBrainz Submit Plugin*: We no longer filter audio files for specific formats—we will attempt the submission process for all formats. #2471
- *MPDUpdate Plugin*: Fix Python 3 compatibility. #2381
- *ReplayGain Plugin*: Fix Python 3 compatibility in the `bs1770gain` backend. #2382
- *BPD Plugin*: Report playback times as integers. #2394
- *MPDStats Plugin*: Fix Python 3 compatibility. The plugin also now requires version 0.4.2 or later of the `python-mpd2` library. #2405
- *MPDStats Plugin*: Improve handling of MPD status queries.
- *Bad Files Plugin*: Fix Python 3 compatibility.

- Fix some cases where album-level ReplayGain/SoundCheck metadata would be written to files incorrectly. #2426
- *Bad Files Plugin*: The command no longer bails out if the validator command is not found or exits with an error. #2430 #2433
- *Lyrics Plugin*: The Google search backend no longer crashes when the server responds with an error. #2437
- *Discogs Plugin*: You can now authenticate with Discogs using a personal access token. #2447
- Fix Python 3 compatibility when extracting rar archives in the importer. Thanks to Lompik. #2443 #2448
- *Duplicates Plugin*: Fix Python 3 compatibility when using the `copy` and `move` options. #2444
- *MusicBrainz Submit Plugin*: The tracks are now sorted properly. Thanks to awesomer. #2457
- *Thumbnails Plugin*: Fix a string-related crash on Python 3. #2466
- *Beatport Plugin*: More than just 10 songs are now fetched per album. #2469
- On Python 3, the *terminal_encoding* setting is respected again for output and printing will no longer crash on systems configured with a limited encoding.
- *Convert Plugin*: The default configuration uses FFmpeg's built-in AAC codec instead of faac. Thanks to jansol. #2484
- Fix the importer's detection of multi-disc albums when other subdirectories are present. #2493
- Invalid date queries now print an error message instead of being silently ignored. Thanks to discopatrick. #2513 #2517
- When the SQLite database stops being accessible, we now print a friendly error message. Thanks to Mary011196. #1676 #2508
- *Web Plugin*: Avoid a crash when sending binary data, such as Chromaprint fingerprints, in music attributes. #2542 #2532
- Fix a hang when parsing templates that end in newlines. #2562
- Fix a crash when reading non-ASCII characters in configuration files on Windows under Python 3. #2456 #2565 #2566

We removed backends from two metadata plugins because of bitrot:

- *Lyrics Plugin*: The Lyrics.com backend has been removed. (It stopped working because of changes to the site's URL structure.) #2548 #2549
- *FetchArt Plugin*: The documentation no longer recommends iTunes Store artwork lookup because the unmaintained python-itunes is broken. Want to adopt it? #2371 #1610

1.7.9 1.4.3 (January 9, 2017)

Happy new year! This new version includes a cornucopia of new features from contributors, including new tags related to classical music and a new *AcousticBrainz Submit Plugin* for performing acoustic analysis on your music. The *Random Plugin* has a new mode that lets you generate time-limited music—for example, you might generate a random playlist that lasts the perfect length for your walk to work. We also access as many Web services as possible over secure connections now—HTTPS everywhere!

The most visible new features are:

- We now support the composer, lyricist, and arranger tags. The MusicBrainz data source will fetch data for these fields when the next version of python-musicbrainzngs is released. Thanks to ibmibmibm. #506 #507 #1547 #2333

- A new *AcousticBrainz Submit Plugin* lets you run acoustic analysis software and upload the results for others to use. Thanks to inytar. #2253 #2342
- *Play Plugin*: The plugin now provides an importer prompt choice to play the music you're about to import. Thanks to diomekes. #2008 #2360
- We now use SSL to access Web services whenever possible. That includes MusicBrainz itself, several album art sources, some lyrics sources, and other servers. Thanks to tigranl. #2307
- *Random Plugin*: A new `--time` option lets you generate a random playlist that takes a given amount of time. Thanks to diomekes. #2305 #2322

Some smaller new features:

- *Zero Plugin*: A new `zero` command manually triggers the zero plugin. Thanks to SJoshBrown. #2274 #2329
- *AcousticBrainz Plugin*: The plugin will avoid re-downloading data for files that already have it by default. You can override this behavior using a new `force` option. Thanks to SusannaMaria. #2347 #2349
- *BPM Plugin*: The `import.write` configuration option now decides whether or not to write tracks after updating their BPM. #1992

And the fixes:

- *BPD Plugin*: Fix a crash on non-ASCII MPD commands. #2332
- *Scrub Plugin*: Avoid a crash when files cannot be read or written. #2351
- *Scrub Plugin*: The image type values on scrubbed files are preserved instead of being reset to "other." #2339
- *Web Plugin*: Fix a crash on Python 3 when serving files from the filesystem. #2353
- *Discogs Plugin*: Improve the handling of releases that contain subtracks. #2318
- *Discogs Plugin*: Fix a crash when a release does not contain format information, and increase robustness when other fields are missing. #2302
- *Lyrics Plugin*: The plugin now reports a beets-specific User-Agent header when requesting lyrics. #2357
- *EmbyUpdate Plugin*: The plugin now checks whether an API key or a password is provided in the configuration.
- *Play Plugin*: The misspelled configuration option `warning_treshold` is no longer supported.

For plugin developers: when providing new importer prompt choices (see *Append Prompt Choices*), you can now provide new candidates for the user to consider. For example, you might provide an alternative strategy for picking between the available alternatives or for looking up a release on MusicBrainz.

1.7.10 1.4.2 (December 16, 2016)

This is just a little bug fix release. With 1.4.2, we're also confident enough to recommend that anyone who's interested give Python 3 a try: bugs may still lurk, but we've deemed things safe enough for broad adoption. If you can, please install beets with `pip3` instead of `pip2` this time and let us know how it goes!

Here are the fixes:

- *Bad Files Plugin*: Fix a crash on non-ASCII filenames. #2299
- The `%asciify{}` path formatting function and the *asciify_paths* setting properly substitute path separators generated by converting some Unicode characters, such as ½ and ¢, into ASCII.
- *Convert Plugin*: Fix a logging-related crash when filenames contain curly braces. Thanks to kierdavis. #2323
- We've rolled back some changes to the included zsh completion script that were causing problems for some users. #2266

Also, we've removed some special handling for logging in the *Discogs Plugin* that we believe was unnecessary. If spurious log messages appear in this version, please let us know by filing a bug.

1.7.11 1.4.1 (November 25, 2016)

Version 1.4 has **alpha-level** Python 3 support. Thanks to the heroic efforts of jrobeson, beets should run both under Python 2.7, as before, and now under Python 3.4 and above. The support is still new: it undoubtedly contains bugs, so it may replace all your music with Limp Bizkit—but if you're brave and you have backups, please try installing on Python 3. Let us know how it goes.

If you package beets for distribution, here's what you'll want to know:

- This version of beets now depends on the six library.
- We also bumped our minimum required version of Mutagen to 1.33 (from 1.27).
- Please don't package beets as a Python 3 application *yet*, even though most things work under Python 3.4 and later.

This version also makes a few changes to the command-line interface and configuration that you may need to know about:

- *Duplicates Plugin*: The `duplicates` command no longer accepts multiple field arguments in the form `-k title albumartist album`. Each argument must be prefixed with `-k`, as in `-k title -k albumartist -k album`.
- The old top-level `colors` configuration option has been removed (the setting is now under `ui`).
- The deprecated `list_format_album` and `list_format_item` configuration options have been removed (see *format_album* and *format_item*).

The are a few new features:

- *MPDUpdate Plugin*, *MPDStats Plugin*: When the `host` option is not set, these plugins will now look for the `$MPD_HOST` environment variable before falling back to `localhost`. Thanks to tarruda. #2175
- *Web Plugin*: Added an `expand` option to show the items of an album. #2050
- *EmbyUpdate Plugin*: The plugin can now use an API key instead of a password to authenticate with Emby. #2045 #2117
- *AcousticBrainz Plugin*: The plugin now adds a `bpm` field.
- `beet --version` now includes the Python version used to run beets.
- *Path Formats* can now include unescaped commas (`,`) when they are not part of a function call. #2166 #2213
- The *update* command takes a new `-F` flag to specify the fields to update. Thanks to dangmai. #2229 #2231

And there are a few bug fixes too:

- *Convert Plugin*: The plugin no longer asks for confirmation if the query did not return anything to convert. #2260 #2262
- *EmbedArt Plugin*: The plugin now uses `jpg` as an extension rather than `jpeg`, to ensure consistency with the *FetchArt Plugin*. Thanks to tweitzel. #2254 #2255
- *EmbedArt Plugin*: The plugin now works for all jpeg files, including those that are only recognizable by their magic bytes. #1545 #2255
- *Web Plugin*: The JSON output is no longer pretty-printed (for a space savings). #2050
- *Permissions Plugin*: Fix a regression in the previous release where the plugin would always fail to set permissions (and log a warning). #2089

- *Beatport Plugin*: Use track numbers from Beatport (instead of determining them from the order of tracks) and set the *medium_index* value.
- With *per_disc_numbering* enabled, some metadata sources (notably, the *Beatport Plugin*) would not set the track number at all. This is fixed. #2085
- *Play Plugin*: Fix `$args` getting passed verbatim to the play command if it was set in the configuration but `-A` or `--args` was omitted.
- With *ignore_hidden* enabled, non-UTF-8 filenames would cause a crash. This is fixed. #2168
- *EmbyUpdate Plugin*: Fixes authentication header problem that caused a problem that it was not possible to get tokens from the Emby API.
- *Lyrics Plugin*: Some titles use a colon to separate the main title from a subtitle. To find more matches, the plugin now also searches for lyrics using the part part preceding the colon character. #2206
- Fix a crash when a query uses a date field and some items are missing that field. #1938
- *Discogs Plugin*: Subtracks are now detected and combined into a single track, two-sided mediums are treated as single discs, and tracks have `media`, `medium_total` and `medium` set correctly. #2222 #2228.
- *Missing Plugin*: `missing` is now treated as an integer, allowing the use of (for example) ranges in queries.
- *Smart Playlist Plugin*: Playlist names will be sanitized to ensure valid filenames. #2258
- The ID3 APIC tag now uses the Latin-1 encoding when possible instead of a Unicode encoding. This should increase compatibility with other software, especially with iTunes and when using ID3v2.3. Thanks to lazka. #899 #2264 #2270

The last release, 1.3.19, also erroneously reported its version as "1.3.18" when you typed `beet version`. This has been corrected.

1.7.12 1.3.19 (June 25, 2016)

This is primarily a bug fix release: it cleans up a couple of regressions that appeared in the last version. But it also features the triumphant return of the *Beatport Plugin* and a modernized *BPD Plugin*.

It's also the first version where beets passes all its tests on Windows! May this herald a new age of cross-platform reliability for beets.

New features:

- *Beatport Plugin*: This metadata source plugin has arisen from the dead! It now works with Beatport's new OAuth-based API. Thanks to jbaiter. #1989 #2067
- *BPD Plugin*: The plugin now uses the modern GStreamer 1.0 instead of the old 0.10. Thanks to philippbeckmann. #2057 #2062
- A new `--force` option for the *remove* command allows removal of items without prompting beforehand. #2042
- A new *duplicate_action* importer config option controls how duplicate albums or tracks treated in import task. #185

Some fixes for Windows:

- Queries are now detected as paths when they contain backslashes (in addition to forward slashes). This only applies on Windows.
- *EmbedArt Plugin*: Image similarity comparison with ImageMagick should now work on Windows.
- *FetchArt Plugin*: The plugin should work more reliably with non-ASCII paths.

And other fixes:

- *ReplayGain Plugin*: The `bs1770gain` backend now correctly calculates sample peak instead of true peak. This comes with a major speed increase. #2031
- *Lyrics Plugin*: Avoid a crash and a spurious warning introduced in the last version about a Google API key, which appeared even when you hadn't enabled the Google lyrics source.
- Fix a hard-coded path to `bash-completion` to work better with Homebrew installations. Thanks to bismark. #2038
- Fix a crash introduced in the previous version when the standard input was connected to a Unix pipe. #2041
- Fix a crash when specifying non-ASCII format strings on the command line with the `-f` option for many commands. #2063
- *FetchArt Plugin*: Determine the file extension for downloaded images based on the image's magic bytes. The plugin prints a warning if result is not consistent with the server-supplied `Content-Type` header. In previous versions, the plugin would use a `.jpg` extension for all images. #2053

1.7.13 1.3.18 (May 31, 2016)

This update adds a new *Hook Plugin* that lets you integrate beets with command-line tools and an *Export Plugin* that can dump data from the beets database as JSON. You can also automatically translate lyrics using a machine translation service.

The `echonest` plugin has been removed in this version because the API it used is shutting down. You might want to try the *AcousticBrainz Plugin* instead.

Some of the larger new features:

- The new *Hook Plugin* lets you execute commands in response to beets events.
- The new *Export Plugin* can export data from beets' database as JSON. Thanks to GuilhermeHideki.
- *Lyrics Plugin*: The plugin can now translate the fetched lyrics to your native language using the Bing translation API. Thanks to Kraymer.
- *FetchArt Plugin*: Album art can now be fetched from fanart.tv.

Smaller new things:

- There are two new functions available in templates: `%first` and `%ifdef`. See *Template Functions*.
- *Convert Plugin*: A new *album_art_maxwidth* setting lets you resize album art while copying it.
- *Convert Plugin*: The *extension* setting is now optional for conversion formats. By default, the extension is the same as the name of the configured format.
- *ImportAdded Plugin*: A new *preserve_write_mtimes* option lets you preserve mtime of files even when beets updates their metadata.
- *FetchArt Plugin*: The *enforce_ratio* option now lets you tolerate images that are *almost* square but differ slightly from an exact 1:1 aspect ratio.
- *FetchArt Plugin*: The plugin can now optionally save the artwork's source in an attribute in the database.
- The *terminal_encoding* configuration option can now also override the *input* encoding. (Previously, it only affected the encoding of the standard *output* stream.)
- A new *ignore_hidden* configuration option lets you ignore files that your OS marks as invisible.
- *Web Plugin*: A new *values* endpoint lets you get the distinct values of a field. Thanks to sumpfralle. #2010

Fixes:

- Fix a problem with the *stats* command in exact mode when filenames on Windows use non-ASCII characters. #1891
- Fix a crash when iTunes Sound Check tags contained invalid data. #1895
- *MusicBrainz Collection Plugin*: The plugin now redacts your MusicBrainz password in the `beet config` output. #1907
- *Scrub Plugin*: Fix an occasional problem where scrubbing on import could undo the *id3v23* setting. #1903
- *Lyrics Plugin*: Add compatibility with some changes to the LyricsWiki page markup. #1912 #1909
- *Lyrics Plugin*: Fix retrieval from Musixmatch by improving the way we guess the URL for lyrics on that service. #1880
- *Edit Plugin*: Fail gracefully when the configured text editor command can't be invoked. #1927
- *FetchArt Plugin*: Fix a crash in the Wikipedia backend on non-ASCII artist and album names. #1960
- *Convert Plugin*: Change the default *ogg* encoding quality from 2 to 3 (to fit the default from the *oggenc(1)* manpage). #1982
- *Convert Plugin*: The *never_convert_lossy_files* option now considers AIFF a lossless format. #2005
- *Web Plugin*: A proper 404 error, instead of an internal exception, is returned when missing album art is requested. Thanks to sumpfralle. #2011
- Tolerate more malformed floating-point numbers in metadata tags. #2014
- The *ignore* configuration option now includes the `lost+found` directory by default.
- *AcousticBrainz Plugin*: AcousticBrainz lookups are now done over HTTPS. Thanks to Freso. #2007

1.7.14 1.3.17 (February 7, 2016)

This release introduces one new plugin to fetch audio information from the AcousticBrainz project and another plugin to make it easier to submit your handcrafted metadata back to MusicBrainz. The importer also gained two oft-requested features: a way to skip the initial search process by specifying an ID ahead of time, and a way to *manually* provide metadata in the middle of the import process (via the *Edit Plugin*).

Also, as of this release, the beets project has some new Internet homes! Our new domain name is beets.io, and we have a shiny new GitHub organization: beetbox.

Here are the big new features:

- A new *AcousticBrainz Plugin* fetches acoustic-analysis information from the AcousticBrainz project. Thanks to opatel99, and thanks to Google Code-In! #1784
- A new *MusicBrainz Submit Plugin* lets you print music's current metadata in a format that the MusicBrainz data parser can understand. You can trigger it during an interactive import session. #1779
- A new `--search-id` importer option lets you manually specify IDs (i.e., MBIDs or Discogs IDs) for imported music. Doing this skips the initial candidate search, which can be important for huge albums where this initial lookup is slow. Also, the `enter Id` prompt choice now accepts several IDs, separated by spaces. #1808
- *Edit Plugin*: You can now edit metadata *on the fly* during the import process. The plugin provides two new interactive options: one to edit *your music's* metadata, and one to edit the *matched metadata* retrieved from MusicBrainz (or another data source). This feature is still in its early stages, so please send feedback if you find anything missing. #1846 #396

There are even more new features:

- *FetchArt Plugin*: The Google Images backend has been restored. It now requires an API key from Google. Thanks to lcharlick. #1778

- *Info Plugin*: A new option will print only fields' names and not their values. Thanks to GuilhermeHideki. #1812
- The *fields* command now displays flexible attributes. Thanks to GuilhermeHideki. #1818
- The *modify* command lets you interactively select which albums or items you want to change. #1843
- The *move* command gained a new `--timid` flag to print and confirm which files you want to move. #1843
- The *move* command no longer prints filenames for files that don't actually need to be moved. #1583

Fixes:

- *Play Plugin*: Fix a regression in the last version where there was no default command. #1793
- *LastImport Plugin*: The plugin now works again after being broken by some unannounced changes to the Last.fm API. #1574
- *Play Plugin*: Fixed a typo in a configuration option. The option is now `warning_threshold` instead of `warning_treshold`, but we kept the old name around for compatibility. Thanks to JesseWeinstein. #1802 #1803
- *Edit Plugin*: Editing metadata now moves files, when appropriate (like the *modify* command). #1804
- The *stats* command no longer crashes when files are missing or inaccessible. #1806
- *FetchArt Plugin*: Possibly fix a Unicode-related crash when using some versions of pyOpenSSL. #1805
- *ReplayGain Plugin*: Fix an intermittent crash with the GStreamer backend. #1855
- *LastImport Plugin*: The plugin now works with the beets API key by default. You can still provide a different key the configuration.
- *ReplayGain Plugin*: Fix a crash using the Python Audio Tools backend. #1873

1.7.15 1.3.16 (December 28, 2015)

The big news in this release is a new *interactive editor plugin*. It's really nifty: you can now change your music's metadata by making changes in a visual text editor, which can sometimes be far more efficient than the built-in *modify* command. No more carefully retyping the same artist name with slight capitalization changes.

This version also adds an oft-requested "not" operator to beets' queries, so you can exclude music from any operation. It also brings friendlier formatting (and querying!) of song durations.

The big new stuff:

- A new *Edit Plugin* lets you manually edit your music's metadata using your favorite text editor. #164 #1706
- Queries can now use "not" logic. Type a `^` before part of a query to *exclude* matching music from the results. For example, `beet list -a beatles ^album:1` will find all your albums by the Beatles except for their singles compilation, "1." See *Query Term Negation*. #819 #1728
- A new *EmbyUpdate Plugin* can trigger a library refresh on an Emby server when your beets database changes.
- Track length is now displayed as "M:SS" rather than a raw number of seconds. Queries on track length also accept this format: for example, `beet list length:5:30..` will find all your tracks that have a duration over 5 minutes and 30 seconds. You can turn off this new behavior using the `format_raw_length` configuration option. #1749

Smaller changes:

- Three commands, `modify`, `update`, and `mbsync`, would previously move files by default after changing their metadata. Now, these commands will only move files if you have the *copy* or *move* options enabled in your importer configuration. This way, if you configure the importer not to touch your filenames, other commands

will respect that decision by default too. Each command also sprouted a `--move` command-line option to override this default (in addition to the `--nomove` flag they already had). #1697

- A new configuration option, `va_name`, controls the album artist name for various-artists albums. The setting defaults to "Various Artists," the MusicBrainz standard. In order to match MusicBrainz, the *Discogs Plugin* also adopts the same setting.
- *Info Plugin*: The `info` command now accepts a `-f/--format` option for customizing how items are displayed, just like the built-in `list` command. #1737

Some changes for developers:

- Two new *plugin hooks*, `albuminfo_received` and `trackinfo_received`, let plugins intercept metadata as soon as it is received, before it is applied to music in the database. #872
- Plugins can now add options to the interactive importer prompts. See *Append Prompt Choices*. #1758

Fixes:

- *PlexUpdate Plugin*: Fix a crash when Plex libraries use non-ASCII collection names. #1649
- *Discogs Plugin*: Maybe fix a crash when using some versions of the `requests` library. #1656
- Fix a race in the importer when importing two albums with the same artist and name in quick succession. The importer would fail to detect them as duplicates, claiming that there were "empty albums" in the database even when there were not. #1652
- *LastGenre Plugin*: Clean up the reggae-related genres somewhat. Thanks to Freso. #1661
- The importer now correctly moves album art files when re-importing. #314
- *FetchArt Plugin*: In auto mode, the plugin now skips albums that already have art attached to them so as not to interfere with re-imports. #314
- *FetchArt Plugin*: The plugin now only resizes album art if necessary, rather than always by default. #1264
- *FetchArt Plugin*: Fix a bug where a database reference to a non-existent album art file would prevent the command from fetching new art. #1126
- *Thumbnails Plugin*: Fix a crash with Unicode paths. #1686
- *EmbedArt Plugin*: The `remove_art_file` option now works on import (as well as with the explicit command). #1662 #1675
- *MetaSync Plugin*: Fix a crash when syncing with recent versions of iTunes. #1700
- *Duplicates Plugin*: Fix a crash when merging items. #1699
- *Smart Playlist Plugin*: More gracefully handle malformed queries and missing configuration.
- Fix a crash with some files with unreadable iTunes SoundCheck metadata. #1666
- *Thumbnails Plugin*: Fix a nasty segmentation fault crash that arose with some library versions. #1433
- *Convert Plugin*: Fix a crash with Unicode paths in `--pretend` mode. #1735
- Fix a crash when sorting by nonexistent fields on queries. #1734
- Probably fix some mysterious errors when dealing with images using ImageMagick on Windows. #1721
- Fix a crash when writing some Unicode comment strings to MP3s that used older encodings. The encoding is now always updated to UTF-8. #879
- *FetchArt Plugin*: The Google Images backend has been removed. It used an API that has been shut down. #1760
- *Lyrics Plugin*: Fix a crash in the Google backend when searching for bands with regular-expression characters in their names, like Sunn O))). #1673

- *Scrub Plugin*: In `auto` mode, the plugin now *actually* only scrubs files on import, as the documentation always claimed it did—not every time files were written, as it previously did. #1657
- *Scrub Plugin*: Also in `auto` mode, album art is now correctly restored. #1657
- Possibly allow flexible attributes to be used with the `%aunique` template function. #1775
- *Lyrics Plugin*: The Genius backend is now more robust to communication errors. The backend has also been disabled by default, since the API it depends on is currently down. #1770

1.7.16 1.3.15 (October 17, 2015)

This release adds a new plugin for checking file quality and a new source for lyrics. The larger features are:

- A new *Bad Files Plugin* helps you scan for corruption in your music collection. Thanks to fxthomas. #1568
- *Lyrics Plugin*: You can now fetch lyrics from Genius.com. Thanks to sadatay. #1626 #1639
- *Zero Plugin*: The plugin can now use a "whitelist" policy as an alternative to the (default) "blacklist" mode. Thanks to adkow. #1621 #1641

And there are smaller new features too:

- Add new color aliases for standard terminal color names (e.g., cyan and magenta). Thanks to mathstuf. #1548
- *Play Plugin*: A new `--args` option lets you specify options for the player command. #1532
- *Play Plugin*: A new `raw` configuration option lets the command work with players (such as VLC) that expect music filenames as arguments, rather than in a playlist. Thanks to nathdwek. #1578
- *Play Plugin*: You can now configure the number of tracks that trigger a "lots of music" warning. #1577
- *EmbedArt Plugin*: A new `remove_art_file` option lets you clean up if you prefer *only* embedded album art. Thanks to jackwilsdon. #1591 #733
- *PlexUpdate Plugin*: A new `library_name` option allows you to select which Plex library to update. #1572 #1595
- A new `include` option lets you import external configuration files.

This release has plenty of fixes:

- *LastGenre Plugin*: Fix a bug that prevented tag popularity from being considered. Thanks to svoos. #1559
- Fixed a bug where plugins wouldn't be notified of the deletion of an item's art, for example with the `clearart` command from the *EmbedArt Plugin*. Thanks to nathdwek. #1565
- *FetchArt Plugin*: The Google Images source is disabled by default (as it was before beets 1.3.9), as is the Wikipedia source (which was causing lots of unnecessary delays due to DBpedia downtime). To re-enable these sources, add `wikipedia google` to your `sources` configuration option.
- The *list* command's help output now has a small query and format string example. Thanks to pkess. #1582
- *FetchArt Plugin*: The plugin now fetches PNGs but not GIFs. (It still fetches JPEGs.) This avoids an error when trying to embed images, since not all formats support GIFs. #1588
- Date fields are now written in the correct order (year-month-day), which eliminates an intermittent bug where the latter two fields would not get written to files. Thanks to jdetrey. #1303 #1589
- *ReplayGain Plugin*: Avoid a crash when the PyAudioTools backend encounters an error. #1592
- The case sensitivity of path queries is more useful now: rather than just guessing based on the platform, we now check the case sensitivity of your filesystem. #1586
- Case-insensitive path queries might have returned nothing because of a wrong SQL query.

- Fix a crash when a query contains a "+" or "-" alone in a component. #1605
- Fixed unit of file size to powers of two (MiB, GiB, etc.) instead of powers of ten (MB, GB, etc.). #1623

1.7.17 1.3.14 (August 2, 2015)

This is mainly a bugfix release, but we also have a nifty new plugin for ipfs and a bunch of new configuration options.

The new features:

- A new *IPFS Plugin* lets you share music via a new, global, decentralized filesystem. #1397
- *Duplicates Plugin*: You can now merge duplicate track metadata (when detecting duplicate items), or duplicate album tracks (when detecting duplicate albums).
- *Duplicates Plugin*: Duplicate resolution now uses an ordering to prioritize duplicates. By default, it prefers music with more complete metadata, but you can configure it to use any list of attributes.
- *MetaSync Plugin*: Added a new backend to fetch metadata from iTunes. This plugin is still in an experimental phase. #1450
- The *move* command has a new `--pretend` option, making the command show how the items will be moved without actually changing anything.
- The importer now supports matching of "pregap" or HTOA (hidden track-one audio) tracks when they are listed in MusicBrainz. (This feature depends on a new version of the python-musicbrainzngs library that is not yet released, but will start working when it is available.) Thanks to ruippeixotog. #1104 #1493
- *PlexUpdate Plugin*: A new `token` configuration option lets you specify a key for Plex Home setups. Thanks to edcarroll. #1494

Fixes:

- *FetchArt Plugin*: Complain when the *enforce_ratio* or *min_width* options are enabled but no local imaging backend is available to carry them out. #1460
- *ImportFeeds Plugin*: Avoid generating incorrect m3u filename when both of the *m3u* and *m3u_multi* options are enabled. #1490
- *Duplicates Plugin*: Avoid a crash when misconfigured. #1457
- *MPDStats Plugin*: Avoid a crash when the music played is not in the beets library. Thanks to CodyReichert. #1443
- Fix a crash with ArtResizer on Windows systems (affecting *EmbedArt Plugin*, *FetchArt Plugin*, and *Thumbnails Plugin*). #1448
- *Permissions Plugin*: Fix an error with non-ASCII paths. #1449
- Fix sorting by paths when the *sort_case_insensitive* option is enabled. #1451
- *EmbedArt Plugin*: Avoid an error when trying to embed invalid images into MPEG-4 files.
- *FetchArt Plugin*: The Wikipedia source can now better deal artists that use non-standard capitalization (e.g., alt-J, dEUS).
- *Web Plugin*: Fix searching for non-ASCII queries. Thanks to oldtopman. #1470
- *MPDUpdate Plugin*: We now recommend the newer `python-mpd2` library instead of its unmaintained parent. Thanks to Somasis. #1472
- The importer interface and log file now output a useful list of files (instead of the word "None") when in album-grouping mode. #1475 #825
- Fix some logging errors when filenames and other user-provided strings contain curly braces. #1481

- Regular expression queries over paths now work more reliably with non-ASCII characters in filenames. #1482
- Fix a bug where the autotagger's *ignored* setting was sometimes, well, ignored. #1487
- Fix a bug with Unicode strings when generating image thumbnails. #1485
- *Key Finder Plugin*: Fix handling of Unicode paths. #1502
- *FetchArt Plugin*: When album art is already present, the message is now printed in the `text_highlight_minor` color (light gray). Thanks to Somasis. #1512
- Some messages in the console UI now use plural nouns correctly. Thanks to JesseWeinstein. #1521
- Sorting numerical fields (such as track) now works again. #1511
- *ReplayGain Plugin*: Missing GStreamer plugins now cause a helpful error message instead of a crash. #1518
- Fix an edge case when producing sanitized filenames where the maximum path length conflicted with the *replace* rules. Thanks to Ben Ockmore. #496 #1361
- Fix an incompatibility with OS X 10.11 (where `/usr/sbin` seems not to be on the user's path by default).
- Fix an incompatibility with certain JPEG files. Here's a relevant Python bug. Thanks to nathdwek. #1545
- Fix the *group_albums* importer mode so that it works correctly when files are not already in order by album. #1550
- The `fields` command no longer separates built-in fields from plugin-provided ones. This distinction was becoming increasingly unreliable.
- *Duplicates Plugin*: Fix a Unicode warning when paths contained non-ASCII characters. #1551
- *FetchArt Plugin*: Work around a urllib3 bug that could cause a crash. #1555 #1556
- When you edit the configuration file with `beet config -e` and the file does not exist, beets creates an empty file before editing it. This fixes an error on OS X, where the `open` command does not work with non-existent files. #1480
- *Convert Plugin*: Fix a problem with filename encoding on Windows under Python 3. #2515 #2516

1.7.18 1.3.13 (April 24, 2015)

This is a tiny bug-fix release. It copes with a dependency upgrade that broke beets. There are just two fixes:

- Fix compatibility with Jellyfish version 0.5.0.
- *EmbedArt Plugin*: In `auto` mode (the import hook), the plugin now respects the `write` config option under `import`. If this is disabled, album art is no longer embedded on import in order to leave files untouched—in effect, `auto` is implicitly disabled. #1427

1.7.19 1.3.12 (April 18, 2015)

This little update makes queries more powerful, sorts music more intelligently, and removes a performance bottleneck. There's an experimental new plugin for synchronizing metadata with music players.

Packagers should also note a new dependency in this version: the Jellyfish Python library makes our text comparisons (a big part of the auto-tagging process) go much faster.

New features:

- Queries can now use **"or" logic**: if you use a comma to separate parts of a query, items and albums will match *either* side of the comma. For example, `beet ls foo , bar` will get all the items matching *foo* or matching *bar*. See *Combining Keywords*. #1423

- The autotagger's **matching algorithm is faster**. We now use the Jellyfish library to compute string similarity, which is better optimized than our hand-rolled edit distance implementation. #1389
- Sorting is now **case insensitive** by default. This means that artists will be sorted lexicographically regardless of case. For example, the artist alt-J will now properly sort before YACHT. (Previously, it would have ended up at the end of the list, after all the capital-letter artists.) You can turn this new behavior off using the *sort_case_insensitive* configuration option. See *Sort Order*. #1429
- An experimental new *MetaSync Plugin* lets you get metadata from your favorite music players, starting with Amarok. #1386
- *FetchArt Plugin*: There are new settings to control what constitutes "acceptable" images. The *minwidth* option constrains the minimum image width in pixels and the *enforce_ratio* option requires that images be square. #1394

Little fixes and improvements:

- *FetchArt Plugin*: Remove a hard size limit when fetching from the Cover Art Archive.
- The output of the *fields* command is now sorted. Thanks to multikatt. #1402
- *ReplayGain Plugin*: Fix a number of issues with the new `bs1770gain` backend on Windows. Also, fix missing debug output in import mode. #1398
- Beets should now be better at guessing the appropriate output encoding on Windows. (Specifically, the console output encoding is guessed separately from the encoding for command-line arguments.) A bug was also fixed where beets would ignore the locale settings and use UTF-8 by default. #1419
- *Discogs Plugin*: Better error handling when we can't communicate with Discogs on setup. #1417
- *ImportAdded Plugin*: Fix a crash when importing singletons in-place. #1416
- *Fuzzy Search Plugin*: Fix a regression causing a crash in the last release. #1422
- Fix a crash when the importer cannot open its log file. Thanks to barsanuphe. #1426
- Fix an error when trying to write tags for items with flexible fields called *date* and *original_date* (which are not built-in beets fields). #1404

1.7.20 1.3.11 (April 5, 2015)

In this release, we refactored the logging system to be more flexible and more useful. There are more granular levels of verbosity, the output from plugins should be more consistent, and several kinds of logging bugs should be impossible in the future.

There are also two new plugins: one for filtering the files you import and an evolved plugin for using album art as directory thumbnails in file managers. There's a new source for album art, and the importer now records the source of match data. This is a particularly huge release—there's lots more below.

There's one big change with this release: **Python 2.6 is no longer supported**. You'll need Python 2.7. Please trust us when we say this let us remove a surprising number of ugly hacks throughout the code.

Major new features and bigger changes:

- There are now **multiple levels of output verbosity**. On the command line, you can make beets somewhat verbose with `-v` or very verbose with `-vv`. For the importer especially, this makes the first verbose mode much more manageable, while still preserving an option for overwhelmingly verbose debug output. #1244
- A new *FileFilter Plugin* lets you write regular expressions to automatically **avoid importing** certain files. Thanks to mried. #1186
- A new *Thumbnails Plugin* generates cover-art **thumbnails for album folders** for Freedesktop.org-compliant file managers. (This replaces the *Freedesktop Plugin*, which only worked with the Dolphin file manager.)

- *ReplayGain Plugin*: There is a new backend that uses the bs1770gain analysis tool. Thanks to jmwatte. #1343
- A new `filesize` field on items indicates the number of bytes in the file. #1291
- A new *searchlimit* configuration option allows you to specify how many search results you wish to see when looking up releases at MusicBrainz during import. #1245
- The importer now records the data source for a match in a new flexible attribute `data_source` on items and albums. #1311
- The colors used in the terminal interface are now configurable via the new config option `colors`, nested under the option `ui`. (Also, the *color* config option has been moved from top-level to under `ui`. Beets will respect the old color setting, but will warn the user with a deprecation message.) #1238
- *FetchArt Plugin*: There's a new Wikipedia image source that uses DBpedia to find albums. Thanks to Tom Jaspers. #1194
- In the *config* command, the output is now redacted by default. Sensitive information like passwords and API keys is not included. The new `--clear` option disables redaction. #1376

You should probably also know about these core changes to the way beets works:

- As mentioned above, Python 2.6 is no longer supported.
- The `tracktotal` attribute is now a *track-level field* instead of an album-level one. This field stores the total number of tracks on the album, or if the *per_disc_numbering* config option is set, the total number of tracks on a particular medium (i.e., disc). The field was causing problems with that *per_disc_numbering* mode: different discs on the same album needed different track totals. The field can now work correctly in either mode.
- To replace `tracktotal` as an album-level field, there is a new `albumtotal` computed attribute that provides the total number of tracks on the album. (The *per_disc_numbering* option has no influence on this field.)
- The *list_format_album* and *list_format_item* configuration keys now affect (almost) every place where objects are printed and logged. (Previously, they only controlled the *list* command and a few other scattered pieces.) #1269
- Relatedly, the `beet` program now accept top-level options `--format-item` and `--format-album` before any subcommand to control how items and albums are displayed. #1271
- *list_format_album* and *list_format_album* have respectively been renamed *format_album* and *format_item*. The old names still work but each triggers a warning message. #1271
- *Path queries* are automatically triggered only if the path targeted by the query exists. Previously, just having a slash somewhere in the query was enough, so `beet ls AC/DC` wouldn't work to refer to the artist.

There are also lots of medium-sized features in this update:

- *Duplicates Plugin*: The command has a new `--strict` option that will only report duplicates if all attributes are explicitly set. #1000
- *Smart Playlist Plugin*: Playlist updating should now be faster: the plugin detects, for each playlist, whether it needs to be regenerated, instead of obliviously regenerating all of them. The `splupdate` command can now also take additional parameters that indicate the names of the playlists to regenerate.
- *Play Plugin*: The command shows the output of the underlying player command and lets you interact with it. #1321
- The summary shown to compare duplicate albums during import now displays the old and new filesizes. #1291
- *LastGenre Plugin*: Add *comedy*, *humor*, and *stand-up* as well as a longer list of classical music genre tags to the built-in whitelist and canonicalization tree. #1206 #1239 #1240
- *Web Plugin*: Add support for *cross-origin resource sharing* for more flexible in-browser clients. Thanks to Andre Miller. #1236 #1237

- *MBSync Plugin*: A new `-f/--format` option controls the output format when listing unrecognized items. The output is also now more helpful by default. #1246
- *FetchArt Plugin*: A new option, `-n`, extracts the cover art of all matched albums into their respective directories. Another new flag, `-a`, associates the extracted files with the albums in the database. #1261
- *Info Plugin*: A new option, `-i`, can display only a specified subset of properties. #1287
- The number of missing/unmatched tracks is shown during import. #1088
- *Permissions Plugin*: The plugin now also adjusts the permissions of the directories. (Previously, it only affected files.) #1308 #1324
- *FtInTitle Plugin*: You can now configure the format that the plugin uses to add the artist to the title. Thanks to amishb. #1377

And many little fixes and improvements:

- *ReplayGain Plugin*: Stop applying replaygain directly to source files when using the mp3gain backend. #1316
- Path queries are case-sensitive on non-Windows OSes. #1165
- *Lyrics Plugin*: Silence a warning about insecure requests in the new MusixMatch backend. #1204
- Fix a crash when `beet` is invoked without arguments. #1205 #1207
- *FetchArt Plugin*: Do not attempt to import directories as album art. #1177 #1211
- *MPDStats Plugin*: Avoid double-counting some play events. #773 #1212
- Fix a crash when the importer deals with Unicode metadata in `--pretend` mode. #1214
- *Smart Playlist Plugin*: Fix `album_query` so that individual files are added to the playlist instead of directories. #1225
- Remove the `beatport` plugin. Beatport has shut off public access to their API and denied our request for an account. We have not heard from the company since 2013, so we are assuming access will not be restored.
- Incremental imports now (once again) show a "skipped N directories" message.
- *EmbedArt Plugin*: Handle errors in ImageMagick's output. #1241
- *Key Finder Plugin*: Parse the underlying tool's output more robustly. #1248
- *EmbedArt Plugin*: We now show a comprehensible error message when `beet embedart -f FILE` is given a non-existent path. #1252
- Fix a crash when a file has an unrecognized image type tag. Thanks to Matthias Kiefer. #1260
- *ImportFeeds Plugin* and *Smart Playlist Plugin*: Automatically create parent directories for playlist files (instead of crashing when the parent directory does not exist). #1266
- The *write* command no longer tries to "write" non-writable fields, such as the bitrate. #1268
- The error message when MusicBrainz is not reachable on the network is now much clearer. Thanks to Tom Jaspers. #1190 #1272
- Improve error messages when parsing query strings with shlex. #1290
- *EmbedArt Plugin*: Fix a crash that occurred when used together with the *check* plugin. #1241
- *Scrub Plugin*: Log an error instead of stopping when the `beet scrub` command cannot write a file. Also, avoid problems on Windows with Unicode filenames. #1297
- *Discogs Plugin*: Handle and log more kinds of communication errors. #1299 #1305
- *LastGenre Plugin*: Bugs in the *pylast* library can no longer crash beets.

- *Convert Plugin*: You can now configure the temporary directory for conversions. Thanks to autochthe. #1382 #1383
- *Rewrite Plugin*: Fix a regression that prevented the plugin's rewriting from applying to album-level fields like `$albumartist`. #1393
- *Play Plugin*: The plugin now sorts items according to the configuration in album mode.
- *FetchArt Plugin*: The name for extracted art files is taken from the `art_filename` configuration option. #1258
- When there's a parse error in a query (for example, when you type a malformed date in a *date query*), beets now stops with an error instead of silently ignoring the query component.
- *Smart Playlist Plugin*: Stream-friendly smart playlists. The `splupdate` command can now also add a URL-encodable prefix to every path in the playlist file.

For developers:

- The `database_change` event now sends the item or album that is subject to a change.
- The `OptionParser` is now a `CommonOptionsParser` that offers facilities for adding usual options (`--album`, `--path` and `--format`). See *Add Commands to the CLI*. #1271
- The logging system in beets has been overhauled. Plugins now each have their own logger, which helps by automatically adjusting the verbosity level in import mode and by prefixing the plugin's name. Logging levels are dynamically set when a plugin is called, depending on how it is called (import stage, event or direct command). Finally, logging calls can (and should!) use modern `{}`-style string formatting lazily. See *Logging* in the plugin API docs.
- A new `import_task_created` event lets you manipulate import tasks immediately after they are initialized. It's also possible to replace the originally created tasks by returning new ones using this event.

1.7.21 1.3.10 (January 5, 2015)

This version adds a healthy helping of new features and fixes a critical MPEG-4–related bug. There are more lyrics sources, there new plugins for managing permissions and integrating with Plex, and the importer has a new `--pretend` flag that shows which music *would* be imported.

One backwards-compatibility note: the *Lyrics Plugin* now requires the requests library. If you use this plugin, you will need to install the library by typing `pip install requests` or the equivalent for your OS.

Also, as an advance warning, this will be one of the last releases to support Python 2.6. If you have a system that cannot run Python 2.7, please consider upgrading soon.

The new features are:

- A new *Permissions Plugin* makes it easy to fix permissions on music files as they are imported. Thanks to xsteadfastx. #1098
- A new *PlexUpdate Plugin* lets you notify a Plex server when the database changes. Thanks again to xsteadfastx. #1120
- The *import* command now has a `--pretend` flag that lists the files that will be imported. Thanks to mried. #1162
- *Lyrics Plugin*: Add Musixmatch source and introduce a new `sources` config option that lets you choose exactly where to look for lyrics and in which order.
- *Lyrics Plugin*: Add Brazilian and Spanish sources to Google custom search engine.
- Add a warning when importing a directory that contains no music. #1116 #1127

- *Zero Plugin*: Can now remove embedded images. #1129 #1100
- The *config* command can now be used to edit the configuration even when it has syntax errors. #1123 #1128
- *Lyrics Plugin*: Added a new `force` config option. #1150

As usual, there are loads of little fixes and improvements:

- Fix a new crash with the latest version of Mutagen (1.26).
- *Lyrics Plugin*: Avoid fetching truncated lyrics from the Google backed by merging text blocks separated by empty `<div>` tags before scraping.
- We now print a better error message when the database file is corrupted.
- *Discogs Plugin*: Only prompt for authentication when running the *import* command. #1123
- When deleting fields with the *modify* command, do not crash when the field cannot be removed (i.e., when it does not exist, when it is a built-in field, or when it is a computed field). #1124
- The deprecated `echonest_tempo` plugin has been removed. Please use the `echonest` plugin instead.
- `echonest` plugin: Fingerprint-based lookup has been removed in accordance with API changes. #1121
- `echonest` plugin: Avoid a crash when the song has no duration information. #896
- *Lyrics Plugin*: Avoid a crash when retrieving non-ASCII lyrics from the Google backend. #1135 #1136
- *Smart Playlist Plugin*: Sort specifiers are now respected in queries. Thanks to djl. #1138 #1137
- *FtInTitle Plugin* and *Lyrics Plugin*: Featuring artists can now be detected when they use the Spanish word *con*. #1060 #1143
- *MusicBrainz Collection Plugin*: Fix an "HTTP 400" error caused by a change in the MusicBrainz API. #1152
- The `%` and _ characters in path queries do not invoke their special SQL meaning anymore. #1146
- *Convert Plugin*: Command-line argument construction now works on Windows. Thanks to mluds. #1026 #1157 #1158
- *EmbedArt Plugin*: Fix an erroneous missing-art error on Windows. Thanks to mluds. #1163
- *ImportAdded Plugin*: Now works with in-place and symlinked imports. #1170
- *FtInTitle Plugin*: The plugin is now quiet when it runs as part of the import process. Thanks to Freso. #1176 #1172
- *FtInTitle Plugin*: Fix weird behavior when the same artist appears twice in the artist string. Thanks to Marc Addeo. #1179 #1181
- *LastGenre Plugin*: Match songs more robustly when they contain dashes. Thanks to djl. #1156
- The *config* command can now use `$EDITOR` variables with arguments.

1.7.22 1.3.9 (November 17, 2014)

This release adds two new standard plugins to beets: one for synchronizing Last.fm listening data and one for integrating with Linux desktops. And at long last, imports can now create symbolic links to music files instead of copying or moving them. We also gained the ability to search for album art on the iTunes Store and a new way to compute ReplayGain levels.

The major new features are:

- A new *LastImport Plugin* lets you download your play count data from Last.fm into a flexible attribute. Thanks to Rafael Bodill.

- A new *Freedesktop Plugin* creates metadata files for Freedesktop.org–compliant file managers. Thanks to kerobaros. #1056, #707
- A new *link* option in the `import` section creates symbolic links during import instead of moving or copying. Thanks to Rovanion Luckey. #710, #114
- *FetchArt Plugin*: You can now search for art on the iTunes Store. There's also a new `sources` config option that lets you choose exactly where to look for images and in which order.
- *ReplayGain Plugin*: A new Python Audio Tools backend was added. Thanks to Francesco Rubino. #1070
- *EmbedArt Plugin*: You can now automatically check that new art looks similar to existing art—ensuring that you only get a better "version" of the art you already have. See *Image Similarity*.
- *FtInTitle Plugin*: The plugin now runs automatically on import. To disable this, unset the `auto` config flag.

There are also core improvements and other substantial additions:

- The `media` attribute is now a *track-level field* instead of an album-level one. This field stores the delivery mechanism for the music, so in its album-level incarnation, it could not represent heterogeneous releases—for example, an album consisting of a CD and a DVD. Now, tracks accurately indicate the media they appear on. Thanks to Heinz Wiesinger.
- Re-imports of your existing music (see *Reimporting*) now preserve its added date and flexible attributes. Thanks to Stig Inge Lea Bjørnsen.
- Slow queries, such as those over flexible attributes, should now be much faster when used with certain commands—notably, the *Play Plugin*.
- *BPD Plugin*: Add a new configuration option for setting the default volume. Thanks to IndiGit.
- *EmbedArt Plugin*: A new `ifempty` config option lets you only embed album art when no album art is present. Thanks to kerobaros.
- *Discogs Plugin*: Authenticate with the Discogs server. The plugin now requires a Discogs account due to new API restrictions. Thanks to multikatt. #1027, #1040

And countless little improvements and fixes:

- Standard cover art in APEv2 metadata is now supported. Thanks to Matthias Kiefer. #1042
- *Convert Plugin*: Avoid a crash when embedding cover art fails.
- *MPDStats Plugin*: Fix an error on start (introduced in the previous version). Thanks to Zach Denton.
- *Convert Plugin*: The `--yes` command-line flag no longer expects an argument.
- *Play Plugin*: Remove the temporary .m3u file after sending it to the player.
- The importer no longer tries to highlight partial differences in numeric quantities (track numbers and durations), which was often confusing.
- Date-based queries that are malformed (not parse-able) no longer crash beets and instead fail silently.
- *Duplicates Plugin*: Emit an error when the `checksum` config option is set incorrectly.
- The migration from pre-1.1, non-YAML configuration files has been removed. If you need to upgrade an old config file, use an older version of beets temporarily.
- *Discogs Plugin*: Recover from HTTP errors when communicating with the Discogs servers. Thanks to Dustin Rodriguez.
- *EmbedArt Plugin*: Do not log "embedding album art into…" messages during the import process.
- Fix a crash in the autotagger when files had only whitespace in their metadata.
- *Play Plugin*: Fix a potential crash when the command outputs special characters. #1041

- *Web Plugin*: Queries typed into the search field are now treated as separate query components. #1045
- Date tags that use slashes instead of dashes as separators are now interpreted correctly. And WMA (ASF) files now map the `comments` field to the "Description" tag (in addition to "WM/Comments"). Thanks to Matthias Kiefer. #1043
- *EmbedArt Plugin*: Avoid resizing the image multiple times when embedding into an album. Thanks to kerobaros. #1028, #1036
- *Discogs Plugin*: Avoid a situation where a trailing comma could be appended to some artist names. #1049
- The output of the *stats* command is slightly different: the approximate size is now marked as such, and the total number of seconds only appears in exact mode.
- *Convert Plugin*: A new `copy_album_art` option puts images alongside converted files. Thanks to Ángel Alonso. #1050, #1055
- There is no longer a "conflict" between two plugins that declare the same field with the same type. Thanks to Peter Schnebel. #1059 #1061
- *Chromaprint/Acoustid Plugin*: Limit the number of releases and recordings fetched as the result of an Acoustid match to avoid extremely long processing times for very popular music. #1068
- Fix an issue where modifying an album's field without actually changing it would not update the corresponding tracks to bring differing tracks back in line with the album. #856
- `echonest` plugin: When communicating with the Echo Nest servers fails repeatedly, log an error instead of exiting. #1096
- *Lyrics Plugin*: Avoid an error when the Google source returns a result without a title. Thanks to Alberto Leal. #1097
- Importing an archive will no longer leave temporary files behind in `/tmp`. Thanks to multikatt. #1067, #1091

1.7.23 1.3.8 (September 17, 2014)

This release has two big new chunks of functionality. Queries now support **sorting** and user-defined fields can now have **types**.

If you want to see all your songs in reverse chronological order, just type `beet list year-`. It couldn't be easier. For details, see *Sort Order*.

Flexible field types mean that some functionality that has previously only worked for built-in fields, like range queries, can now work with plugin- and user-defined fields too. For starters, the `echonest` plugin and *MPDStats Plugin* now mark the types of the fields they provide—so you can now say, for example, `beet ls liveness:0.5..1.5` for the Echo Nest "liveness" attribute. The *Types Plugin* makes it easy to specify field types in your config file.

One upgrade note: if you use the *Discogs Plugin*, you will need to upgrade the Discogs client library to use this version. Just type `pip install -U discogs-client`.

Other new features:

- *Info Plugin*: Target files can now be specified through library queries (in addition to filenames). The `--library` option prints library fields instead of tags. Multiple files can be summarized together with the new `--summarize` option.
- *MusicBrainz Collection Plugin*: A new option lets you automatically update your collection on import. Thanks to Olin Gay.
- *Convert Plugin*: A new `never_convert_lossy_files` option can prevent lossy transcoding. Thanks to Simon Kohlmeyer.
- *Convert Plugin*: A new `--yes` command-line flag skips the confirmation.

Still more fixes and little improvements:

- Invalid state files don't crash the importer.
- *Lyrics Plugin*: Only strip featured artists and parenthesized title suffixes if no lyrics for the original artist and title were found.
- Fix a crash when reading some files with missing tags.
- *Discogs Plugin*: Compatibility with the new 2.0 version of the discogs_client Python library. If you were using the old version, you wil need to upgrade to the latest version of the library to use the correspondingly new version of the plugin (e.g., with `pip install -U discogs-client`). Thanks to Andriy Kohut.
- Fix a crash when writing files that can't be read. Thanks to Jocelyn De La Rosa.
- The *stats* command now counts album artists. The album count also more accurately reflects the number of albums in the database.
- *Convert Plugin*: Avoid crashes when tags cannot be written to newly converted files.
- Formatting templates with item data no longer confusingly shows album-level data when the two are inconsistent.
- Resuming imports and beginning incremental imports should now be much faster when there is a lot of previously-imported music to skip.
- *Lyrics Plugin*: Remove `<script>` tags from scraped lyrics. Thanks to Bombardment.
- *Play Plugin*: Add a `relative_to` config option. Thanks to BrainDamage.
- Fix a crash when a MusicBrainz release has zero tracks.
- The `--version` flag now works as an alias for the `version` command.
- *LastGenre Plugin*: Remove some unhelpful genres from the default whitelist. Thanks to gwern.
- *ImportFeeds Plugin*: A new `echo` output mode prints files' paths to standard error. Thanks to robotanarchy.
- *ReplayGain Plugin*: Restore some error handling when `mp3gain` output cannot be parsed. The verbose log now contains the bad tool output in this case.
- *Convert Plugin*: Fix filename extensions when converting automatically.
- The `write` plugin event allows plugins to change the tags that are written to a media file.
- *Zero Plugin*: Do not delete database values; only media file tags are affected.

1.7.24 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.

1.7.25 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.

1.7.26 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 `echonest` 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:

- `echonest` 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.

1.7.27 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.
- `echonest` 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.
- `echonest` 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.

1.7.28 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:

- `echonest` 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.
- `echonest` 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.

1.7.29 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 `echonest` 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`: 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.

1.7.30 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`: 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.

1.7.31 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.

1.7.32 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.

1.7.33 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`, *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` 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.

1.7.34 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.

1.7.35 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.

- *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.

1.7.36 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-musicbrainzngs 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`: 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`: Catch socket errors that are not handled by the Echo Nest library.
- *Chromaprint/Acoustid Plugin*: Catch Acoustid Web service errors when submitting fingerprints.

1.7.37 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`: 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.

1.7.38 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).

- 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.

1.7.39 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.

1.7.40 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`: 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".

1.7.41 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` 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 `$path`).
- The output of the *update*, *remove*, and *modify* commands now respects the *format_album* and *format_item* config options. Thanks to Mike Kazantsev.
- The *art_filename* option can now be a template rather than a simple string. Thanks to Jarrod Beardwood.
- Fix album queries for `artpath` and other non-item fields.
- Null values in the database can now be matched with the empty-string regular expression, `^$`.
- 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.

1.7.42 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 *format_item* and *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.

1.7.43 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.

1.7.44 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).

1.7.45 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.

1.7.46 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-musicbrainzngs. 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.

1.7.47 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.

1.7.48 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.

1.7.49 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.

1.7.50 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.

1.7.51 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.

1.7.52 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.

1.7.53 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.

1.7.54 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!

1.7.55 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.

1.7.56 1.0b1 (June 17, 2010)

Initial release.

Index

Symbols

__init__() (*beets.library.Album method*), 142
__init__() (*beets.library.Item method*), 141
__init__() (*beets.library.LibModel method*), 139
__init__() (*beets.library.Library method*), 138
_fields (*beets.library.LibModel attribute*), 139
_types (*beets.library.LibModel attribute*), 139

A

add() (*beets.library.LibModel method*), 139
add() (*beets.library.Library method*), 138
add_album() (*beets.library.Library method*), 138
add_media_field() (*beets.plugins.BeetsPlugin method*), 135
Album (*class in beets.library*), 142
albums() (*beets.library.Library method*), 139
all_keys() (*beets.library.LibModel class method*), 139
art_destination() (*beets.library.Album method*), 143

C

current_mtime() (*beets.library.Item method*), 141

D

destination() (*beets.library.Item method*), 141

F

from_path() (*beets.library.Item class method*), 141

G

get() (*beets.library.LibModel method*), 140
get_album() (*beets.library.Item method*), 141
get_album() (*beets.library.Library method*), 139
get_item() (*beets.library.Library method*), 139

I

Item (*class in beets.library*), 140
item_dir() (*beets.library.Album method*), 142
item_keys (*beets.library.Album attribute*), 142
items() (*beets.library.LibModel method*), 140
items() (*beets.library.Library method*), 138

K

keys() (*beets.library.LibModel method*), 140

L

LibModel (*class in beets.library*), 139
Library (*class in beets.library*), 138
load() (*beets.library.LibModel method*), 139

M

move() (*beets.library.Album method*), 142
move() (*beets.library.Item method*), 141
move_art() (*beets.library.Album method*), 143
mutate() (*beets.dbcore.db.Transaction method*), 143

Q

query() (*beets.dbcore.db.Transaction method*), 143

R

read() (*beets.library.Item method*), 141
remove() (*beets.library.Album method*), 143
remove() (*beets.library.Item method*), 142
remove() (*beets.library.LibModel method*), 139

S

script() (*beets.dbcore.db.Transaction method*), 143
set_art() (*beets.library.Album method*), 143
store() (*beets.library.Album method*), 142
store() (*beets.library.LibModel method*), 139

T

Transaction (*class in beets.dbcore.db*), 143
transaction() (*beets.library.Library method*), 139
try_sync() (*beets.library.Album method*), 142
try_sync() (*beets.library.Item method*), 141
try_write() (*beets.library.Item method*), 141

U

update() (*beets.library.LibModel method*), 140

W

write() (*beets.library.Item method*), 141