Mopidy Documentation

Release 1.1.2-2-g1c19dd5d

Stein Magnus Jodal and contributors

2016

Usage

Mopidy is an extensible music server written in Python.

Mopidy plays music from local disk, Spotify, SoundCloud, Google Play Music, and more. You edit the playlist from any phone, tablet, or computer using a range of MPD and web clients.

Stream music from the cloud

Vanilla Mopidy only plays music from your *local disk* and *radio streams*. Through *extensions*, Mopidy can play music from cloud services like Spotify, SoundCloud, and Google Play Music. With Mopidy's extension support, backends for new music sources can be easily added.

Mopidy is just a server

Mopidy is a Python application that runs in a terminal or in the background on Linux computers or Macs that have network connectivity and audio output. Out of the box, Mopidy is an *MPD* and *HTTP* server. *Additional frontends* for controlling Mopidy can be installed from extensions.

Everybody use their favorite client

You and the people around you can all connect their favorite *MPD* or *web client* to the Mopidy server to search for music and manage the playlist together. With a browser or MPD client, which is available for all popular operating systems, you can control the music from any phone, tablet, or computer.

Mopidy on Raspberry Pi

The *Raspberry Pi* is a popular device to run Mopidy on, either using Raspbian or Arch Linux. It is quite slow, but it is very affordable. In fact, the Kickstarter funded Gramofon: Modern Cloud Jukebox project used Mopidy on a Raspberry Pi to prototype the Gramofon device. Mopidy is also a major building block in the Pi Musicbox integrated audio jukebox system for Raspberry Pi.

Mopidy is hackable

Mopidy's extension support and *Python*, *JSON-RPC*, and *JavaScript APIs* makes Mopidy perfect for building your own hacks. In one project, a Raspberry Pi was embedded in an old cassette player. The buttons and volume control are wired up with GPIO on the Raspberry Pi, and is used to control playback through a custom Mopidy extension. The cassettes have NFC tags used to select playlists from Spotify.

Getting started

To get started with Mopidy, start by reading *Installation*.

Getting help

If you get stuck, you can get help at the Mopidy discussion forum. We also hang around at IRC on the `#mopidy` channel at irc.freenode.net. The IRC channel has public searchable logs.

If you stumble into a bug or have a feature request, please create an issue in the issue tracker. If you're unsure if it's a bug or not, ask for help in the forum or at IRC first. The source code may also be of help.

If you want to stay up to date on Mopidy developments, you can follow @mopidy on Twitter. There's also a mailing list used for announcements related to Mopidy and Mopidy extensions.

CHAPTER 1

Installation

There are several ways to install Mopidy. What way is best depends upon your OS and/or distribution.

If you want to contribute to the development of Mopidy, you should first follow the instructions here to install a regular install of Mopidy, then continue with reading *Contributing* and *Development environment*.

1.1 Debian/Ubuntu: Install from apt.mopidy.com

If you run a Debian based Linux distribution, like Ubuntu, the easiest way to install Mopidy is from the Mopidy APT archive. When installing from the APT archive, you will automatically get updates to Mopidy in the same way as you get updates to the rest of your system.

If you're on a Raspberry Pi running Debian or Raspbian, the following instructions should work for you as well. If you're setting up a Raspberry Pi from scratch, we have a guide for installing Debian/Raspbian and Mopidy. See *Raspberry Pi*.

The packages are built for:

- Debian wheezy (oldstable), which also works for Raspbian wheezy and Ubuntu 12.04 LTS.
- Debian jessie (stable), which also works for Raspbian jessie and Ubuntu 14.04 LTS and newer.

The packages are available for multiple CPU architectures: i386, amd64, armel, and armhf (compatible with Raspberry Pi 1 and 2).

Note: This is just what we currently support, not a promise to continue to support the same in the future. We *will* drop support for older distributions and architectures when supporting those stops us from moving forward with the project.

1. Add the archive's GPG key:

```
wget -q -O - https://apt.mopidy.com/mopidy.gpg | sudo apt-key add -
```

2. If you run Debian wheezy or Ubuntu 12.04 LTS:

```
sudo wget -q -O /etc/apt/sources.list.d/mopidy.list https://apt.mopidy.com/wheezy.
↪list
```

Or, if you run any newer Debian/Ubuntu distro:

```
sudo wget -q -O /etc/apt/sources.list.d/mopidy.list https://apt.mopidy.com/jessie.
↪list
```

3. Install Mopidy and all dependencies:

```
sudo apt-get update
sudo apt-get install mopidy
```

4. Finally, you need to set a couple of *config values*, and then you're ready to *run Mopidy* or run Mopidy as a *service*.

When a new release of Mopidy is out, and you can't wait for you system to figure it out for itself, run the following to upgrade right away:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

1.1.1 Installing extensions

If you want to use any Mopidy extensions, like Spotify support or Last.fm scrobbling, you need to install additional packages.

To list all the extensions available from apt.mopidy.com, you can run:

```
apt-cache search mopidy
```

To install one of the listed packages, e.g. `mopidy-spotify`, simply run:

```
sudo apt-get install mopidy-spotify
```

You can also install any Mopidy extension directly from PyPI with `pip`. To list all the extensions available from PyPI, run:

```
pip search mopidy
```

Note that extensions installed from PyPI will only automatically install Python dependencies. Please refer to the extension's documentation for information about any other requirements needed for the extension to work properly.

For a full list of available Mopidy extensions, including those not installable from apt.mopidy.com, see *Extensions*.

1.2 Arch Linux: Install from community

If you are running Arch Linux, you can install Mopidy using the mopidy package found in `community`.

1. To install Mopidy with all dependencies, you can use:

```
pacman -S mopidy
```

To upgrade Mopidy to future releases, just upgrade your system using:

```
pacman -Syu
```

2. Finally, you need to set a couple of *config values*, and then you're ready to *run Mopidy* or run Mopidy as a *service*.

1.2.1 Installing extensions

If you want to use any Mopidy extensions, like Spotify support or Last.fm scrobbling, AUR has packages for lots of Mopidy extensions.

You can also install any Mopidy extension directly from PyPI with `pip`. To list all the extensions available from PyPI, run:

```
pip search mopidy
```

Note that extensions installed from PyPI will only automatically install Python dependencies. Please refer to the extension's documentation for information about any other requirements needed for the extension to work properly.

For a full list of available Mopidy extensions, including those not installable from AUR, see *Extensions*.

1.3 OS X: Install from Homebrew

If you are running OS X, you can install everything needed with Homebrew.

1. Install Xcode command line developer tools. Do this even if you already have Xcode installed:

```
xcode-select --install
```

2. Install XQuartz. This is needed by GStreamer which Mopidy use heavily.
3. Install Homebrew.
4. If you are already using Homebrew, make sure your installation is up to date before you continue:

```
brew update
brew upgrade --all
```

Notice that this will upgrade all software on your system that have been installed with Homebrew.

5. Mopidy works out of box if you have installed Python from Homebrew:

```
brew install python
```

Note: If you want to use the Python version bundled with OS X, you'll need to include Python packages installed by Homebrew in your `PYTHONPATH`. If you don't do this, the `mopidy` executable will not find its dependencies and will crash.

You can either amend your `PYTHONPATH` permanently, by adding the following statement to your shell's init file, e.g. `~/.bashrc`:

```
export PYTHONPATH=$(brew --prefix)/lib/python2.7/site-packages:$PYTHONPATH
```

And then reload the shell's init file or restart your terminal:

```
source ~/.bashrc
```

Or, you can prefix the Mopidy command every time you run it:

```
PYTHONPATH=$(brew --prefix)/lib/python2.7/site-packages mopidy
```

---

6. Mopidy has its own Homebrew formula repo, called a "tap". To enable our Homebrew tap, run:

   ```
   brew tap mopidy/mopidy
   ```

7. To install Mopidy, run:

   ```
   brew install mopidy
   ```

8. Finally, you need to set a couple of *config values*, and then you're ready to *run Mopidy*.

1.3.1 Installing extensions

If you want to use any Mopidy extensions, like Spotify support or Last.fm scrobbling, the Homebrew tap has formulas for several Mopidy extensions as well. Extensions installed from Homebrew will come complete with all dependencies, both Python and non-Python ones.

To list all the extensions available from our tap, you can run:

```
brew search mopidy
```

You can also install any Mopidy extension directly from PyPI with `pip`, just like on Linux. To list all the extensions available from PyPI, run:

```
pip search mopidy
```

Note that extensions installed from PyPI will only automatically install Python dependencies. Please refer to the extension's documentation for information about any other requirements needed for the extension to work properly.

For a full list of available Mopidy extensions, including those not installable from Homebrew, see *Extensions*.

1.3.2 Running Mopidy automatically on login

On OS X, you can use launchd to start Mopidy automatically at login.

If you installed Mopidy from Homebrew, simply run `brew info mopidy` and follow the instructions in the "Caveats" section:

```
$ brew info mopidy
...
==> Caveats
To have launchd start mopidy at login:
    ln -sfv /usr/local/opt/mopidy/*.plist ~/Library/LaunchAgents
Then to load mopidy now:
    launchctl load ~/Library/LaunchAgents/homebrew.mopidy.mopidy.plist
Or, if you don't want/need launchctl, you can just run:
    mopidy
```

If you happen to be on OS X, but didn't install Mopidy with Homebrew, you can get the same effect by adding the file `~/Library/LaunchAgents/mopidy.plist` with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
↪PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>mopidy</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/mopidy</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>KeepAlive</key>
  <true/>
</dict>
</plist>
```

You might need to adjust the path to the `mopidy` executable, `/usr/local/bin/mopidy`, to match your system.

Then, to start Mopidy with launchd right away:

```
launchctl load ~/Library/LaunchAgents/mopidy.plist
```

1.4 Install from source

If you are on Linux, but can't install *from the APT archive* or *from the Arch Linux repository*, you can install Mopidy from PyPI using the `pip` installer.

If you are looking to contribute or wish to install from source using `git` please follow the directions *here*.

1. First of all, you need Python 2.7. Check if you have Python and what version by running:

   ```
   python --version
   ```

2. You need to make sure you have `pip`, the Python package installer. You'll also need a C compiler and the Python development headers to build pyspotify later.

   This is how you install it on Debian/Ubuntu:

   ```
   sudo apt-get install build-essential python-dev python-pip
   ```

   And on Arch Linux from the official repository:

   ```
   sudo pacman -S base-devel python2-pip
   ```

   And on Fedora Linux from the official repositories:

   ```
   sudo yum install -y gcc python-devel python-pip
   ```

   Note: On Fedora Linux, you must replace `pip` with `pip-python` in the following steps.

3. Then you'll need to install GStreamer 0.10 (>= 0.10.31, < 0.11), with Python bindings. GStreamer is packaged for most popular Linux distributions. Search for GStreamer in your package manager, and make sure to install the Python bindings, and the "good" and "ugly" plugin sets.

If you use Debian/Ubuntu you can install GStreamer like this:

```
sudo apt-get install python-gst0.10 gstreamer0.10-plugins-good \
    gstreamer0.10-plugins-ugly gstreamer0.10-tools
```

If you use Arch Linux, install the following packages from the official repository:

```
sudo pacman -S gstreamer0.10-python gstreamer0.10-good-plugins \
    gstreamer0.10-ugly-plugins
```

If you use Fedora you can install GStreamer like this:

```
sudo yum install -y python-gst0.10 gstreamer0.10-plugins-good \
    gstreamer0.10-plugins-ugly gstreamer0.10-tools
```

If you use Gentoo you need to be careful because GStreamer 0.10 is in a different lower slot than 1.0, the default. Your emerge commands will need to include the slot:

```
emerge -av gst-python gst-plugins-bad:0.10 gst-plugins-good:0.10 \
    gst-plugins-ugly:0.10 gst-plugins-meta:0.10
```

`gst-plugins-meta:0.10` is the one that actually pulls in the plugins you want, so pay attention to the use flags, e.g. `alsa`, `mp3`, etc.

4. Install the latest release of Mopidy:

```
sudo pip install -U mopidy
```

This will use `pip` to install the latest release of Mopidy from PyPI. To upgrade Mopidy to future releases, just rerun this command.

Alternatively, if you want to track Mopidy development closer, you may install a snapshot of Mopidy's `develop` Git branch using pip:

```
sudo pip install --allow-unverified=mopidy mopidy==dev
```

5. Finally, you need to set a couple of *config values*, and then you're ready to *run Mopidy*.

1.4.1 Installing extensions

If you want to use any Mopidy extensions, like Spotify support or Last.fm scrobbling, you need to install additional Mopidy extensions.

You can install any Mopidy extension directly from PyPI with `pip`. To list all the extensions available from PyPI, run:

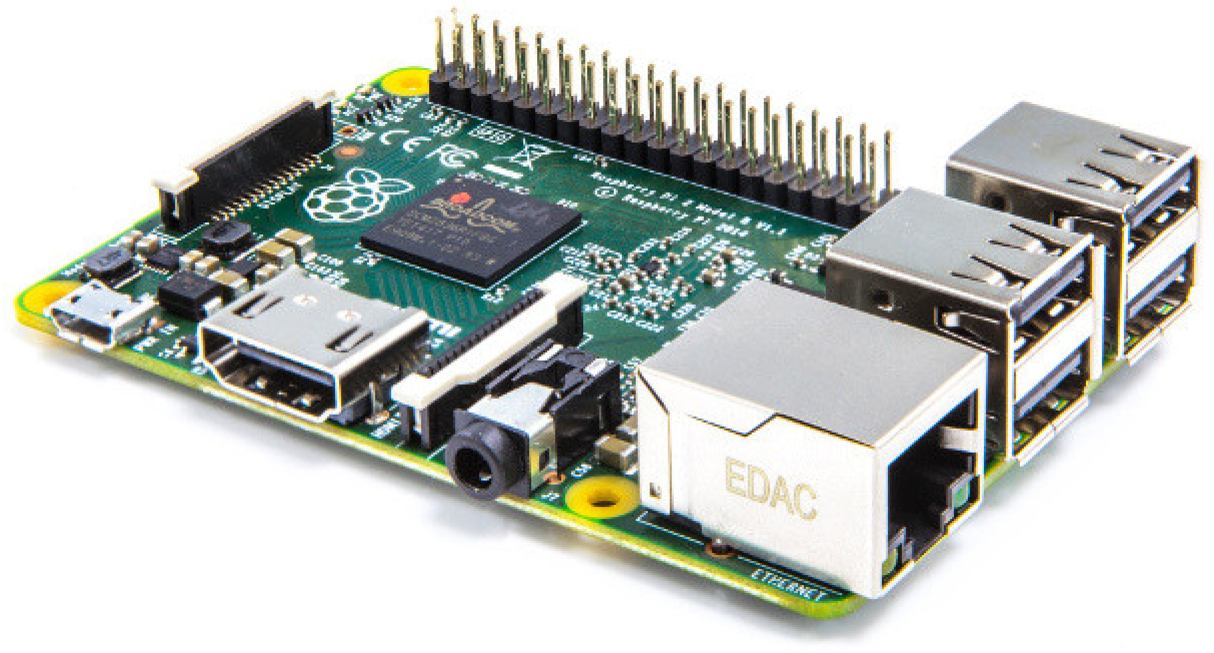
```
pip search mopidy
```

Note that extensions installed from PyPI will only automatically install Python dependencies. Please refer to the extension's documentation for information about any other requirements needed for the extension to work properly.

For a full list of available Mopidy extensions see *Extensions*.

1.5 Raspberry Pi

Mopidy runs on all versions of Raspberry Pi. However, note that Raspberry Pi 2 B's CPU is approximately six times as powerful as Raspberry Pi 1 and Raspberry Pi Zero, so Mopidy will be more joyful to use on a Raspberry Pi 2.



1.5.1 How to for Raspbian Jessie

1. Download the latest Jessie or Jessie Lite disk image from http://www.raspberrypi.org/downloads/raspbian/.

   If you're only using your Pi for Mopidy, go with Jessie Lite as you won't need the full graphical desktop included in the Jessie image.

2. Flash the Raspbian image you downloaded to your SD card.

   See the Raspberry Pi installation docs for instructions.

3. If you connect a monitor and a keyboard, you'll see that the Pi boots right into the `raspi-config` tool.

   If you boot with only a network cable connected, you'll have to find the IP address of the Pi yourself, e.g. by looking in the client list on your router/DHCP server. When you have found the Pi's IP address, you can SSH to the IP address and login with the user `pi` and password `raspberry`. Once logged in, run `sudo raspi-config` to start the config tool as the `root` user.

4. Use the `raspi-config` tool to setup the basics of your Pi. You might want to do one or more of the following:

   - Expand the file system to fill the SD card.
   - Change the password of the `pi` user.
   - Change the time zone.

   Under "Advanced Options":

   - Set a hostname.
   - Enable SSH if not already enabled.
   - If your will use HDMI for display and 3.5mm jack for audio, force the audio output to the 3.5mm jack. By default it will use HDMI for audio output if an HDMI cable is connected and the 3.5mm jack if not.

Once done, select "Finish" and restart your Pi.

If you want to change any settings later, you can simply rerun `sudo raspi-config`.

5. Once you've rebooted and has logged in as the `pi` user, you can enter `sudo -i` to become `root`.
6. Install Mopidy and its dependencies as described in *Debian/Ubuntu: Install from apt.mopidy.com*.
7. Finally, you need to set a couple of *config values*, and then you're ready to *run Mopidy*. Alternatively you may want to have Mopidy run as a system service, automatically starting at boot.

1.5.2 Testing sound output

You can test sound output independent of Mopidy by running:

```
aplay /usr/share/sounds/alsa/Front_Center.wav
```

If you hear a voice saying "Front Center", then your sound is working.

If you want to change your audio output setting, simply rerun `sudo raspi-config`. Alternatively, you can change the audio output setting directly by running:

- Auto (HDMI if connected, else 3.5mm jack): `sudo amixer cset numid=3 0`
- Use 3.5mm jack: `sudo amixer cset numid=3 1`
- Use HDMI: `sudo amixer cset numid=3 2`

CHAPTER 2

Configuration

Mopidy has a lot of config values you can tweak, but you only need to change a few to get up and running. A complete `~/.config/mopidy/mopidy.conf` may be as simple as this:

```
[mpd]
hostname = ::

[spotify]
username = alice
password = mysecret
```

Mopidy primarily reads config from the file `~/.config/mopidy/mopidy.conf`, where ~ means your *home directory*. If your username is `alice` and you are running Linux, the config file should probably be at `/home/alice/.config/mopidy/mopidy.conf`. You can either create the configuration file yourself, or run the `mopidy` command, and it will create an empty config file for you and print what config values must be set to successfully start Mopidy.

When you have created the configuration file, open it in a text editor, and add the config values you want to change. If you want to keep the default for a config value, you **should not** add it to the config file, but leave it out so that when we change the default value in a future version, you won't have to change your configuration accordingly.

To see what's the effective configuration for your Mopidy installation, you can run:

```
mopidy config
```

This will print your full effective config with passwords masked out so that you safely can share the output with others for debugging.

You can find a description of all config values belonging to Mopidy's core below, together with their default values. In addition, all *extensions* got additional config values. The extension's config values and config defaults are documented on the *extension pages*.

2.1 Default core configuration

```
[core]
cache_dir = $XDG_CACHE_DIR/mopidy
config_dir = $XDG_CONFIG_DIR/mopidy
data_dir = $XDG_DATA_DIR/mopidy
max_tracklist_length = 10000

[logging]
color = true
console_format = %(levelname)-8s %(message)s
debug_format = %(levelname)-8s %(asctime)s [%(process)d:%(threadName)s] %(name)s\n
↪%(message)s
debug_file = mopidy.log
config_file =

[audio]
mixer = software
mixer_volume =
output = autoaudiosink

[proxy]
scheme =
hostname =
port =
username =
password =
```

2.2 Core configuration values

Mopidy's core has the following configuration values that you can change.

2.2.1 Core configuration

core/cache_dir

Path to base directory for storing cached data.

Mopidy and extensions will use this path to cache data that can safely be thrown away.

If your system is running from an SD card, it can help avoid wear and corruption of your SD card by pointing this config to another location. If you have enough RAM, a tmpfs might be a good choice.

When running Mopidy as a regular user, this should usually be `$XDG_CACHE_DIR/mopidy`, i.e. `~/.cache/mopidy`.

When running Mopidy as a system service, this should usually be `/var/cache/mopidy`.

core/config_dir

Path to base directory for config files.

When running Mopidy as a regular user, this should usually be `$XDG_CONFIG_DIR/mopidy`, i.e. `~/.config/mopidy`.

When running Mopidy as a system service, this should usually be `/etc/mopidy`.

`core/data_dir`
Path to base directory for persistent data files.

Mopidy and extensions will use this path to store data that cannot be be thrown away and reproduced without some effort. Examples include Mopidy-Local's index of your media library and Mopidy-M3U's stored playlists.

When running Mopidy as a regular user, this should usually be `$XDG_DATA_DIR/mopidy`, i.e. `~/.local/share/mopidy`.

When running Mopidy as a system service, this should usually be `/var/lib/mopidy`.

`core/max_tracklist_length`
Max length of the tracklist. Defaults to 10000.

The original MPD server only supports 10000 tracks in the tracklist. Some MPD clients will crash if this limit is exceeded.

2.2.2 Audio configuration

`audio/mixer`
Audio mixer to use.

The default is `software`, which does volume control inside Mopidy before the audio is sent to the audio output. This mixer does not affect the volume of any other audio playback on the system. It is the only mixer that will affect the audio volume if you're streaming the audio from Mopidy through Shoutcast.

If you want to disable audio mixing set the value to `none`.

If you want to use a hardware mixer, you need to install a Mopidy extension which integrates with your sound subsystem. E.g. for ALSA, install Mopidy-ALSAMixer.

`audio/mixer_volume`
Initial volume for the audio mixer.

Expects an integer between 0 and 100.

Setting the config value to blank leaves the audio mixer volume unchanged. For the software mixer blank means 100.

`audio/output`
Audio output to use.

Expects a GStreamer sink. Typical values are `autoaudiosink`, `alsasink`, `osssink`, `oss4sink`, `pulsesink`, and `shout2send`, and additional arguments specific to each sink. You can use the command `gst-inspect-0.10` to see what output properties can be set on the sink. For example: `gst-inspect-0.10 shout2send`

2.2.3 Logging configuration

`logging/color`
Whether or not to colorize the console log based on log level. Defaults to `true`.

`logging/console_format`
The log format used for informational logging.

See the Python logging docs for details on the format.

`logging/debug_format`
The log format used for debug logging.

See the Python logging docs for details on the format.

logging/debug_file
: The file to dump debug log data to when Mopidy is run with the *`mopidy --save-debug-log`* option.

logging/config_file
: Config file that overrides all logging config values, see the Python logging docs for details.

loglevels/*
: The `loglevels` config section can be used to change the log level for specific parts of Mopidy during development or debugging. Each key in the config section should match the name of a logger. The value is the log level to use for that logger, one of `debug`, `info`, `warning`, `error`, or `critical`.

logcolors/*
: The `logcolors` config section can be used to change the log color for specific parts of Mopidy during development or debugging. Each key in the config section should match the name of a logger. The value is the color to use for that logger, one of `black`, `red`, `green`, `yellow`, `blue`, `magenta`, `cyan` or `white`.

2.2.4 Proxy configuration

Not all parts of Mopidy or all Mopidy extensions respect the proxy server configuration when connecting to the Internt. Currently, this is at least used when Mopidy's audio subsystem reads media directly from the network, like when listening to Internet radio streams, and by the Mopidy-Spotify extension. With time, we hope that more of the Mopidy ecosystem will respect these configurations to help users on locked down networks.

proxy/scheme
: URI scheme for the proxy server. Typically `http`, `https`, `socks4`, or `socks5`.

proxy/hostname
: Hostname of the proxy server.

proxy/port
: Port number of the proxy server.

proxy/username
: Username for the proxy server, if needed.

proxy/password
: Password for the proxy server, if needed.

2.3 Extension configuration

Mopidy's extensions have their own config values that you may want to tweak. For the available config values, please refer to the docs for each extension. Most, if not all, can be found at *Extensions*.

Mopidy extensions are enabled by default when they are installed. If you want to disable an extension without uninstalling it, all extensions support the `enabled` config value even if it isn't explicitly documented by all extensions. If the `enabled` config value is set to `false` the extension will not be started. For example, to disable the Spotify extension, add the following to your `mopidy.conf`:

```
[spotify]
enabled = false
```

2.4 Advanced configurations

2.4.1 Custom audio sink

If you have successfully installed GStreamer, and then run the `gst-inspect` or `gst-inspect-0.10` command, you should see a long listing of installed plugins, ending in a summary line:

```
$ gst-inspect-0.10
... long list of installed plugins ...
Total count: 254 plugins (1 blacklist entry not shown), 1156 features
```

Next, you should be able to produce a audible tone by running:

```
gst-launch-0.10 audiotestsrc ! audioresample ! autoaudiosink
```

If you cannot hear any sound when running this command, you won't hear any sound from Mopidy either, as Mopidy by default uses GStreamer's `autoaudiosink` to play audio. Thus, make this work before you file a bug against Mopidy.

If you for some reason want to use some other GStreamer audio sink than `autoaudiosink`, you can set the *`audio/output`* config value to a partial GStreamer pipeline description describing the GStreamer sink you want to use.

Example `mopidy.conf` for using OSS4:

```
[audio]
output = oss4sink
```

Again, this is the equivalent of the following `gst-inspect` command, so make this work first:

```
gst-launch-0.10 audiotestsrc ! audioresample ! oss4sink
```

2.4.2 Streaming through SHOUTcast/Icecast

Warning: Known issue

Currently, Mopidy does not handle end-of-track vs end-of-stream signalling in GStreamer correctly. This causes the SHOUTcast stream to be disconnected at the end of each track, rendering it quite useless. For further details, see #492. You can also try the *workaround* mentioned below.

If you want to play the audio on another computer than the one running Mopidy, you can stream the audio from Mopidy through an SHOUTcast or Icecast audio streaming server. Multiple media players can then be connected to the streaming server simultaneously. To use the SHOUTcast output, do the following:

1. Install, configure and start the Icecast server. It can be found in the `icecast2` package in Debian/Ubuntu.
2. Set the *`audio/output`* config value to `lame ! shout2send`. An Ogg Vorbis encoder could be used instead of the lame MP3 encoder.
3. You might also need to change the `shout2send` default settings, run `gst-inspect-0.10 shout2send` to see the available settings. Most likely you want to change `ip`, `username`, `password`, and `mount`.

   Example for MP3 streaming:

   ```
   [audio]
   output = lame ! shout2send mount=mopidy ip=127.0.0.1 port=8000 password=hackme
   ```

Example for Ogg Vorbis streaming:

```
[audio]
output = audioresample ! audioconvert ! vorbisenc ! oggmux ! shout2send␣
↪mount=mopidy ip=127.0.0.1 port=8000 password=hackme
```

Other advanced setups are also possible for outputs. Basically, anything you can use with the `gst-launch-0.10` command can be plugged into *`audio/output`*.

Workaround for end-of-track issues - fallback streams

By using a *fallback stream* playing silence, you can somewhat mitigate the signalling issues.

Example Icecast configuration:

```
<mount>
  <mount-name>/mopidy</mount-name>
  <fallback-mount>/silence.mp3</fallback-mount>
  <fallback-override>1</fallback-override>
</mount>
```

The `silence.mp3` file needs to be placed in the directory defined by `<webroot>...</webroot>`.

2.4.3 New configuration values

Mopidy's config validator will stop you from defining any config values in your config file that Mopidy doesn't know about. This may sound obnoxious, but it helps us detect typos in your config, and deprecated config values that should be removed or updated.

If you're extending Mopidy, and want to use Mopidy's configuration system, you can add new sections to the config without triggering the config validator. We recommend that you choose a good and unique name for the config section so that multiple extensions to Mopidy can be used at the same time without any danger of naming collisions.

CHAPTER 3

Running Mopidy

To start Mopidy, simply open a terminal and run:

```
mopidy
```

For a complete reference to the Mopidy commands and their command line options, see *mopidy command*.

When Mopidy says `MPD server running at [127.0.0.1]:6600` it's ready to accept connections by any MPD client. Check out our non-exhaustive *MPD clients* list to find recommended clients.

3.1 Updating the library

To update the library, e.g. after audio files have changed, run:

```
mopidy local scan
```

Afterwards, to refresh the library (which is for now only available through the API) it is necessary to run:

```
curl -d '{"jsonrpc": "2.0", "id": 1, "method": "core.library.refresh"}' http://
↪localhost:6680/mopidy/rpc
```

This makes the changes in the library visible to the clients.

3.2 Stopping Mopidy

To stop Mopidy, press `CTRL+C` in the terminal where you started Mopidy.

Mopidy will also shut down properly if you send it the TERM signal, e.g. by using `pkill`:

```
pkill mopidy
```

3.3 Running as a service

Once you’re done exploring Mopidy and want to run it as a proper service, check out *Running as a service*.

CHAPTER 4

Running as a service

If you want to run Mopidy as a service using either an init script or a systemd service, there's a few differences from running Mopidy as your own user you'll want to know about. The following applies to Debian, Ubuntu, Raspbian, and Arch. Hopefully, other distributions packaging Mopidy will make sure this works the same way on their distribution.

4.1 Configuration

All configuration is in `/etc/mopidy/mopidy.conf`, not in your user's home directory.

4.2 mopidy user

The Mopidy service runs as the `mopidy` user, which is automatically created when you install the Mopidy package. The `mopidy` user will need read access to any local music you want Mopidy to play.

4.3 Subcommands

To run Mopidy subcommands with the same user and config files as the service uses, you can use `sudo mopidyctl <subcommand>`. In other words, where you'll usually run:

```
mopidy config
```

You should instead run the following to inspect the service's configuration:

```
sudo mopidyctl config
```

The same applies to scanning your local music collection. Where you'll normally run:

```
mopidy local scan
```

You should instead run:

```
sudo mopidyctl local scan
```

4.4 Service management with systemd

On modern systems using systemd you can enable the Mopidy service by running:

```
sudo systemctl enable mopidy
```

This will make Mopidy start when the system boots.

Mopidy is started, stopped, and restarted just like any other systemd service:

```
sudo systemctl start mopidy
sudo systemctl stop mopidy
sudo systemctl restart mopidy
```

You can check if Mopidy is currently running as a service by running:

```
sudo systemctl status mopidy
```

4.5 Service management on Debian

On Debian systems (both those using systemd and not) you can enable the Mopidy service by running:

```
sudo dpkg-reconfigure mopidy
```

Mopidy can be started, stopped, and restarted using the `service` command:

```
sudo service mopidy start
sudo service mopidy stop
sudo service mopidy restart
```

You can check if Mopidy is currently running as a service by running:

```
sudo service mopidy status
```

4.6 Service on OS X

If you're installing Mopidy on OS X, see *Running Mopidy automatically on login*.

CHAPTER 5

Troubleshooting

If you run into problems with Mopidy, we usually hang around at `#mopidy` on irc.freenode.net and also have a discussion forum. If you stumble into a bug or have a feature request, please create an issue in the issue tracker.

When you're debugging yourself or asking for help, there are some tools built into Mopidy that you should know about.

5.1 Sharing config and log output

If you're getting help at IRC, we recommend that you use a pastebin, like pastebin.com or GitHub Gist, to share your configuration and log output. Pasting more than a couple of lines on IRC is generally frowned upon. On the mailing list or when reporting an issue, somewhat longer text dumps are accepted, but large logs should still be shared through a pastebin.

5.2 Show effective configuration

The command `mopidy config` will print your full effective configuration the way Mopidy sees it after all defaults and all config files have been merged into a single config document. Any secret values like passwords are masked out, so the output of the command should be safe to share with others for debugging.

5.3 Show installed dependencies

The command `mopidy deps` will list the paths to and versions of any dependency Mopidy or the extensions might need to work. This is very useful data for checking that you're using the right versions, and that you're using the right installation if you have multiple installations of a dependency on your system.

5.4 Debug logging

If you run *mopidy -v* or `mopidy -vv` or `mopidy -vvv` Mopidy will print more and more debug log to stdout. All three options will give you debug level output from Mopidy and extensions, while `-vv` and `-vvv` will give you more log output from their dependencies as well.

If you run *mopidy --save-debug-log*, it will save the log equivalent with `-vvv` to the file `mopidy.log` in the directory you ran the command from.

If you want to reduce the logging for some component, see the docs for the *loglevels/** config section.

5.5 Debugging deadlocks

If Mopidy hangs without an obvious explanation, you can send the `SIGUSR1` signal to the Mopidy process. If Mopidy's main thread is still responsive, it will log a traceback for each running thread, showing what the threads are currently doing. This is a very useful tool for understanding exactly how the system is deadlocking. If you have the `pkill` command installed, you can use this by simply running:

```
pkill -SIGUSR1 mopidy
```

5.6 Debugging GStreamer

If you really want to dig in and debug GStreamer behaviour, then check out the Debugging section of GStreamer's documentation for your options. Note that Mopidy does not support the GStreamer command line options, like `--gst-debug-level=3`, but setting GStreamer environment variables, like `GST_DEBUG`, works with Mopidy. For example, to run Mopidy with debug logging and GStreamer logging at level 3, you can run:

```
GST_DEBUG=3 mopidy -v
```

This will produce a lot of output, but given some GStreamer knowledge this is very useful for debugging GStreamer pipeline issues. Additionally `GST_DEBUG_FILE=gstreamer.log` can be used to redirect the debug logging to a file instead of standard out.

Lastly `GST_DEBUG_DUMP_DOT_DIR` can be used to get descriptions of the current pipeline in dot format. Currently we trigger a dump of the pipeline on every completed state change:

```
GST_DEBUG_DUMP_DOT_DIR=. mopidy
```

CHAPTER 6

Mopidy-Local

Mopidy-Local is an extension for playing music from your local music archive. It is bundled with Mopidy and enabled by default. Though, you'll have to scan your music collection to build a cache of metadata before the Mopidy-Local will be able to play your music.

This backend handles URIs starting with `local:`.

6.1 Generating a local library

The command **`mopidy local scan`** will scan the path set in the *`local/media_dir`* config value for any audio files and build a library of metadata.

To make a local library for your music available for Mopidy:

1. Ensure that the *`local/media_dir`* config value points to where your music is located. Check the current setting by running:

   ```
   mopidy config
   ```

2. Scan your media library.:

   ```
   mopidy local scan
   ```

3. Start Mopidy, find the music library in a client, and play some local music!

6.2 Updating the local library

When you've added or removed music in your collection and want to update Mopidy's index of your local library, you need to rescan:

```
mopidy local scan
```

Note that if you are using the default local library storage, `json`, you need to restart Mopidy after the scan completes for the updated index to be used.

If you want index updates to come into effect immediately, you can try out Mopidy-Local-SQLite, which will probably become the default backend in the near future.

6.3 Pluggable library support

Local libraries are fully pluggable. What this means is that users may opt to disable the current default library `json`, replacing it with a third party one. When running **`mopidy local scan`** Mopidy will populate whatever the current active library is with data. Only one library may be active at a time.

To create a new library provider you must create class that implements the *`mopidy.local.Library`* interface and install it in the extension registry under `local:library`. Any data that the library needs to store on disc should be stored in the extension's data dir, as returned by *`get_data_dir()`*.

6.4 Configuration

See *Configuration* for general help on configuring Mopidy.

```
[local]
enabled = true
library = json
media_dir = $XDG_MUSIC_DIR
data_dir = $XDG_DATA_DIR/mopidy/local
scan_timeout = 1000
scan_flush_threshold = 100
scan_follow_symlinks = false
excluded_file_extensions =
  .directory
  .html
  .jpeg
  .jpg
  .log
  .nfo
  .png
  .txt
```

`local/enabled`
If the local extension should be enabled or not.

`local/library`
Local library provider to use, change this if you want to use a third party library for local files.

`local/media_dir`
Path to directory with local media files.

`local/data_dir`
Path to directory to store local metadata such as libraries and playlists in.

`local/playlists_dir`
Path to playlists directory with m3u files for local media.

`local/scan_timeout`
Number of milliseconds before giving up scanning a file and moving on to the next file.

`local/scan_follow_symlinks`
If we should follow symlinks found in *`local/media_dir`*

`local/scan_flush_threshold`
Number of tracks to wait before telling library it should try and store its progress so far. Some libraries might not respect this setting. Set this to zero to disable flushing.

`local/excluded_file_extensions`
File extensions to exclude when scanning the media directory. Values should be separated by either comma or newline.

CHAPTER 7

Mopidy-File

Mopidy-File is an extension for playing music from your local music archive. It is bundled with Mopidy and enabled by default. It allows you to browse through your local file system. Only files that are considered playable will be shown.

This backend handles URIs starting with `file:`.

7.1 Configuration

See *Configuration* for general help on configuring Mopidy.

```
[file]
enabled = true
media_dirs =
    $XDG_MUSIC_DIR|Music
    ~/|Home
show_dotfiles = false
follow_symlinks = false
metadata_timeout = 1000
```

file/enabled
If the file extension should be enabled or not.

file/media_dirs
A list of directories to be browsable. Optionally the path can be followed by `|` and a name that will be shown for that path.

file/show_dotfiles
Whether to show hidden files and directories that start with a dot. Default is false.

file/follow_symlinks
Whether to follow symbolic links found in `files/media_dir`. Directories and files that are outside the configured directories will not be shown. Default is false.

`file/metadata_timeout`
Number of milliseconds before giving up scanning a file and moving on to the next file. Reducing the value might speed up the directory listing, but can lead to some tracks not being shown.

CHAPTER 8

Mopidy-M3U

Mopidy-M3U is an extension for reading and writing M3U playlists stored on disk. It is bundled with Mopidy and enabled by default.

This backend handles URIs starting with `m3u:`.

8.1 Migrating from Mopidy-Local playlists

Mopidy-M3U was split out of the Mopidy-Local extension in Mopidy 1.0. To migrate your playlists from Mopidy-Local, simply move them from the *`local/playlists_dir`* directory to the *`m3u/playlists_dir`* directory. Assuming you have not changed the default config, run the following commands to migrate:

```
mkdir -p ~/.local/share/mopidy/m3u/
mv ~/.local/share/mopidy/local/playlists/* ~/.local/share/mopidy/m3u/
```

8.2 Editing playlists

There is a core playlist API in place for editing playlists. This is supported by a few Mopidy clients, but not through Mopidy's MPD server yet.

It is possible to edit playlists by editing the M3U files located in the *`m3u/playlists_dir`* directory, usually `~/.local/share/mopidy/m3u/`, by hand with a text editor. See Wikipedia for a short description of the quite simple M3U playlist format.

8.3 Configuration

See *Configuration* for general help on configuring Mopidy.

```
[m3u]
enabled = true
playlists_dir =
```

`m3u/enabled`
If the M3U extension should be enabled or not.

`m3u/playlists_dir`
Path to directory with M3U files. Unset by default, in which case the extension's data dir is used to store playlists.

CHAPTER 9

Mopidy-Stream

Mopidy-Stream is an extension for playing streaming music. It is bundled with Mopidy and enabled by default.

This backend does not provide a library or playlist storage. It simply accepts any URI added to Mopidy's tracklist that matches any of the protocols in the *stream/protocols* config value. It then tries to retrieve metadata and play back the URI using GStreamer. For example, if you're using an MPD client, you'll just have to find your clients "add URI" interface, and provide it with the URI of a stream.

In addition to playing streams, the extension also understands how to extract streams from a lot of playlist formats. This is convenient as most Internet radio stations links to playlists instead of directly to the radio streams.

If you're having trouble playing back a stream, run the `mopidy deps` command to check if you have all relevant GStreamer plugins installed.

9.1 Configuration

See *Configuration* for general help on configuring Mopidy.

```
[stream]
enabled = true
protocols =
    http
    https
    mms
    rtmp
    rtmps
    rtsp
timeout = 5000
metadata_blacklist =
```

stream/enabled
If the stream extension should be enabled or not.

stream/protocols
Whitelist of URI schemas to allow streaming from. Values should be separated by either comma or newline.

`stream/timeout`
Number of milliseconds before giving up looking up stream metadata.

`stream/metadata_blacklist`
List of URI globs to not fetch metadata from before playing. This feature is typically needed for play once URIs provided by certain streaming providers. Regular POSIX glob semantics apply, so `http://*.example.com/*` would match all example.com sub-domains.

CHAPTER 10

Mopidy-HTTP

Mopidy-HTTP is an extension that lets you control Mopidy through HTTP and WebSockets, for example from a web client. It is bundled with Mopidy and enabled by default.

When it is enabled it starts a web server at the port specified by the `http/port` config value.

Warning: As a simple security measure, the web server is by default only available from localhost. To make it available from other computers, change the `http/hostname` config value. Before you do so, note that the HTTP extension does not feature any form of user authentication or authorization. Anyone able to access the web server can use the full core API of Mopidy. Thus, you probably only want to make the web server available from your local network or place it behind a web proxy which takes care or user authentication. You have been warned.

10.1 Hosting web clients

Mopidy-HTTP's web server can also host Tornado apps or any static files, for example the HTML, CSS, JavaScript, and images needed for a web based Mopidy client. See *HTTP server side API* for how to make static files or server-side functionality from a Mopidy extension available through Mopidy's web server.

If you're making a web based client and want to do server side development using some other technology than Tornado, you are of course free to run your own web server and just use Mopidy's web server to host the API endpoints. But, for clients implemented purely in JavaScript, letting Mopidy host the files is a simpler solution.

See *HTTP JSON-RPC API* for details on how to integrate with Mopidy over HTTP. If you're looking for a web based client for Mopidy, go check out *HTTP clients*.

10.2 Configuration

See *Configuration* for general help on configuring Mopidy.

```
[http]
enabled = true
hostname = 127.0.0.1
port = 6680
static_dir =
zeroconf = Mopidy HTTP server on $hostname
```

`http/enabled`
If the HTTP extension should be enabled or not.

`http/hostname`
Which address the HTTP server should bind to.

`127.0.0.1` Listens only on the IPv4 loopback interface

`::1` Listens only on the IPv6 loopback interface

`0.0.0.0` Listens on all IPv4 interfaces

`::` Listens on all interfaces, both IPv4 and IPv6

`http/port`
Which TCP port the HTTP server should listen to.

`http/static_dir`
Deprecated: This config is deprecated and will be removed in a future version of Mopidy.

Which directory the HTTP server should serve at "/"

Change this to have Mopidy serve e.g. files for your JavaScript client. "/mopidy" will continue to work as usual even if you change this setting, but any other Mopidy webclient installed with pip to be served at "/ext_name" will stop working if you set this config.

You're strongly encouraged to make Mopidy extensions which use the the *HTTP server side API* to host static files on Mopidy's web server instead of using `http/static_dir`. That way, installation of your web client will be a lot easier for your end users, and multiple web clients can easily share the same web server.

`http/zeroconf`
Name of the HTTP service when published through Zeroconf. The variables `$hostname` and `$port` can be used in the name.

If set, the Zeroconf services `_http._tcp` and `_mopidy-http._tcp` will be published.

Set to an empty string to disable Zeroconf for HTTP.

CHAPTER 11

Mopidy-MPD

Mopidy-MPD is an extension that provides a full MPD server implementation to make Mopidy available to *MPD clients*. It is bundled with Mopidy and enabled by default.

> **Warning:** As a simple security measure, the MPD server is by default only available from localhost. To make it available from other computers, change the `mpd/hostname` config value. Before you do so, note that the MPD server does not support any form of encryption and only a single clear text password (see `mpd/password`) for weak authentication. Anyone able to access the MPD server can control music playback on your computer. Thus, you probably only want to make the MPD server available from your local network. You have been warned.

MPD stands for Music Player Daemon, which is also the name of the original MPD server project. Mopidy does not depend on the original MPD server, but implements the MPD protocol itself, and is thus compatible with clients for the original MPD server.

For more details on our MPD server implementation, see `mopidy.mpd`.

11.1 Limitations

This is a non exhaustive list of MPD features that Mopidy doesn't support. Items on this list will probably not be supported in the near future.

- Only a single password is supported. It gives all-or-nothing access.
- Toggling of audio outputs is not supported
- Channels for client-to-client communication are not supported
- Stickers are not supported
- Crossfade is not supported
- Replay gain is not supported
- `stats` does not provide any statistics

- `decoders` does not provide information about available decoders

The following items are currently not supported, but should be added in the near future:

- Modifying stored playlists is not supported
- `tagtypes` is not supported
- Live update of the music database is not supported

11.2 Configuration

See *Configuration* for general help on configuring Mopidy.

```
[mpd]
enabled = true
hostname = 127.0.0.1
port = 6600
password =
max_connections = 20
connection_timeout = 60
zeroconf = Mopidy MPD server on $hostname
command_blacklist = listall,listallinfo
```

`mpd/enabled`
If the MPD extension should be enabled or not.

`mpd/hostname`
Which address the MPD server should bind to.

`127.0.0.1` Listens only on the IPv4 loopback interface

`::1` Listens only on the IPv6 loopback interface

`0.0.0.0` Listens on all IPv4 interfaces

`::` Listens on all interfaces, both IPv4 and IPv6

`mpd/port`
Which TCP port the MPD server should listen to.

`mpd/password`
The password required for connecting to the MPD server. If blank, no password is required.

`mpd/max_connections`
The maximum number of concurrent connections the MPD server will accept.

`mpd/connection_timeout`
Number of seconds an MPD client can stay inactive before the connection is closed by the server.

`mpd/zeroconf`
Name of the MPD service when published through Zeroconf. The variables `$hostname` and `$port` can be used in the name.

Set to an empty string to disable Zeroconf for MPD.

`mpd/command_blacklist`
List of MPD commands which are disabled by the server. By default this setting blacklists `listall` and `listallinfo`. These commands don't fit well with many of Mopidy's backends and are better left disabled unless you know what you are doing.

CHAPTER 12

Mopidy-SoftwareMixer

Mopidy-SoftwareMixer is an extension for controlling audio volume in software through GStreamer. It is the only mixer bundled with Mopidy and is enabled by default.

If you use PulseAudio, the software mixer will control the per-application volume for Mopidy in PulseAudio, and any changes to the per-application volume done from outside Mopidy will be reflected by the software mixer.

If you don't use PulseAudio, the mixer will adjust the volume internally in Mopidy's GStreamer pipeline.

12.1 Configuration

Multiple mixers can be installed and enabled at the same time, but only the mixer pointed to by the *`audio/mixer`* config value will actually be used.

See *Configuration* for general help on configuring Mopidy.

```
[softwaremixer]
enabled = true
```

`softwaremixer/enabled`
If the software mixer should be enabled or not. Usually you don't want to change this, but instead change the *`audio/mixer`* config value to decide which mixer is actually used.

CHAPTER 13

Mixer extensions

Here you can find a list of external packages that extend Mopidy with additional audio mixers by implementing the *mopidy.mixer — Audio mixer API* which was added in Mopidy 0.19.

This list is moderated and updated on a regular basis. If you want your package to show up here, follow the *guide on creating extensions*.

13.1 Mopidy-ALSAMixer

https://github.com/mopidy/mopidy-alsamixer

Extension for controlling volume on a Linux system using ALSA.

13.2 Mopidy-Arcam

https://github.com/TooDizzy/mopidy-arcam

Extension for controlling volume using an external Arcam amplifier. Developed and tested with an Arcam AVR-300.

13.3 Mopidy-dam1021

https://github.com/fortaa/mopidy-dam1021

Extension for controlling volume using a dam1021 DAC device.

13.4 Mopidy-NAD

https://github.com/mopidy/mopidy-nad

Extension for controlling volume using an external NAD amplifier. Developed and tested with a NAD C355BEE.

13.5 Mopidy-SoftwareMixer

Bundled with Mopidy. See *Mopidy-SoftwareMixer*.

13.6 Mopidy-Yamaha

https://github.com/knutz3n/mopidy-yamaha

Extension for controlling volume using an external Yamaha network connected amplifier.

CHAPTER 14

Backend extensions

Here you can find a list of external packages that extend Mopidy with additional music sources by implementing the *mopidy.backend — Backend API*.

This list is moderated and updated on a regular basis. If you want your package to show up here, follow the *guide on creating extensions*.

14.1 Mopidy-AudioAddict

https://github.com/nilicule/mopidy-audioaddict

Provides a backend for playing music from the AudioAddict network of sites, including Digitally Imported, Radio-Tunes, RockRadio, JazzRadio, and FrescaRadio.

14.2 Mopidy-Banshee

https://github.com/tamland/mopidy-banshee

Provides a backend for playing music from the Banshee music player's music library.

14.3 Mopidy-Bassdrive

https://github.com/felixb/mopidy-Bassdrive

Provides a backend for playing radio streams from BassDrive.

14.4 Mopidy-Beets

https://github.com/mopidy/mopidy-beets

Provides a backend for playing music from your Beets music library through Beets' web extension.

14.5 Mopidy-Dirble

https://github.com/mopidy/mopidy-dirble

Provides a backend for browsing the Internet radio channels from the Dirble directory.

14.6 Mopidy-dLeyna

https://github.com/tkem/mopidy-dleyna

Provides a backend for playing music from Digital Media Servers using the dLeyna D-Bus interface.

14.7 Mopidy-File

Bundled with Mopidy. See *Mopidy-File*.

14.8 Mopidy-Grooveshark

https://github.com/camilonova/mopidy-grooveshark

Provides a backend for playing music from Grooveshark.

14.9 Mopidy-GMusic

https://github.com/hechtus/mopidy-gmusic

Provides a backend for playing music from Google Play Music.

14.10 Mopidy-InternetArchive

https://github.com/tkem/mopidy-internetarchive

Extension for playing music and audio from the Internet Archive.

14.11 Mopidy-LeftAsRain

https://github.com/naglis/mopidy-leftasrain

Extension for playing music from the leftasrain.com music blog.

14.12 Mopidy-Local

Bundled with Mopidy. See *Mopidy-Local*.

14.13 Mopidy-Local-Images

https://github.com/tkem/mopidy-local-images

Extension which plugs into Mopidy-Local to allow Web clients access to album art embedded in local media files. Not to be used on its own, but acting as a proxy between `mopidy local scan` and the actual local library provider being used.

14.14 Mopidy-Local-SQLite

https://github.com/tkem/mopidy-local-sqlite

Extension which plugs into Mopidy-Local to use an SQLite database to keep track of your local media. This extension lets you browse your music collection by album, artist, composer and performer, and provides full-text search capabilities based on SQLite's FTS modules. It also notices updates via `mopidy local scan` while Mopidy is running, so you can scan your media library periodically from a cron job, for example.

14.15 Mopidy-OE1

https://github.com/tischlda/mopidy-oe1

Extension for playing the live stream and browsing the 7-day archive of the Austrian radio station OE1.

14.16 Mopidy-Podcast

https://github.com/tkem/mopidy-podcast

Extension for browsing RSS feeds of podcasts and stream the episodes.

14.17 Mopidy-Podcast-gpodder.net

https://github.com/tkem/mopidy-podcast-gpodder

Extension for Mopidy-Podcast that lets you search and browse podcasts from the gpodder.net web site.

14.18 Mopidy-Podcast-iTunes

https://github.com/tkem/mopidy-podcast-itunes

Extension for Mopidy-Podcast that lets you search and browse podcasts from the Apple iTunes Store.

14.19 Mopidy-radio-de

https://github.com/hechtus/mopidy-radio-de

Extension for listening to Internet radio stations and podcasts listed at radio.de, rad.io, radio.fr, and radio.at.

14.20 Mopidy-SomaFM

https://github.com/AlexandrePTJ/mopidy-somafm

Provides a backend for playing music from the SomaFM service.

14.21 Mopidy-SoundCloud

https://github.com/mopidy/mopidy-soundcloud

Provides a backend for playing music from the SoundCloud service.

14.22 Mopidy-Spotify

https://github.com/mopidy/mopidy-spotify

Extension for playing music from the Spotify music streaming service.

14.23 Mopidy-Spotify-Tunigo

https://github.com/trygveaa/mopidy-spotify-tunigo

Extension for providing the browse feature of Spotify. This lets you browse playlists, genres and new releases.

14.24 Mopidy-Stream

Bundled with Mopidy. See *Mopidy-Stream*.

14.25 Mopidy-Subsonic

https://github.com/rattboi/mopidy-subsonic

Provides a backend for playing music from a Subsonic Music Streamer library.

14.26 Mopidy-TuneIn

https://github.com/kingosticks/mopidy-tunein

Provides a backend for playing music from the TuneIn online radio service.

14.27 Mopidy-VKontakte

https://github.com/sibuser/mopidy-vkontakte

Provides a backend for playing music from the VKontakte social network.

14.28 Mopidy-YouTube

https://github.com/dz0ny/mopidy-youtube

Provides a backend for playing music from the YouTube service.

CHAPTER 15

Frontend extensions

Here you can find a list of external packages that extend Mopidy with additional frontends, which includes just about anything that use the *mopidy.core — Core API*.

This list is moderated and updated on a regular basis. If you want your package to show up here, follow the *guide on creating extensions*.

15.1 Mopidy-EvtDev

https://github.com/liamw9534/mopidy-evtdev

Extension for controll Mopidy from virtual input devices.

15.2 Mopidy-HTTP

Bundled with Mopidy. See *Mopidy-HTTP*.

15.3 Mopidy-MPD

Bundled with Mopidy. See *Mopidy-MPD*.

15.4 Mopidy-MPRIS

https://github.com/mopidy/mopidy-mpris

Extension for controlling Mopidy through the MPRIS D-Bus interface, for example using the Ubuntu Sound Menu.

15.5 Mopidy-Notifier

https://github.com/sauberfred/mopidy-notifier

Extension for displaying track info as User Notifications in Mac OS X.

15.6 Mopidy-Scrobbler

https://github.com/mopidy/mopidy-scrobbler

Extension for scrobbling played tracks to Last.fm.

15.7 Mopidy-Touchscreen

https://github.com/9and3r/mopidy-touchscreen

Extension for displaying track info and controlling Mopidy from a touch screen using PyGame/SDL.

15.8 Mopidy-TtsGpio

https://github.com/9and3r/mopidy-ttsgpio

Extension for controlling Mopidy without a display by using e.g. buttons connected to GPIO and text-to-speech for track information.

15.9 Mopidy-Webhooks

https://github.com/paddycarey/mopidy-webhooks

Extension for sending HTTP POST requests with JSON payloads to a remote server when Mopidy core triggers an event and on regular intervals.

CHAPTER 16

Web extensions

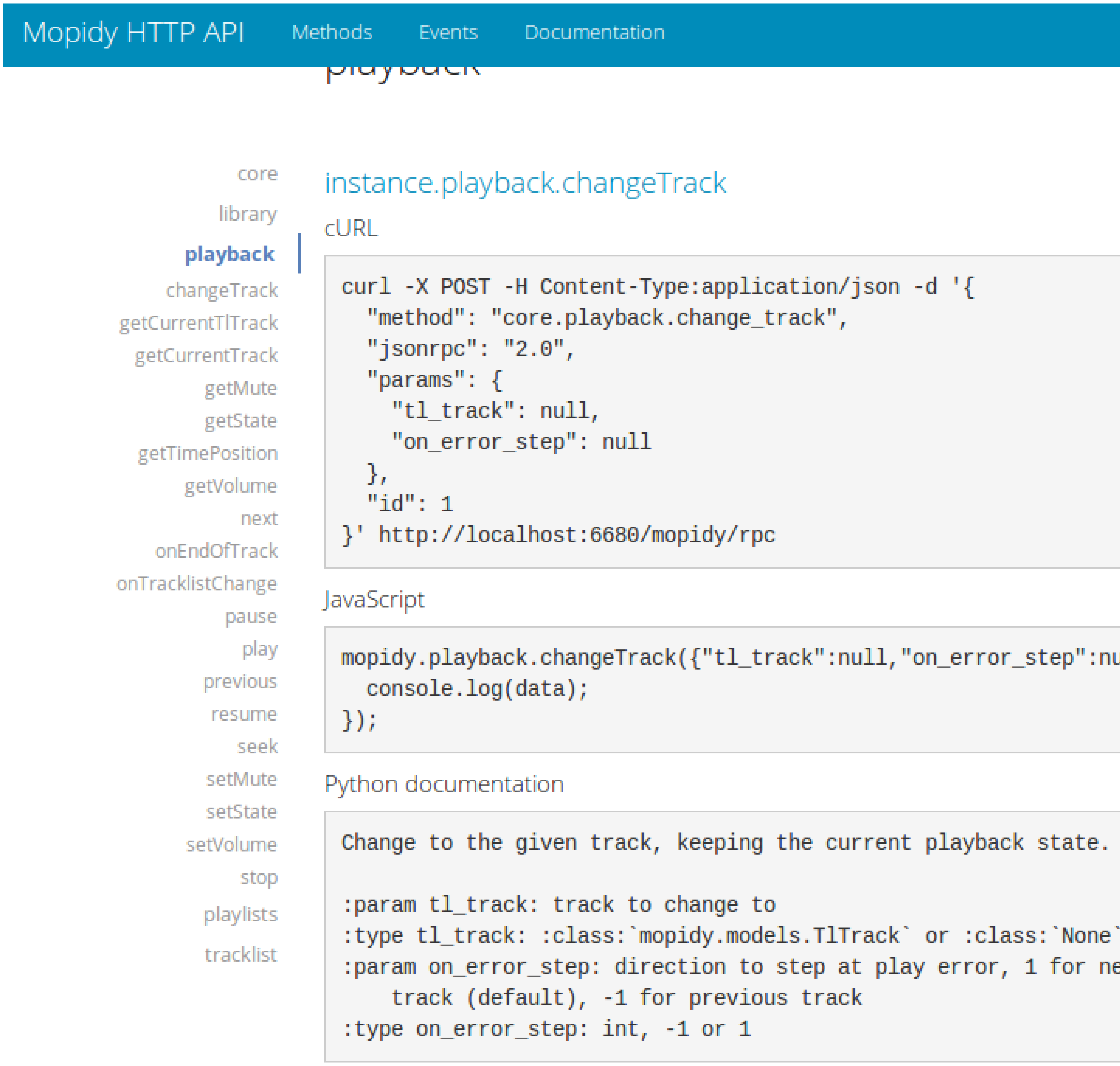
Here you can find a list of external packages that extend Mopidy with additional web interfaces by implementing the *HTTP server side API*, which was added in Mopidy 0.19, and optionally using the *HTTP JSON-RPC API*.

This list is moderated and updated on a regular basis. If you want your package to show up here, follow the *guide on creating extensions*.

16.1 Mopidy-API-Explorer

https://github.com/dz0ny/mopidy-api-explorer

Web extension for browsing the Mopidy HTTP API.



instance.playback.changeTrack

cURL

```
curl -X POST -H Content-Type:application/json -d '{
  "method": "core.playback.change_track",
  "jsonrpc": "2.0",
  "params": {
    "tl_track": null,
    "on_error_step": null
  },
  "id": 1
}' http://localhost:6680/mopidy/rpc
```

JavaScript

```
mopidy.playback.changeTrack({"tl_track":null,"on_error_step":nu
  console.log(data);
});
```

Python documentation

```
Change to the given track, keeping the current playback state.

:param tl_track: track to change to
:type tl_track: :class:`mopidy.models.TlTrack` or :class:`None`
:param on_error_step: direction to step at play error, 1 for ne
    track (default), -1 for previous track
:type on_error_step: int, -1 or 1
```

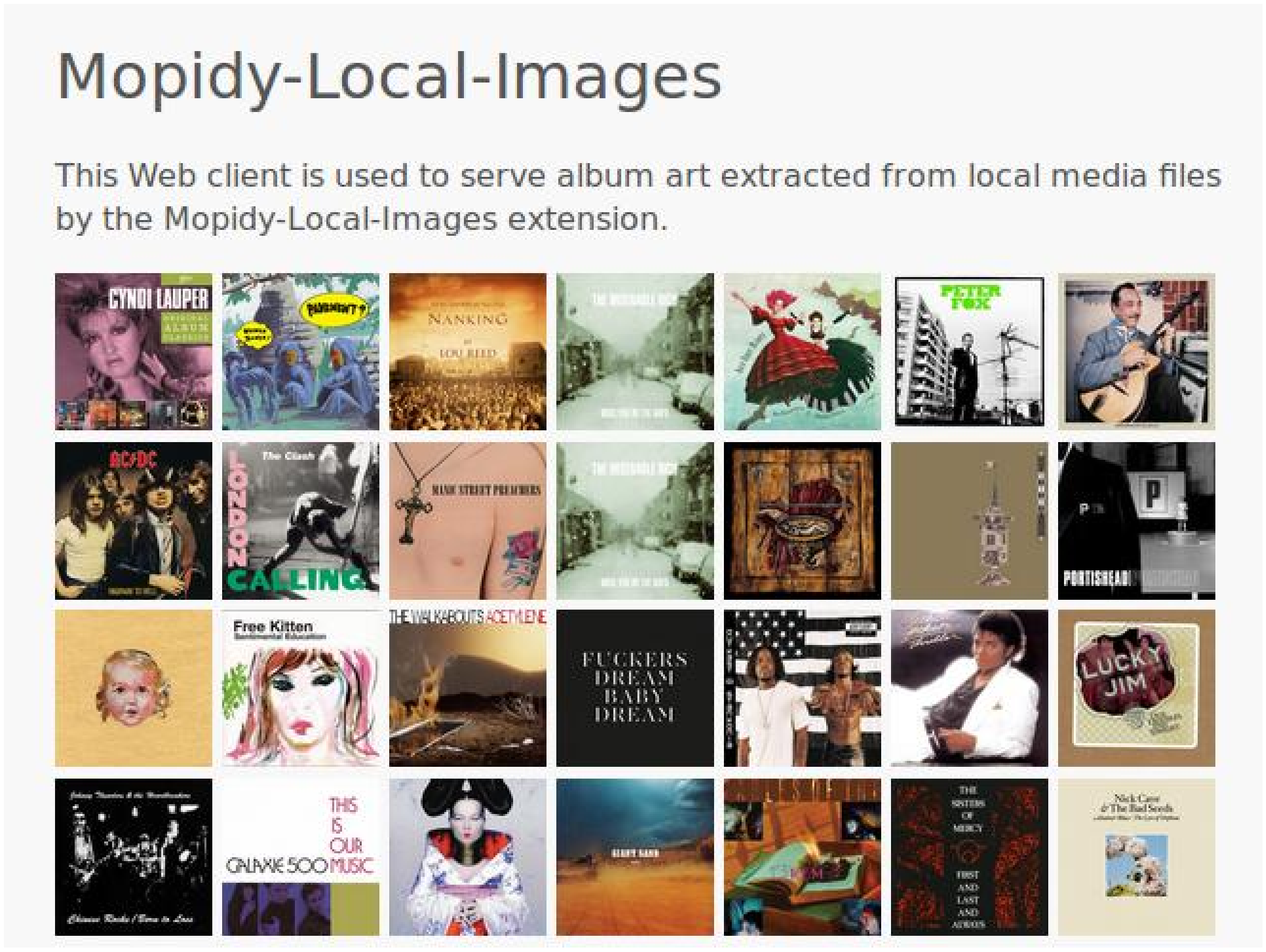
To install, run:

```
pip install Mopidy-API-Explorer
```

16.2 Mopidy-Local-Images

https://github.com/tkem/mopidy-local-images

Not a full-featured web client, but rather a local library and web extension which allows other web clients access to album art embedded in local media files.


Mopidy-Local-Images

This Web client is used to serve album art extracted from local media files by the Mopidy-Local-Images extension.

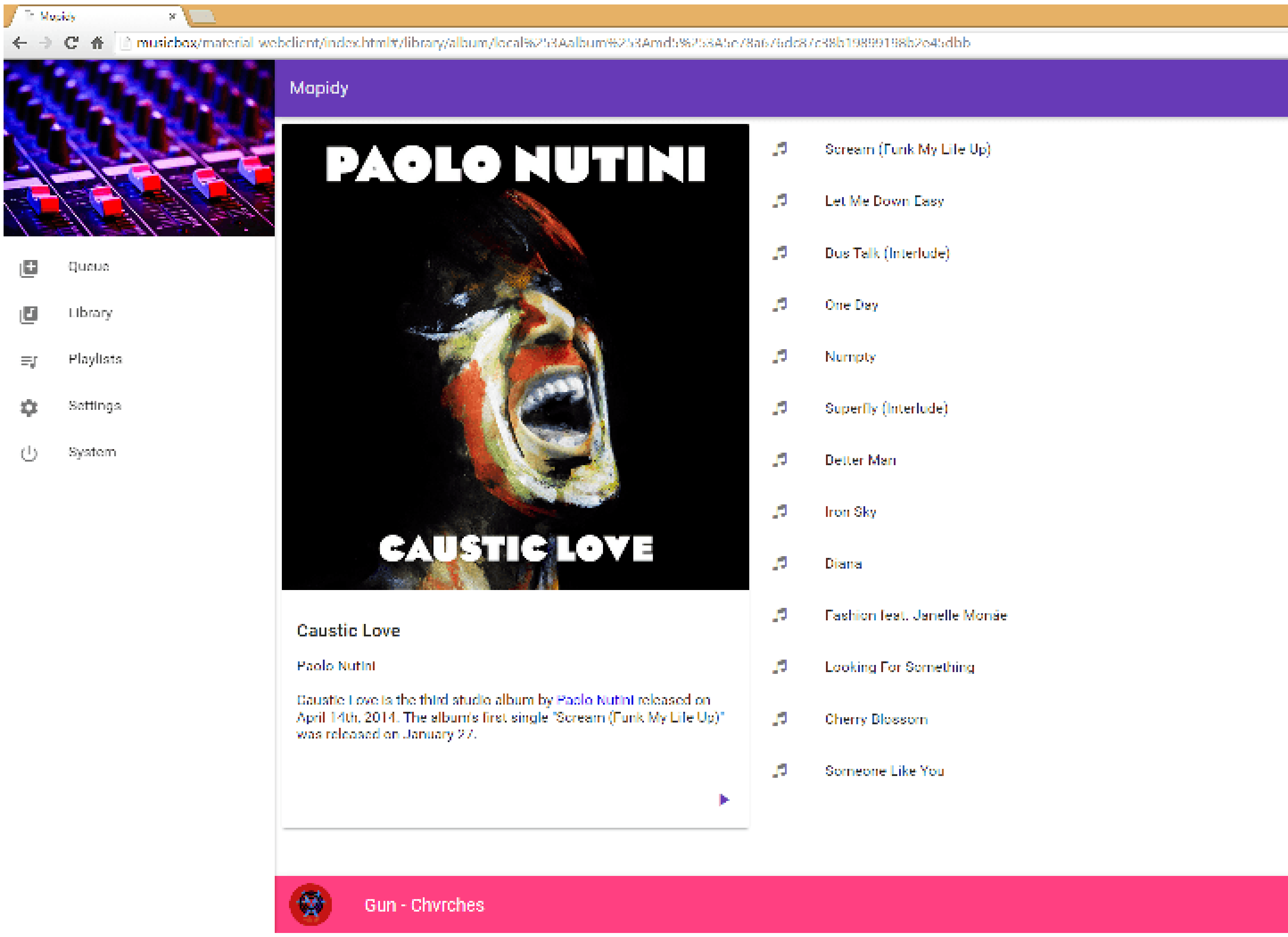
To install, run:

```
pip install Mopidy-Local-Images
```

16.3 Mopidy-Material-Webclient

https://github.com/matgallacher/mopidy-material-webclient

A Mopidy web client with an Android Material design feel.



Caustic Love

Paolo Nutini

Caustic Love is the third studio album by Paolo Nutini released on April 14th, 2014. The album's first single "Scream (Funk My Life Up)" was released on January 27.

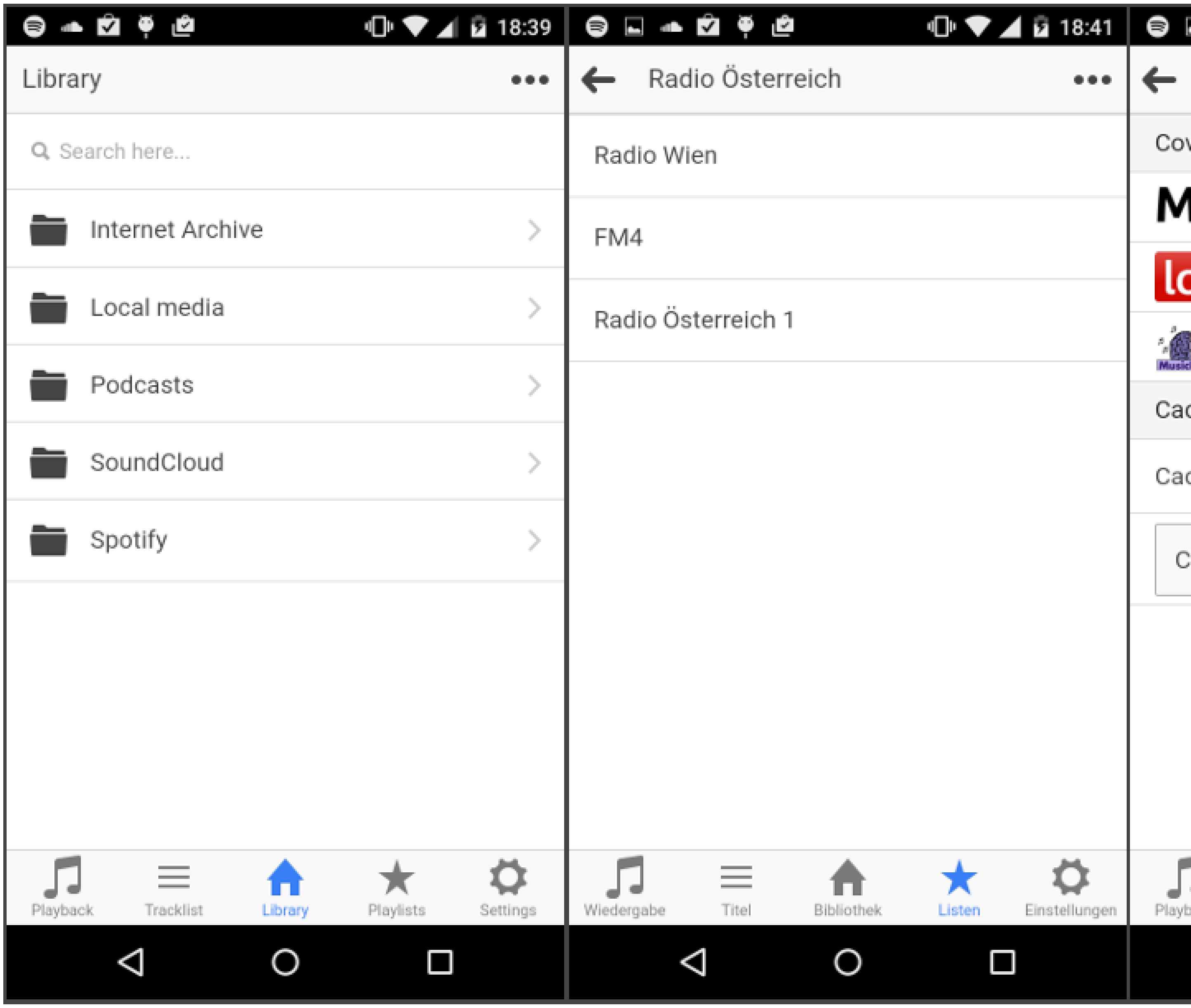
To install, run:

```
pip install Mopidy-Material-Webclient
```

16.4 Mopidy-Mobile

https://github.com/tkem/mopidy-mobile

A Mopidy web client extension and hybrid mobile app, made with Ionic, AngularJS and Apache Cordova by Thomas Kemmer.



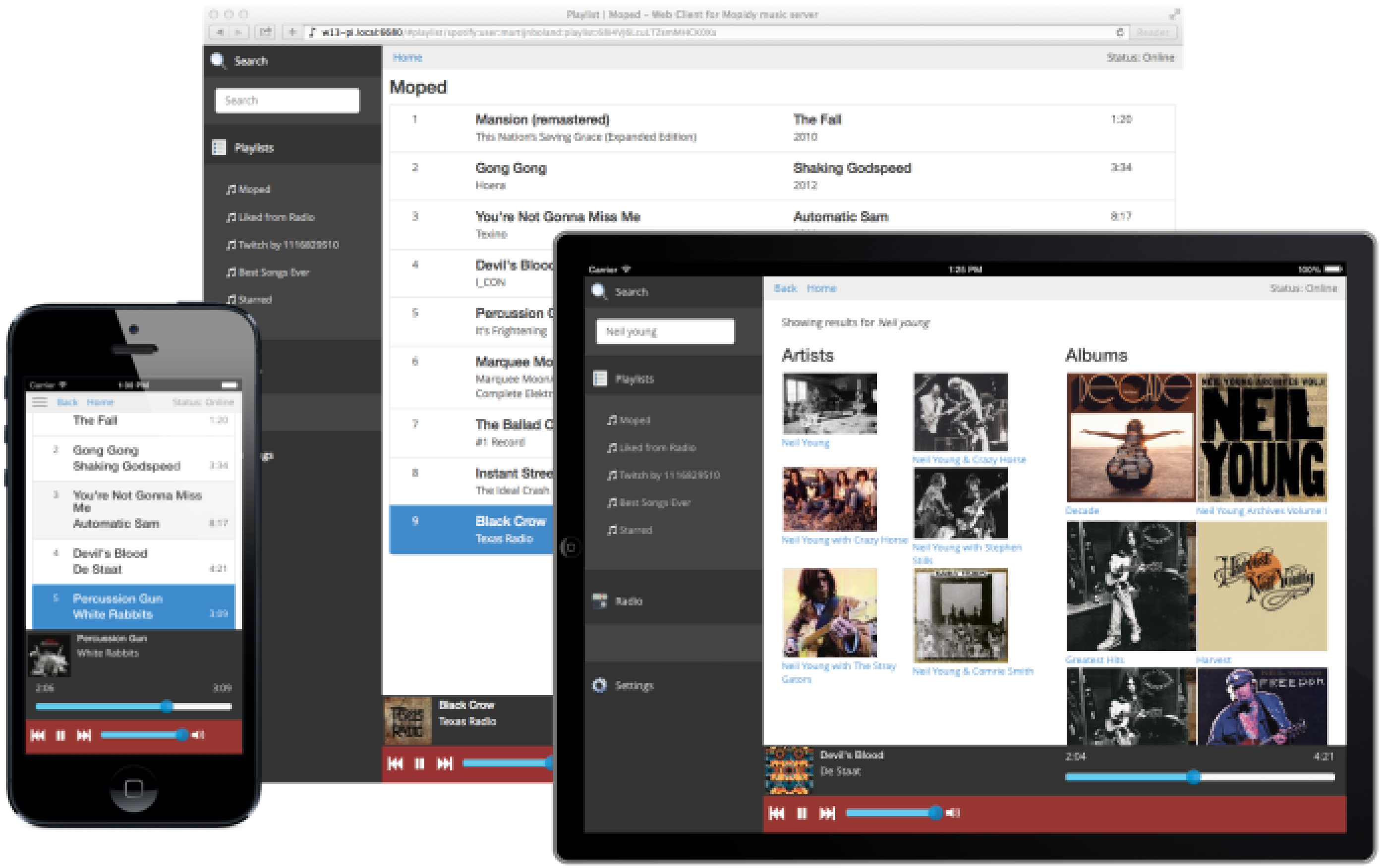
To install, run:

```
pip install Mopidy-Mobile
```

16.5 Mopidy-Moped

https://github.com/martijnboland/moped

A Mopidy web client made with AngularJS by Martijn Boland.



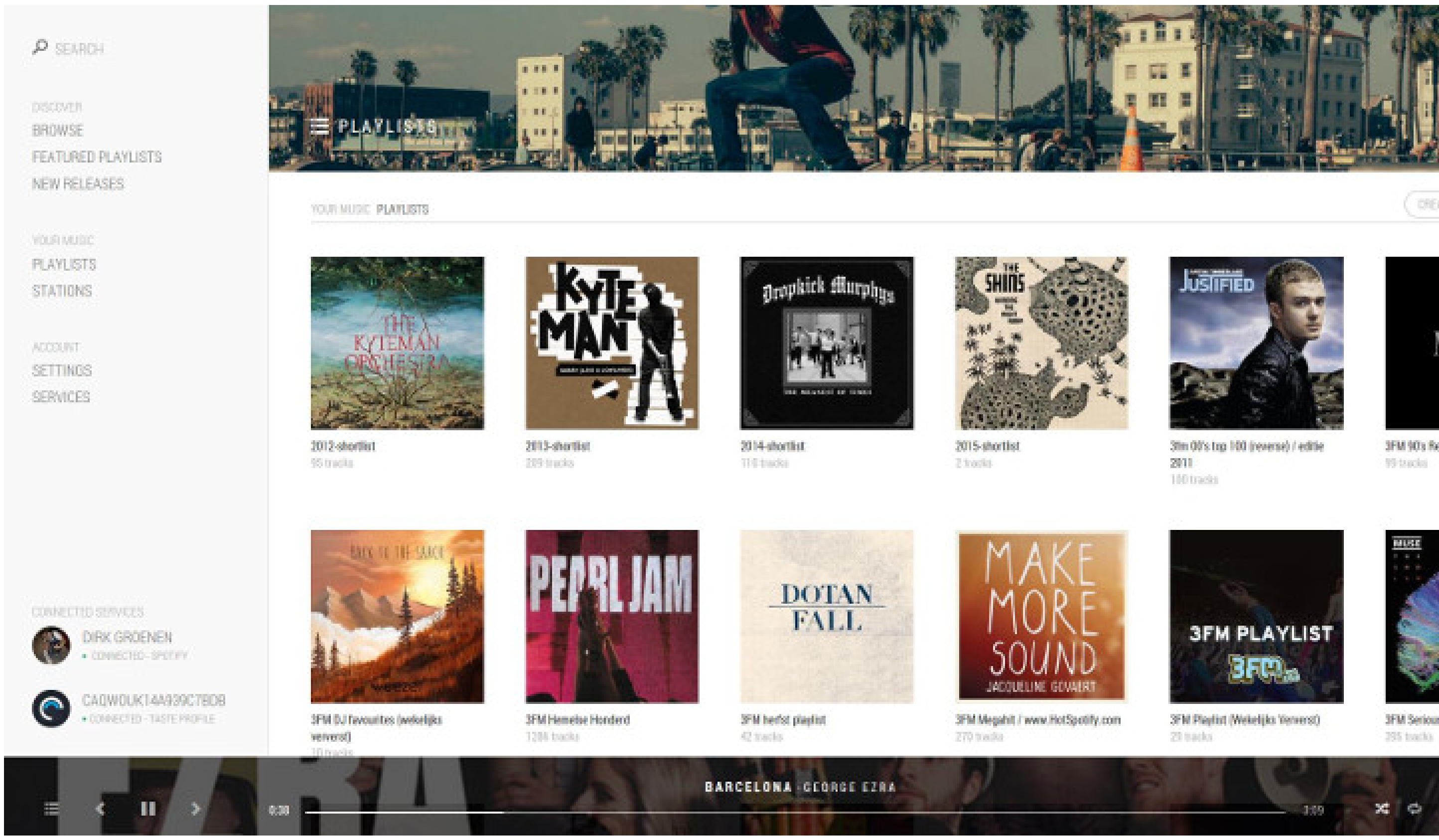
To install, run:

```
pip install Mopidy-Moped
```

16.6 Mopidy-Mopify

https://github.com/dirkgroenen/mopidy-mopify

A web client that uses external web services to provide additional features and a more “complete” Spotify music experience. It’s currently targeted at people using Spotify through Mopidy. Made by Dirk Groenen.



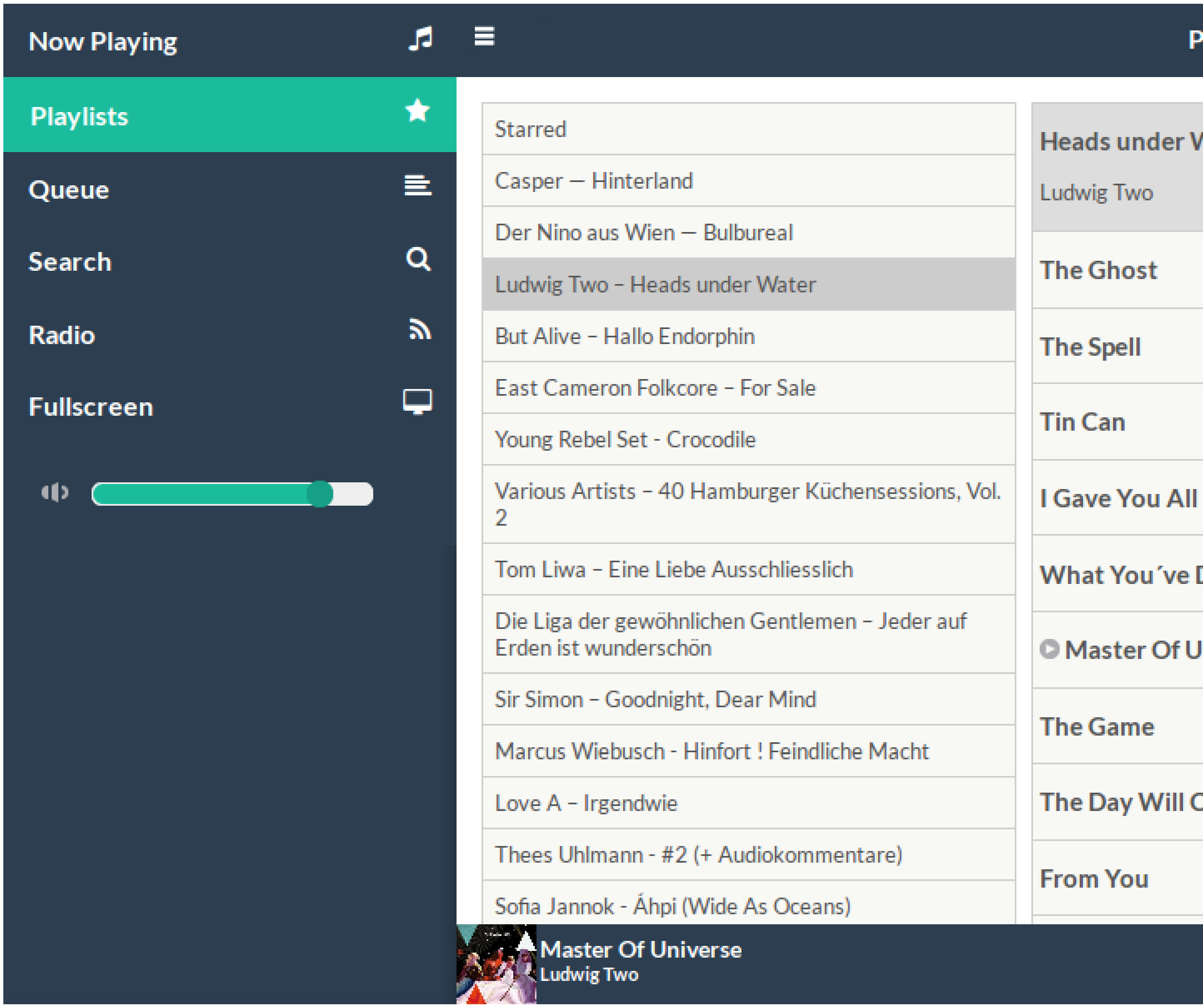
To install, run:

```
pip install Mopidy-Mopify
```

16.7 Mopidy-MusicBox-Webclient

https://github.com/woutervanwijk/Mopidy-MusicBox-Webclient

The first web client for Mopidy, made with jQuery Mobile by Wouter van Wijk. Also the web client used for Wouter's popular Pi Musicbox image for Raspberry Pi.



Starred
Casper — Hinterland
Der Nino aus Wien — Bulbureal
Ludwig Two - Heads under Water
But Alive - Hallo Endorphin
East Cameron Folkcore - For Sale
Young Rebel Set - Crocodile
Various Artists - 40 Hamburger Küchensessions, Vol. 2
Tom Liwa - Eine Liebe Ausschliesslich
Die Liga der gewöhnlichen Gentlemen - Jeder auf Erden ist wunderschön
Sir Simon - Goodnight, Dear Mind
Marcus Wiebusch - Hinfort ! Feindliche Macht
Love A - Irgendwie
Thees Uhlmann - #2 (+ Audiokommentare)
Sofia Jannok - Áhpi (Wide As Oceans)

Heads under V
Ludwig Two
The Ghost
The Spell
Tin Can
I Gave You All
What You´ve D
Master Of U
The Game
The Day Will C
From You

Master Of Universe
Ludwig Two

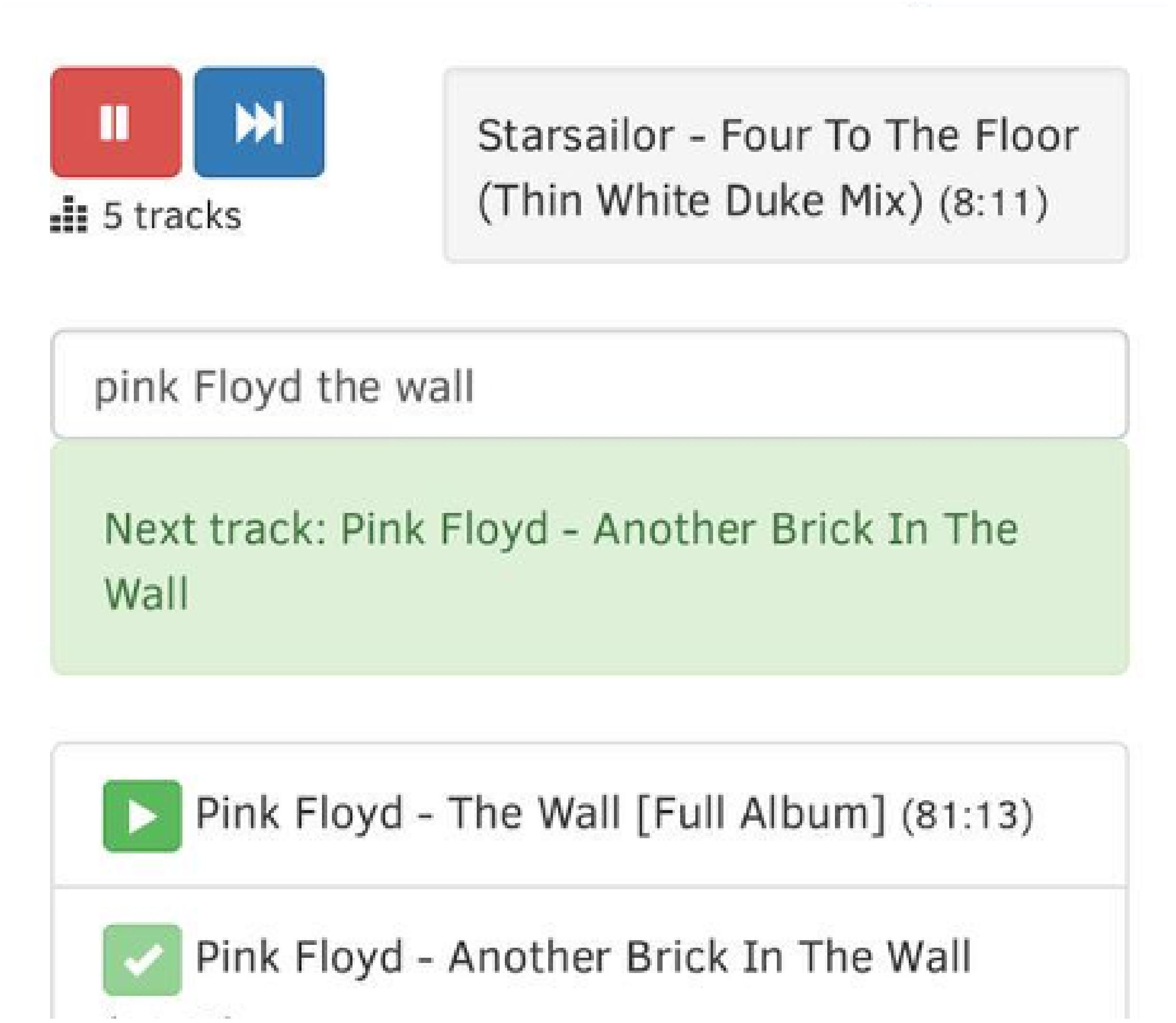
To install, run:

```
pip install Mopidy-MusicBox-Webclient
```

16.8 Mopidy-Party

https://github.com/Lesterpig/mopidy-party

Minimal web client designed for collaborative music management during parties.


5 tracks
Starsailor - Four To The Floor (Thin White Duke Mix) (8:11)
pink Floyd the wall
Next track: Pink Floyd - Another Brick In The Wall
Pink Floyd - The Wall [Full Album] (81:13)
Pink Floyd - Another Brick In The Wall

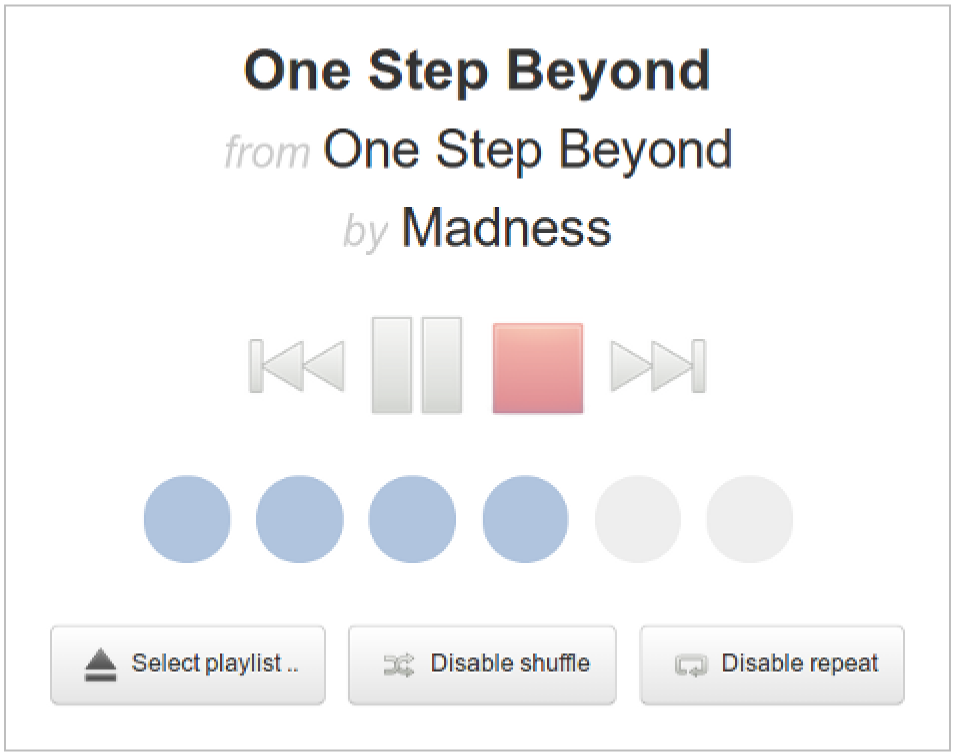
To install, run:

```
pip install Mopidy-Party
```

16.9 Mopidy-Simple-Webclient

https://github.com/xolox/mopidy-simple-webclient

A minimalistic web client targeted for mobile devices. Made with jQuery and Bootstrap by Peter Odding.



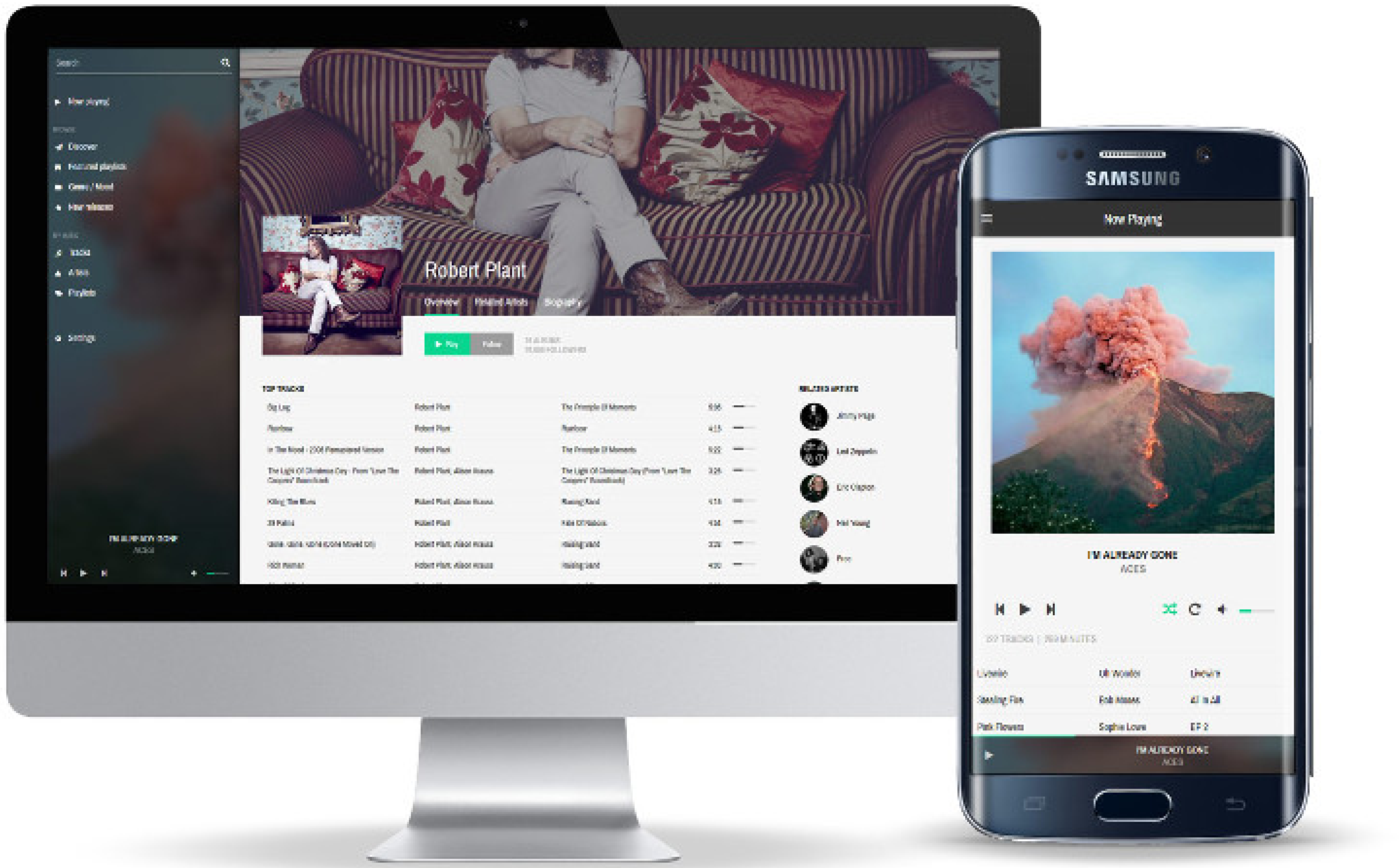
To install, run:

```
pip install Mopidy-Simple-Webclient
```

16.10 Mopidy-Spotmop

https://github.com/jaedb/spotmop

A client targeted at Spotify users. Made by James Barnsley.



To install, run:

```
pip install Mopidy-Spotmop
```

16.11 Mopidy-WebSettings

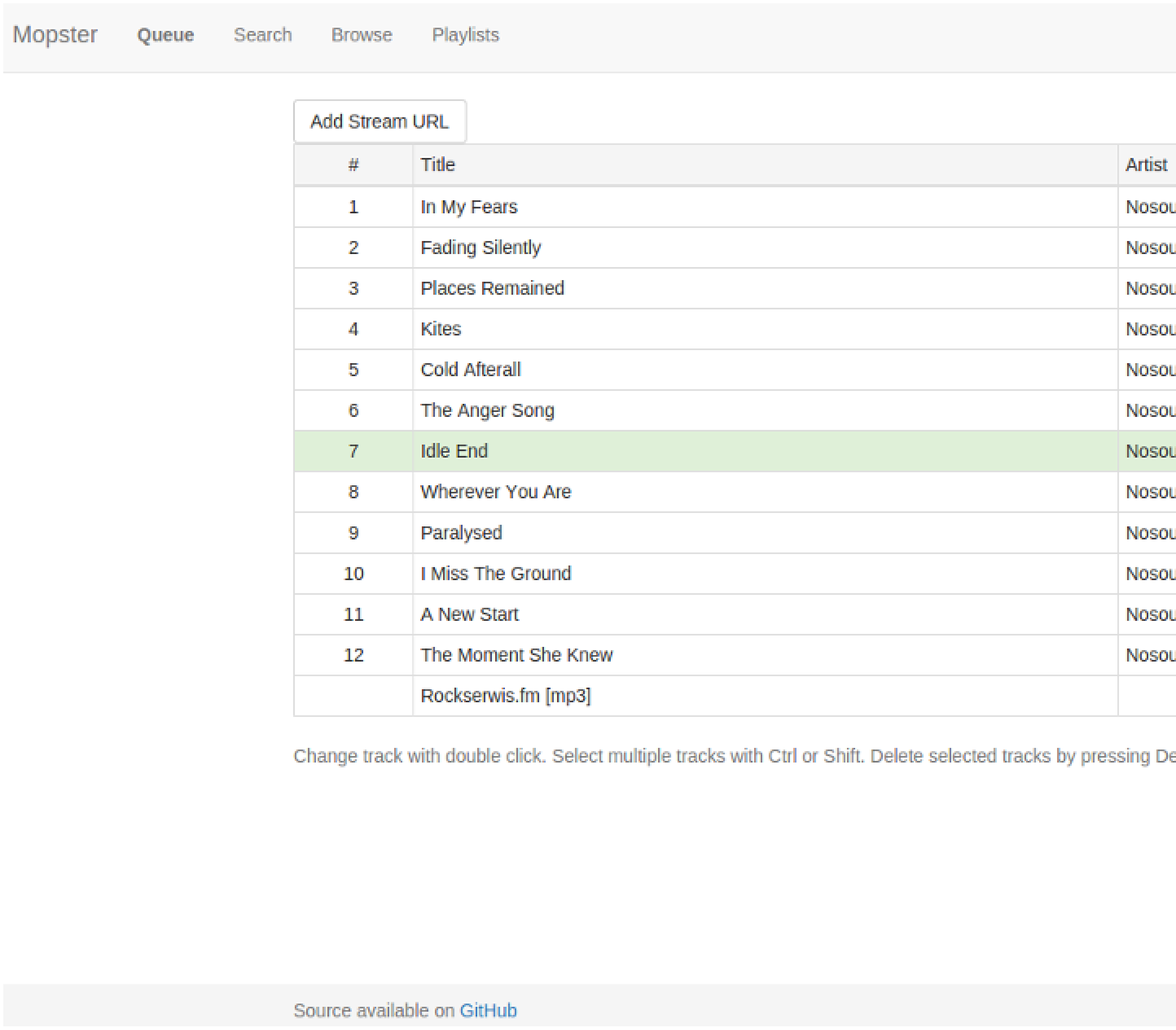
https://github.com/woutervanwijk/mopidy-websettings

A web extension for changing settings. Used by the Pi MusicBox distribution for Raspberry Pi, but also usable for other projects.

16.12 Mopster

https://github.com/cowbell/mopster

Simple web client hosted online written in Ember.js and styled using basic Bootstrap by Wojciech Wnętrzak.



#	Title	Artist
1	In My Fears	Nosou
2	Fading Silently	Nosou
3	Places Remained	Nosou
4	Kites	Nosou
5	Cold Afterall	Nosou
6	The Anger Song	Nosou
7	Idle End	Nosou
8	Wherever You Are	Nosou
9	Paralysed	Nosou
10	I Miss The Ground	Nosou
11	A New Start	Nosou
12	The Moment She Knew	Nosou
	Rockserwis.fm [mp3]	

Change track with double click. Select multiple tracks with Ctrl or Shift. Delete selected tracks by pressing De

Source available on GitHub

To use, just visit http://mopster.cowbell-labs.com/.

16.13 Other web clients

There's also some other web clients for Mopidy that use the *HTTP JSON-RPC API*, but isn't installable using `pip`:

- Apollo Player
- JukePi

In addition, there's several web based MPD clients, which doesn't use the *Mopidy-HTTP* frontend at all, but connect to Mopidy through our *Mopidy-MPD* frontend. For a list of those, see *MPD web clients*.

CHAPTER 17

HTTP clients

See *Web extensions*.

CHAPTER 18

MPD clients

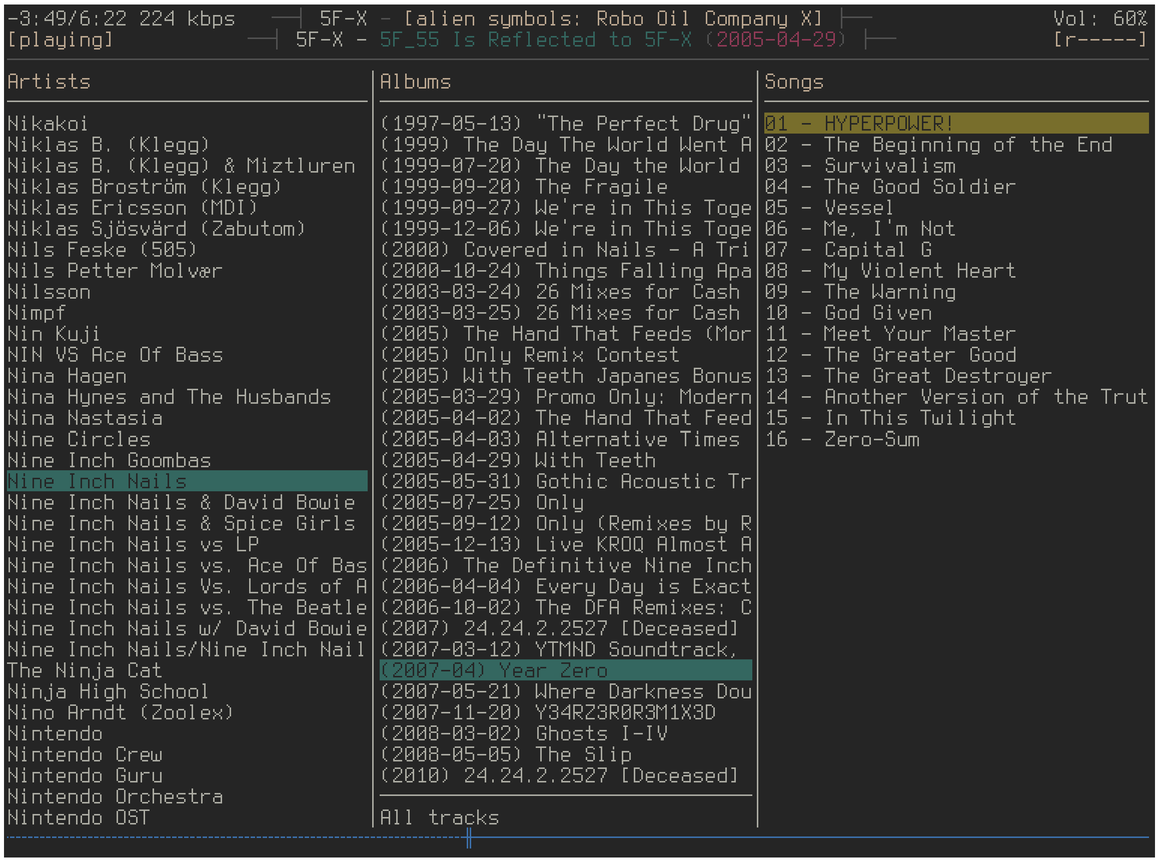
This is a list of MPD clients we either know works well with Mopidy, or that we know won't work well. For a more exhaustive list of MPD clients, see http://mpd.wikia.com/wiki/Clients.

Contents

18.1 MPD console clients

18.1.1 ncmpcpp

A console client that works well with Mopidy, and is regularly used by Mopidy developers.



```
-3:49/6:22 224 kbps    ─┤ 5F-X - [alien symbols: Robo Oil Company X] ├─          Vol: 60%
[playing]            ─┤ 5F-X - 5F_55 Is Reflected to 5F-X (2005-04-29) ├─        [r-----]

Artists                       │Albums                         │Songs
                              │                               │
Nikakoi                       │(1997-05-13) "The Perfect Drug"│01 - HYPERPOWER!
Niklas B. (Klegg)             │(1999) The Day The World Went A│02 - The Beginning of the End
Niklas B. (Klegg) & Miztluren │(1999-07-20) The Day the World │03 - Survivalism
Niklas Broström (Klegg)       │(1999-09-20) The Fragile       │04 - The Good Soldier
Niklas Ericsson (MDI)         │(1999-09-27) We're in This Toge│05 - Vessel
Niklas Sjösvärd (Zabutom)     │(1999-12-06) We're in This Toge│06 - Me, I'm Not
Nils Feske (505)              │(2000) Covered in Nails - A Tri│07 - Capital G
Nils Petter Molvær            │(2000-10-24) Things Falling Apa│08 - My Violent Heart
Nilsson                       │(2003-03-24) 26 Mixes for Cash │09 - The Warning
Nimpf                         │(2003-03-25) 26 Mixes for Cash │10 - God Given
Nin Kuji                      │(2005) The Hand That Feeds (Mor│11 - Meet Your Master
NIN VS Ace Of Bass            │(2005) Only Remix Contest      │12 - The Greater Good
Nina Hagen                    │(2005) With Teeth Japanes Bonus│13 - The Great Destroyer
Nina Hynes and The Husbands   │(2005-03-29) Promo Only: Modern│14 - Another Version of the Trut
Nina Nastasia                 │(2005-04-02) The Hand That Feed│15 - In This Twilight
Nine Circles                  │(2005-04-03) Alternative Times │16 - Zero-Sum
Nine Inch Goombas             │(2005-04-29) With Teeth        │
Nine Inch Nails               │(2005-05-31) Gothic Acoustic Tr│
Nine Inch Nails & David Bowie │(2005-07-25) Only              │
Nine Inch Nails & Spice Girls │(2005-09-12) Only (Remixes by R│
Nine Inch Nails vs LP         │(2005-12-13) Live KROQ Almost A│
Nine Inch Nails vs. Ace Of Bas│(2006) The Definitive Nine Inch│
Nine Inch Nails Vs. Lords of A│(2006-04-04) Every Day is Exact│
Nine Inch Nails vs. The Beatle│(2006-10-02) The DFA Remixes: C│
Nine Inch Nails w/ David Bowie│(2007) 24.24.2.2527 [Deceased] │
Nine Inch Nails/Nine Inch Nail│(2007-03-12) YTMND Soundtrack, │
The Ninja Cat                 │(2007-04) Year Zero            │
Ninja High School             │(2007-05-21) Where Darkness Dou│
Nino Arndt (Zoolex)           │(2007-11-20) Y34RZ3R0R3M1X3D   │
Nintendo                      │(2008-03-02) Ghosts I-IV       │
Nintendo Crew                 │(2008-05-05) The Slip          │
Nintendo Guru                 │(2010) 24.24.2.2527 [Deceased] │
Nintendo Orchestra            │                               │
Nintendo OST                  │All tracks                     │
```

Search does not work in the "Match if tag contains search phrase (regexes supported)" mode because the client tries to fetch all known metadata and do the search on the client side. The two other search modes works nicely, so this is not a problem.

The library view is very slow when used together with Mopidy-Spotify. A workaround is to edit the ncmpcpp configuration file (`~/.ncmpcpp/config`) and set:

```
media_library_display_date = "no"
```

With this change ncmpcpp's library view will still be a bit slow, but usable.

18.1.2 ncmpc

A console client. Works with Mopidy 0.6 and upwards. Uses the `idle` MPD command, but in a resource inefficient way.

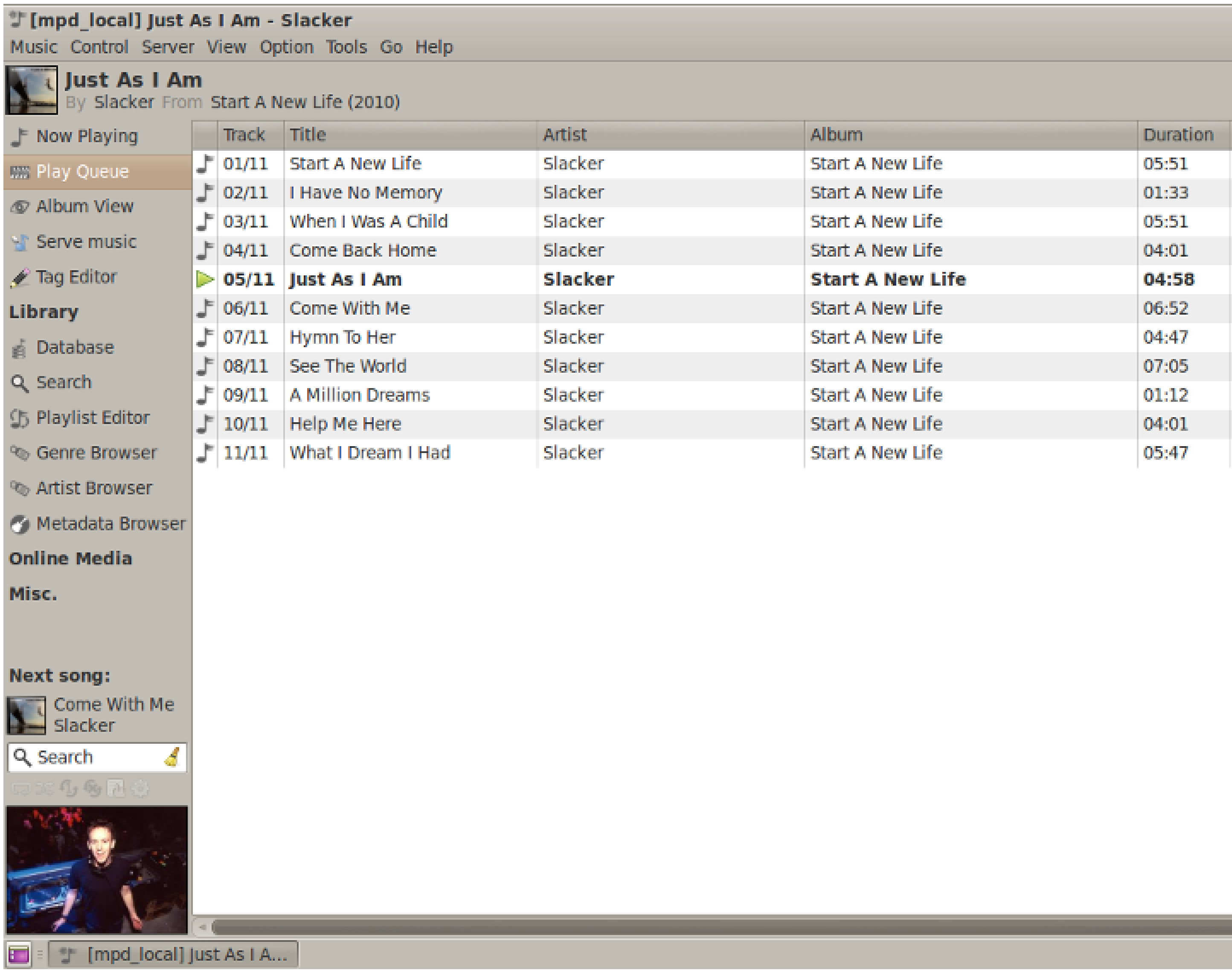
18.1.3 mpc

A command line client. Version 0.16 and upwards seems to work nicely with Mopidy.

18.2 MPD graphical clients

18.2.1 GMPC

GMPC is a graphical MPD client (GTK+) which works well with Mopidy.



Just As I Am
By Slacker From Start A New Life (2010)

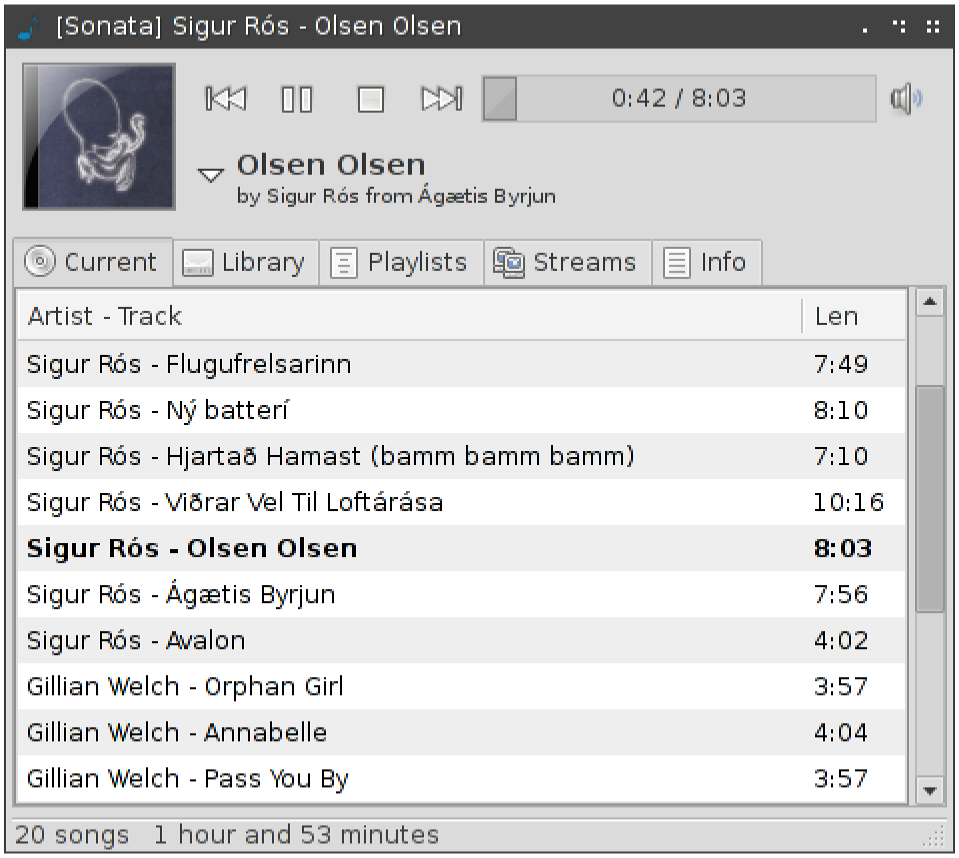
Track	Title	Artist	Album	Duration
01/11	Start A New Life	Slacker	Start A New Life	05:51
02/11	I Have No Memory	Slacker	Start A New Life	01:33
03/11	When I Was A Child	Slacker	Start A New Life	05:51
04/11	Come Back Home	Slacker	Start A New Life	04:01
05/11	**Just As I Am**	**Slacker**	**Start A New Life**	**04:58**
06/11	Come With Me	Slacker	Start A New Life	06:52
07/11	Hymn To Her	Slacker	Start A New Life	04:47
08/11	See The World	Slacker	Start A New Life	07:05
09/11	A Million Dreams	Slacker	Start A New Life	01:12
10/11	Help Me Here	Slacker	Start A New Life	04:01
11/11	What I Dream I Had	Slacker	Start A New Life	05:47

Come With Me
Slacker

GMPC may sometimes requests a lot of meta data of related albums, artists, etc. This takes more time with Mopidy, which needs to query Spotify for the data, than with a normal MPD server, which has a local cache of meta data. Thus, GMPC may sometimes feel frozen, but usually you just need to give it a bit of slack before it will catch up.

18.2.2 Sonata

Sonata is a graphical MPD client (GTK+). It generally works well with Mopidy, except for search.


[Sonata] Sigur Rós - Olsen Olsen
0:42 / 8:03
Olsen Olsen
by Sigur Rós from Ágætis Byrjun
Current
Library
Playlists
Streams
Info

Artist - Track	Len
Sigur Rós - Flugufrelsarinn	7:49
Sigur Rós - Ný batterí	8:10
Sigur Rós - Hjartað Hamast (bamm bamm bamm)	7:10
Sigur Rós - Viðrar Vel Til Loftárása	10:16
Sigur Rós - Olsen Olsen	**8:03**
Sigur Rós - Ágætis Byrjun	7:56
Sigur Rós - Avalon	4:02
Gillian Welch - Orphan Girl	3:57
Gillian Welch - Annabelle	4:04
Gillian Welch - Pass You By	3:57

20 songs 1 hour and 53 minutes

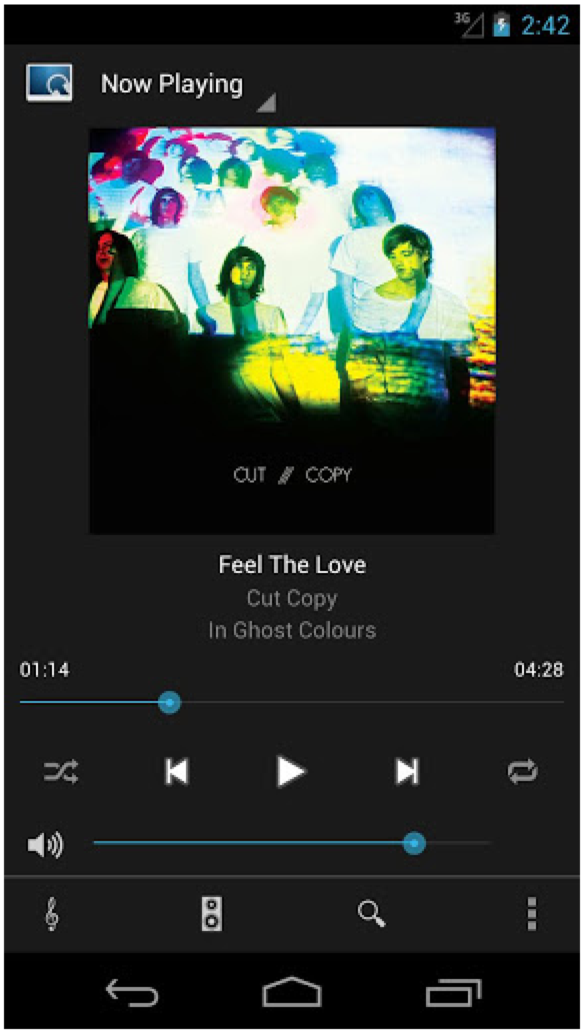
When you search in Sonata, it only sends the first to letters of the search query to Mopidy, and then does the rest of the filtering itself on the client side. Since Spotify has a collection of millions of tracks and they only return the first 100 hits for any search query, searching for two-letter combinations seldom returns any useful results. See #1 and the closed Sonata bug for details.

18.2.3 Theremin

Theremin is a graphical MPD client for OS X. It is unmaintained, but generally works well with Mopidy.

18.3 MPD Android clients

18.3.1 MPDroid

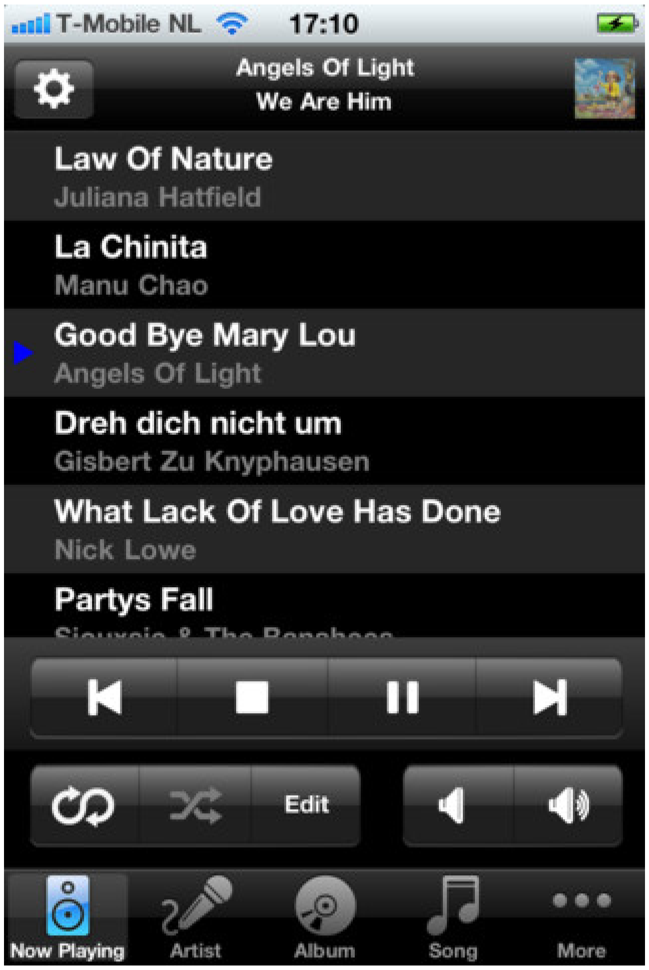


You can get MPDroid from Google Play.

MPDroid is a good MPD client, and really the only one we can recommend.

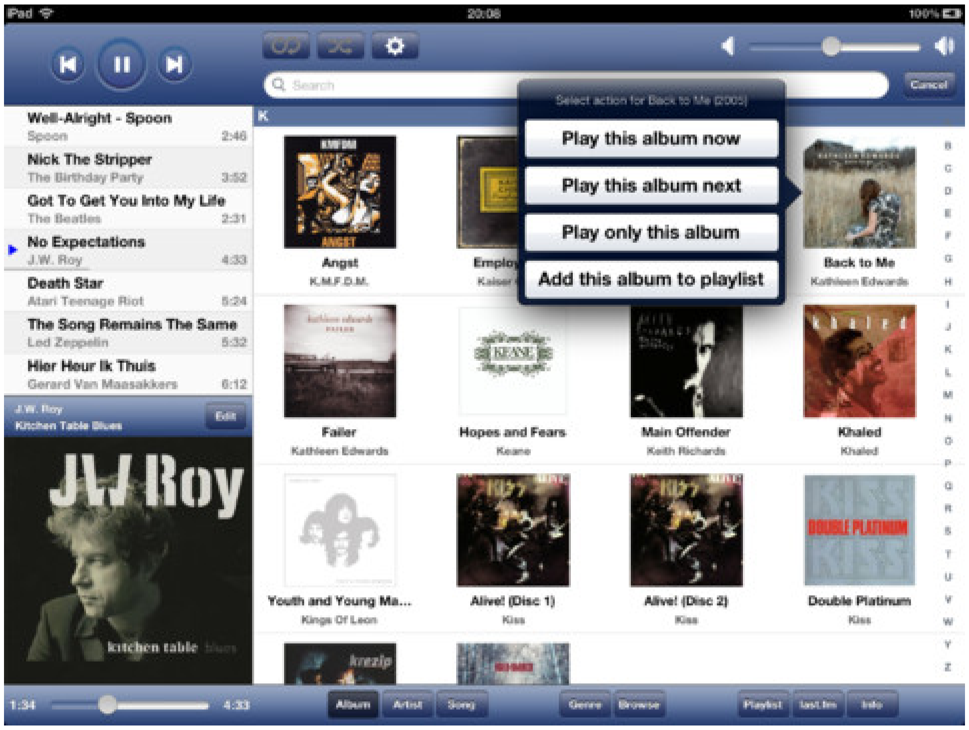
18.4 MPD iOS clients

18.4.1 MPoD



The MPoD iPhone/iPod Touch app can be installed from MPoD at iTunes Store.

18.4.2 MPaD

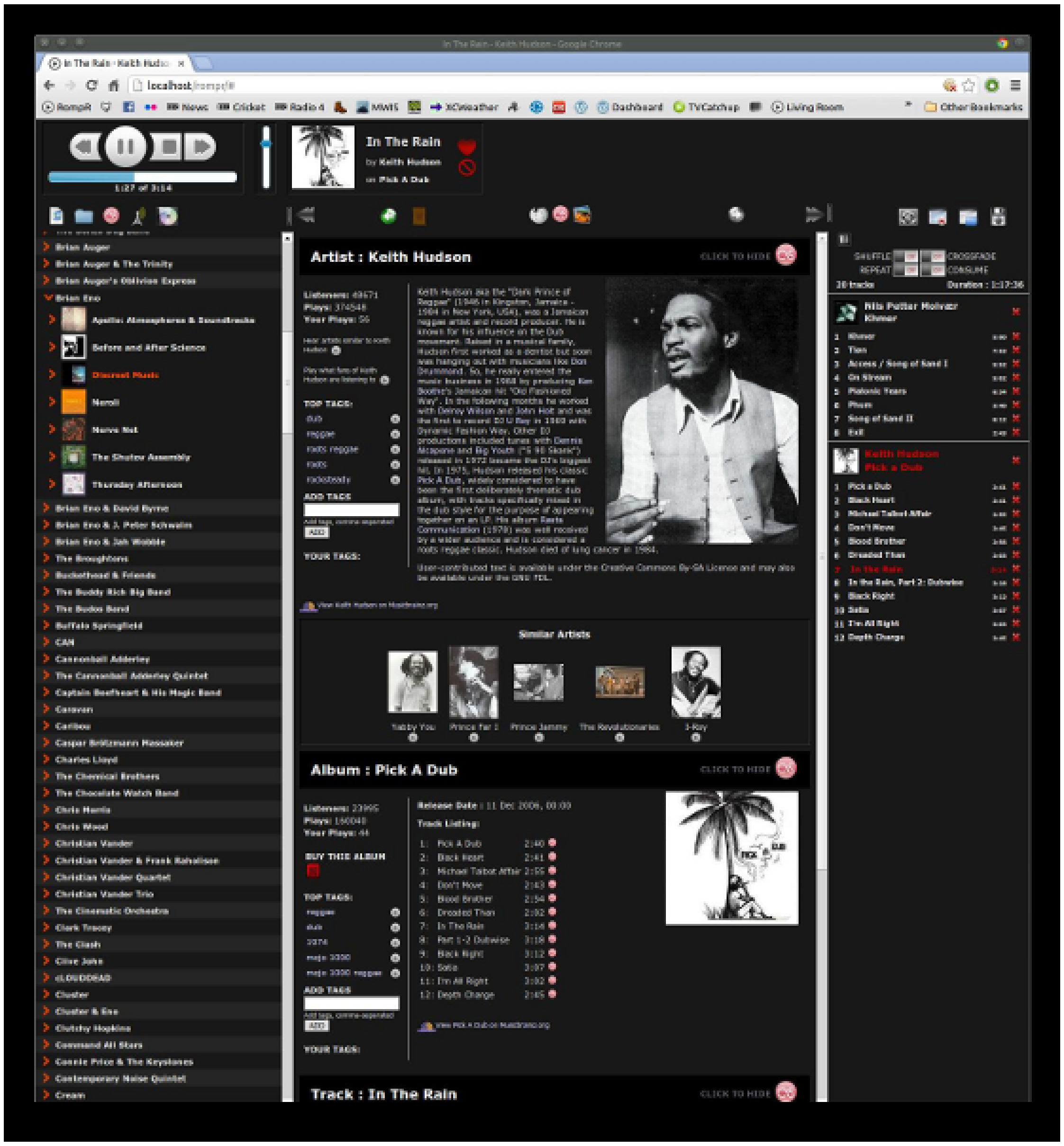


The MPaD iPad app can be purchased from MPaD at iTunes Store

18.5 MPD web clients

The following web clients use the MPD protocol to communicate with Mopidy. For other web clients, see *HTTP clients*.

18.5.1 Rompr



Rompr is a web based MPD client. mrvanes, a Mopidy and Rompr user, said: "These projects are a real match made in heaven."

18.5.2 Partify

Partify is a web based MPD client focussing on making music playing collaborative and social.

CHAPTER 19

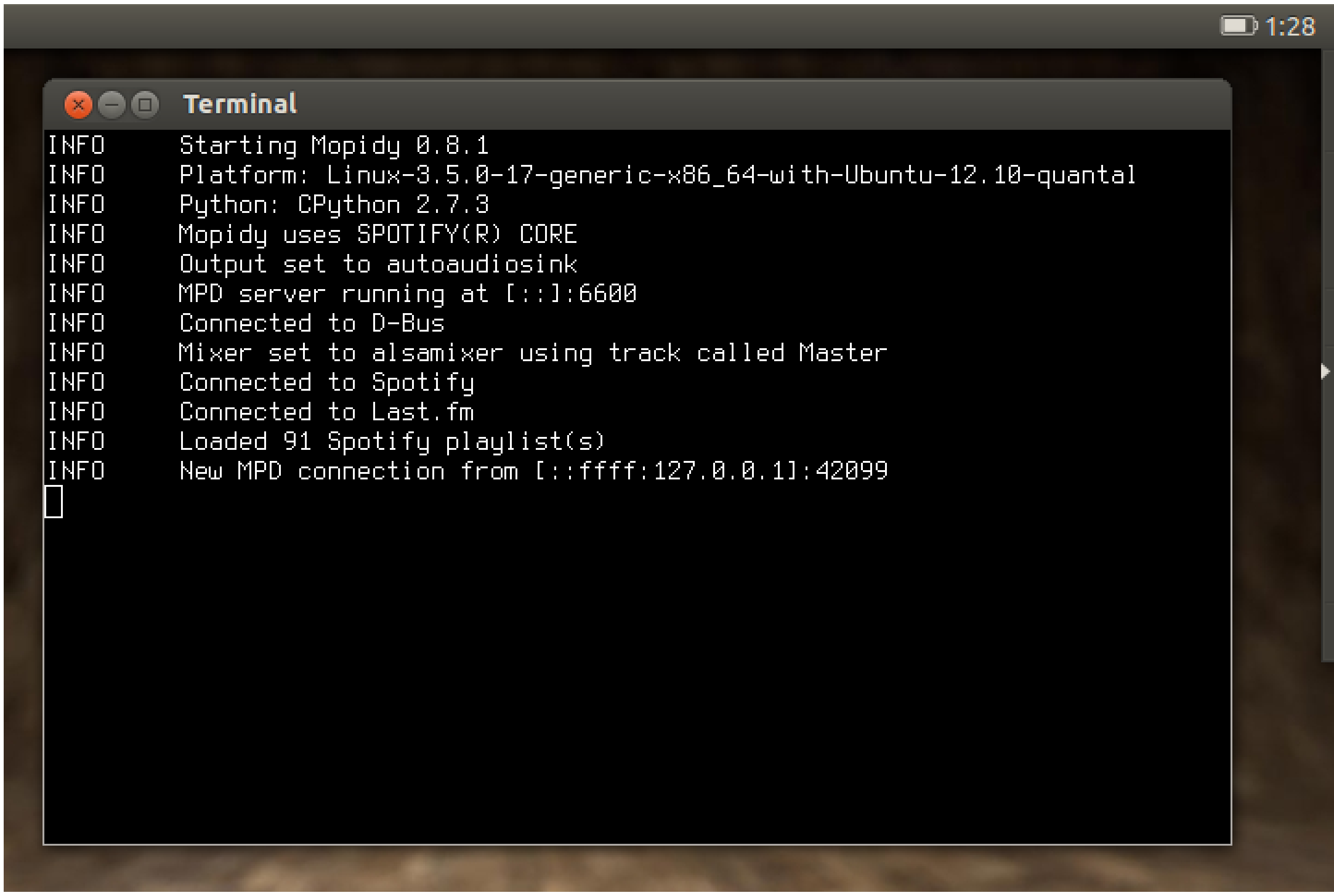
MPRIS clients

MPRIS is short for Media Player Remote Interfacing Specification. It's a spec that describes a standard D-Bus interface for making media players available to other applications on the same system.

The MPRIS frontend provided by the Mopidy-MPRIS extension currently implements all required parts of the MPRIS spec, plus the optional playlist interface. It does not implement the optional tracklist interface.

19.1 Ubuntu Sound Menu

The Ubuntu Sound Menu is the default sound menu in Ubuntu since 10.10 or 11.04. By default, it only includes the Rhytmbox music player, but many other players can integrate with the sound menu, including the official Spotify player and Mopidy.



```
INFO     Starting Mopidy 0.8.1
INFO     Platform: Linux-3.5.0-17-generic-x86_64-with-Ubuntu-12.10-quantal
INFO     Python: CPython 2.7.3
INFO     Mopidy uses SPOTIFY(R) CORE
INFO     Output set to autoaudiosink
INFO     MPD server running at [::]:6600
INFO     Connected to D-Bus
INFO     Mixer set to alsamixer using track called Master
INFO     Connected to Spotify
INFO     Connected to Last.fm
INFO     Loaded 91 Spotify playlist(s)
INFO     New MPD connection from [::ffff:127.0.0.1]:42099
```

If you install Mopidy from apt.mopidy.com, the sound menu should work out of the box. If you install Mopidy in any other way, you need to make sure that the file located at `extra/desktop/mopidy.desktop` in the Mopidy git repo is installed as `/usr/share/applications/mopidy.desktop`, and that the properties `TryExec` and `Exec` in the file points to an existing executable file, preferably your Mopidy executable. If this isn't in place, the sound menu will not detect that Mopidy is running.

Next, Mopidy's MPRIS frontend must be running for the sound menu to be able to control Mopidy. The frontend is enabled by default, so as long as you have all its dependencies available, you should be good to go. Keep an eye out for warnings or errors from the MPRIS frontend when you start Mopidy, since it may fail because of missing dependencies or because Mopidy is started outside of X; the frontend won't work if `$DISPLAY` isn't set when Mopidy is started.

Under normal use, if Mopidy isn't running and you open the menu and click on "Mopidy Music Server", a terminal window will open and automatically start Mopidy. If Mopidy is already running, you'll see that Mopidy is marked with an arrow to the left of its name, like in the screen shot above, and the player controls will be visible. Mopidy doesn't support the MPRIS spec's optional playlist interface yet, so you'll not be able to select what track to play from the sound menu. If you use an MPD client to queue a playlist, you can use the sound menu to check what you're currently playing, pause, resume, and skip to the next and previous track.

In summary, Mopidy's sound menu integration is currently not a full featured client, but it's a convenient addition to an MPD client since it's always easily available on Unity's menu bar.

19.2 Rygel

Rygel is an application that will translate between Mopidy's MPRIS interface and UPnP, and thus make Mopidy controllable from devices compatible with UPnP and/or DLNA. To read more about this, see *UPnP clients*.

CHAPTER 20

UPnP clients

UPnP is a set of specifications for media sharing, playing, remote control, etc, across a home network. The specs are supported by a lot of consumer devices (like smartphones, TVs, Xbox, and PlayStation) that are often labeled as being DLNA compatible or certified.

The DLNA guidelines and UPnP specifications defines several device roles, of which Mopidy may play two:

DLNA Digital Media Server (DMS) / UPnP AV MediaServer:

> A MediaServer provides a library of media and is capable of streaming that media to a MediaRenderer. If Mopidy was a MediaServer, you could browse and play Mopidy's music on a TV, smartphone, or tablet supporting UPnP. Mopidy does not currently support this, but we may in the future. #52 is the relevant wishlist issue.

DLNA Digital Media Renderer (DMR) / UPnP AV MediaRenderer:

> A MediaRenderer is asked by some remote controller to play some given media, typically served by a MediaServer. If Mopidy was a MediaRenderer, you could use e.g. your smartphone or tablet to make Mopidy play media. Mopidy *does already* have experimental support for being a MediaRenderer, as you can read more about below.

20.1 Mopidy as an UPnP MediaRenderer

There are two ways Mopidy can be made available as an UPnP MediaRenderer: Using Mopidy-MPRIS and Rygel, or using Mopidy-MPD and upmpdcli.

20.1.1 upmpdcli

upmpdcli is recommended, since it is easier to setup, and offers OpenHome compatibility. upmpdcli exposes a UPnP MediaRenderer to the network, while using the MPD protocol to control Mopidy.

1. Install upmpdcli. On Debian/Ubuntu:

```
apt-get install upmpdcli
```

Alternatively, follow the instructions from the upmpdcli website.

2. The default settings of upmpdcli will work with the default settings of *Mopidy-MPD*. Edit `/etc/upmpdcli.conf` if you want to use different ports, hosts, or other settings.

3. Start upmpdcli using the command:

```
upmpdcli
```

Or, run it in the background as a service:

```
sudo service upmpdcli start
```

4. A UPnP renderer should be available now.

20.1.2 Rygel

With the help of the Rygel project Mopidy can be made available as an UPnP MediaRenderer. Rygel will interface with the MPRIS interface provided by the Mopidy-MPRIS extension, and make Mopidy available as a MediaRenderer on the local network. Since this depends on the MPRIS frontend, which again depends on D-Bus being available, this will only work on Linux, and not OS X. MPRIS/D-Bus is only available to other applications on the same host, so Rygel must be running on the same machine as Mopidy.

1. Start Mopidy and make sure the MPRIS frontend is working. It is activated by default when the Mopidy-MPRIS extension is installed, but you may miss dependencies or be using OS X, in which case it will not work. Check the console output when Mopidy is started for any errors related to the MPRIS frontend. If you're unsure it is working, there are instructions for how to test it in the Mopidy-MPRIS readme.

2. Install Rygel. On Debian/Ubuntu:

```
sudo apt-get install rygel
```

3. Enable Rygel's MPRIS plugin. On Debian/Ubuntu, edit `/etc/rygel.conf`, find the `[MPRIS]` section, and change `enabled=false` to `enabled=true`.

4. Start Rygel by running:

```
rygel
```

Example output:

```
$ rygel
Rygel-Message: New plugin 'MediaExport' available
Rygel-Message: New plugin 'org.mpris.MediaPlayer2.mopidy' available
```

In the above example, you can see that Rygel found Mopidy, and it is now making Mopidy available through Rygel.

20.2 The UPnP-Inspector client

UPnP-Inspector is a graphical analyzer and debugging tool for UPnP services. It will detect any UPnP devices on your network, and show these in a tree structure. This is not a tool for your everyday music listening while relaxing on the couch, but it may be of use for testing that your setup works correctly.

1. Install UPnP-Inspector. On Debian/Ubuntu:

   ```
   sudo apt-get install upnp-inspector
   ```

2. Run it:

   ```
   upnp-inspector
   ```

3. Assuming that Mopidy is running with a working MPRIS frontend, and that Rygel is running on the same machine, Mopidy should now appear in UPnP-Inspector's device list.
4. If you expand the tree item saying `Mopidy (MediaRenderer:2)` or similiar, and then the sub element named `AVTransport:2` or similar, you'll find a list of commands you can invoke. E.g. if you double-click the `Pause` command, you'll get a new window where you can press an `Invoke` button, and then Mopidy should be paused.

Note that if you have a firewall on the host running Mopidy and Rygel, and you want this to be exposed to the rest of your local network, you need to open up your firewall for UPnP traffic. UPnP use UDP port 1900 as well as some dynamically assigned ports. I've only verified that this procedure works across the network by temporarily disabling the firewall on the the two hosts involved, so I'll leave any firewall configuration as an exercise to the reader.

20.3 Other clients

For a long list of UPnP clients for all possible platforms, see Wikipedia's List of UPnP AV media servers and clients.

CHAPTER 21

Authors

Mopidy is copyright 2009-2016 Stein Magnus Jodal and contributors. Mopidy is licensed under the Apache License, Version 2.0.

The following persons have contributed to Mopidy. The list is in the order of first contribution. For details on who have contributed what, please refer to our Git repository.

- Stein Magnus Jodal <stein.magnus@jodal.no>
- Johannes Knutsen <johannes@knutseninfo.no>
- Thomas Adamcik <thomas@adamcik.no>
- Kristian Klette <klette@samfundet.no>
- Martins Grunskis <martins@grunskis.com>
- Henrik Olsson <henrik@fixme.se>
- Antoine Pierlot-Garcin <antoine@bokbox.com>
- John Bäckstrand <sopues@gmail.com>
- Fred Hatfull <fred.hatfull@gmail.com>
- Erling Børresen <erling@fenicore.net>
- David Caruso <deibido.caruso@gmail.com>
- Christian Johansen <christian@cjohansen.no>
- Matt Bray <mattjbray@gmail.com>
- Trygve Aaberge <trygveaa@gmail.com>
- Wouter van Wijk <woutervanwijk@gmail.com>
- Jeremy B. Merrill <jeremybmerrill@gmail.com>
- Adam Rigg <adam@adamrigg.id.au>
- Ernst Bammer <herr.ernst@gmail.com>

- Nick Steel <kingosticks@gmail.com>
- Zan Dobersek <zandobersek@gmail.com>
- Thomas Refis <refis.thomas@gmail.com>
- Janez Troha <janez.troha@gmail.com>
- Tobias Sauerwein <cgtobi@gmail.com>
- Alli Witheford <alzeih@gmail.com>
- Alexandre Petitjean <alpetitjean@gmail.com>
- Terje Larsen <terlar@gmail.com>
- Javier Domingo Cansino <javierdo1@gmail.com>
- Pavol Babincak <scroolik@gmail.com>
- Javier Domingo <javierdo1@gmail.com>
- Lasse Bigum <lasse@bigum.org>
- David Eisner <david.eisner@oriel.oxon.org>
- Pål Ruud <ruudud@gmail.com>
- Thomas Kemmer <tkemmer@computer.org>
- Paul Connolley <paul.connolley@gmail.com>
- Luke Giuliani <luke@giuliani.com.au>
- Colin Montgomerie <kiteflyingmonkey@gmail.com>
- Simon de Bakker <simon@simbits.nl>
- Arnaud Barisain-Monrose <abarisain@gmail.com>
- Nathan Harper <nathan.sam.harper@gmail.com>
- Pierpaolo Frasa <pfrasa@smail.uni-koeln.de>
- Thomas Scholtes <thomas-scholtes@gmx.de>
- Sam Willcocks <sam@wlcx.cc>
- Ignasi Fosch <natx@y10k.ws>
- Arjun Naik <arjun@arjunnaik.in>
- Christopher Schirner <christopher@hackerspace-bamberg.de>
- Dmitry Sandalov <dmitry@sandalov.org>
- Lukas Vogel <lukas@vogelnest.org>
- Thomas Amland <thomas.amland@gmail.com>
- Deni Bertovic <deni@kset.org>
- Ali Ukani <ali.ukani@gmail.com>
- Dirk Groenen <dirk_groenen@live.nl>
- John Cass <john.cass77@gmail.com>
- Laura Barber <laura.c.barber@gmail.com>
- Jakab Kristóf <jaksi07c8@gmail.com>

- Ronald Zielaznicki <zielaznickizm@g.cofc.edu>
- Wojciech Wnętrzak <w.wnetrzak@gmail.com>
- Camilo Nova <camilo.nova@gmail.com>
- Dražen Lučanin <kermit666@gmail.com>
- Naglis Jonaitis <njonaitis@gmail.com>
- Kyle Heyne <kyleheyne@gmail.com>
- Tom Roth <rawdlite@googlemail.com>
- Mark Greenwood <fatgerman@gmail.com>
- Stein Karlsen <karlsen.stein@gmail.com>
- Dejan Prokić <dejanp@nordeus.eu>
- Eric Jahn <ejahn@newstore.com>
- Mikhail Golubev <qsolo825@gmail.com>
- Danilo Bargen <mail@dbrgn.ch>
- Bjørnar Snoksrud <bjornar@snoksrud.no>
- Giorgos Logiotatidis <seadog@sealabs.net>
- Ben Evans <ben@bensbit.co.uk>
- vrs01 <vrs01@users.noreply.github.com>
- Loïck Bonniot <git@lesterpig.com>
- Cadel Watson <cadel@cadelwatson.com>
- Daniel Hahler <git@thequod.de>

If want to help us making Mopidy better, the best way to do so is to contribute back to the community, either through code, documentation, tests, bug reports, or by helping other users, spreading the word, etc. See *Contributing* for a head start.

CHAPTER 22

Sponsors

The Mopidy project would like to thank the following sponsors for supporting the project.

22.1 Rackspace

Rackspace lets Mopidy use their hosting services for free. We use their services for the following sites:

- Hosting of the APT package repository at https://apt.mopidy.com.
- Hosting of the Discourse forum at https://discuss.mopidy.com.
- Mailgun for sending emails from the Discourse forum.

22.2 Fastly

Fastly lets Mopidy use their CDN for free. We accelerate requests to all Mopidy services, including:

- https://apt.mopidy.com/dists/, which is used to distribute Debian packages.
- https://dl.mopidy.com/pimusicbox/, which is used to distribute Pi Musicbox images.

22.3 GlobalSign

GlobalSign provides Mopidy with a free SSL certificate for mopidy.com, which we use to secure access to all our web sites.

CHAPTER 23

Changelog

This changelog is used to track all major changes to Mopidy.

23.1 v1.1.2 (2016-01-18)

Bug fix release.

- Main: Catch errors when loading the *logging/config_file* file. (Fixes: #1320)
- Core: If changing to another track while the player is paused, the new track would not be added to the history or marked as currently playing. (Fixes: #1352, PR: #1356)
- Core: Skips over unplayable tracks if the user attempts to change tracks while paused, like we already did if in playing state. (Fixes #1378, PR: #1379)
- Core: Make *lookup()* ignore tracks with empty URIs. (Partly fixes: #1340, PR: #1381)
- Core: Fix crash if backends emits events with wrong names or arguments. (Fixes: #1383)
- Stream: If an URI is considered playable, don't consider it as a candidate for playlist parsing. Just looking at MIME type prefixes isn't enough, as for example Ogg Vorbis has the MIME type `application/ogg`. (Fixes: #1299)
- Local: If the scan or clear commands are used on a library that does not exist, exit with an error. (Fixes: #1298)
- MPD: Notify idling clients when a seek is performed. (Fixes: #1331)
- MPD: Don't return tracks with empty URIs. (Partly fixes: #1340, PR: #1343)
- MPD: Add `volume` command that was reintroduced, though still as a deprecated command, in MPD 0.18 and is in use by some clients like mpc. (Fixes: #1393, PR: #1397)
- Proxy: Handle case where *proxy/port* is either missing from config or set to an empty string. (PR: #1371)

23.2 v1.1.1 (2015-09-14)

Bug fix release.

- Dependencies: Specify that we need Requests >= 2.0, not just any version. This ensures that we fail earlier if Mopidy is used with a too old Requests.
- Core: Make `mopidy.core.LibraryController.refresh()` work for all backends with a library provider. Previously, it wrongly worked for all backends with a playlists provider. (Fixes: #1257)
- Core: Respect `core/cache_dir` and `core/data_dir` config values added in 1.1.0 when creating the dirs Mopidy need to store data. This should not change the behavior for desktop users running Mopidy. When running Mopidy as a system service installed from a package which sets the core dir configs properly (e.g. Debian and Arch packages), this fix avoids the creation of a couple of directories that should not be used, typically `/var/lib/mopidy/.local` and `/var/lib/mopidy/.cache`. (Fixes: #1259, PR: #1266)
- Core: Fix error in `get_eot_tlid()` docstring. (Fixes: #1269)
- Audio: Add `timeout` parameter to `scan()`. (Part of: #1250, PR: #1281)
- Extension support: Make `get_cache_dir()`, `get_config_dir()`, and `get_data_dir()` class methods, so they can be used without creating an instance of the `Extension` class. (Fixes: #1275)
- Local: Deprecate `local/data_dir` and respect `core/data_dir` instead. This does not change the defaults for desktop users, only system services installed from packages that properly set `core/data_dir`, like the Debian and Arch packages. (Fixes: #1259, PR: #1266)
- Local: Change default value of `local/scan_flush_threshold` from 1000 to 100 to shorten the time Mopidy-Local-SQLite blocks incoming requests while scanning the local library.
- M3U: Changed default for the `m3u/playlists_dir` from `$XDG_DATA_DIR/mopidy/m3u` to unset, which now means the extension's data dir. This does not change the defaults for desktop users, only system services installed from packages that properly set `core/data_dir`, like the Debian and Arch pakages. (Fixes: #1259, PR: #1266)
- Stream: Expand nested playlists to find the stream URI. This used to work, but regressed in 1.1.0 with the extraction of stream playlist parsing from GStreamer to being handled by the Mopidy-Stream backend. (Fixes: #1250, PR: #1281)
- Stream: If "file" is present in the `stream/protocols` config value and the *Mopidy-File* extension is enabled, we exited with an error because two extensions claimed the same URI scheme. We now log a warning recommending to remove "file" from the `stream/protocols` config, and then proceed startup. (Fixes: #1248, PR: #1254)
- Stream: Fix bug in new playlist parser. A non-ASCII char in an urilist comment would cause a crash while parsing due to comparision of a non-ASCII bytestring with a Unicode string. (Fixes: #1265)
- File: Adjust log levels when failing to expand `$XDG_MUSIC_DIR` into a real path. This usually happens when running Mopidy as a system service, and thus with a limited set of environment variables. (Fixes: #1249, PR: #1255)
- File: When browsing files, we no longer scan the files to check if they're playable. This makes browsing of the file hierarchy instant for HTTP clients, which do no scanning of the files' metadata, and a bit faster for MPD clients, which no longer scan the files twice. (Fixes: #1260, PR: #1261)
- File: Allow looking up metadata about any `file://` URI, just like we did in Mopidy 1.0.x, where Mopidy-Stream handled `file://` URIs. In Mopidy 1.1.0, Mopidy-File did not allow one to lookup files outside the directories listed in `file/media_dir`. This broke Mopidy-Local-SQLite when the `local/media_dir` directory was not within one of the `file/media_dirs` directories. For browsing of files, we still limit access to files inside the `file/media_dir` directories. For lookup, you can now read metadata for any file you know the path of. (Fixes: #1268, PR: #1273)

- Audio: Fix timeout handling in scanner. This regression caused timeouts to expire before it should, causing scans to fail.
- Audio: Update scanner to emit MIME type instead of an error when missing a plugin.

23.3 v1.1.0 (2015-08-09)

Mopidy 1.1 is here!

Since the release of 1.0, we've closed or merged approximately 65 issues and pull requests through about 400 commits by a record high 20 extraordinary people, including 14 newcomers. That's less issues and commits than in the 1.0 release, but even more contributors, and a doubling of the number of newcomers. Thanks to *everyone* who has *contributed*, especially those that joined the sprint at EuroPython 2015 in Bilbao, Spain a couple of weeks ago!

As we promised with the release of Mopidy 1.0, any extension working with Mopidy 1.0 should continue working with all Mopidy 1.x releases. However, this release brings a lot stronger enforcement of our documented APIs. If an extension doesn't use the APIs properly, it may no longer work. The advantage of this change is that Mopidy is now more robust against errors in extensions, and also provides vastly better error messages when extensions misbehave. This should make it easier to create quality extensions.

The major features of Mopidy 1.1 are:

- Validation of the arguments to all core API methods, as well as all responses from backends and all data model attributes.
- New bundled backend, Mopidy-File. It is similar to Mopidy-Local, but allows you to browse and play music from local disk without running a scan to index the music first. The drawback is that it doesn't support searching.
- The Mopidy-MPD server should now be up to date with the 0.19 version of the MPD protocol.

23.3.1 Dependencies

- Mopidy now requires Requests.
- Heads up: Porting from GStreamer 0.10 to 1.x and support for running Mopidy with Python 3.4+ is not far off on our roadmap.

23.3.2 Core API

- **Deprecated:** Calling the following methods with `kwargs` is being deprecated. (PR: #1090)
  - *`mopidy.core.LibraryController.search()`*
  - *`mopidy.core.PlaylistsController.filter()`*
  - *`mopidy.core.TracklistController.filter()`*
  - *`mopidy.core.TracklistController.remove()`*
- Updated core controllers to handle backend exceptions in all calls that rely on multiple backends. (Issue: #667)
- Update core methods to do strict input checking. (Fixes: #700)
- Add `tlid` alternatives to methods that take `tl_track` and also add `get_{eot,next,previous}_tlid` methods as light weight alternatives to the `tl_track` versions of the calls. (Fixes: #1131, PR: #1136, #1140)
- Add `mopidy.core.PlaybackController.get_current_tlid()`. (Part of: #1137)
- Update core to handle backend crashes and bad data. (Fixes: #1161)

- Add `core/max_tracklist_length` config and limitation. (Fixes: #997 PR: #1225)
- Added `playlist_deleted` event. (Fixes: #996)

23.3.3 Models

- Added type checks and other sanity checks to model construction and serialization. (Fixes: #865)
- Memory usage for models has been greatly improved. We now have a lower overhead per instance by using slots, interned identifiers and automatically reuse instances. For the test data set this was developed against, a library of ~14.000 tracks, went from needing ~75MB to ~17MB. (Fixes: #348)
- Added `mopidy.models.Artist.sortname` field that is mapped to `musicbrainz-sortname` tag. (Fixes: #940)

23.3.4 Configuration

- Add new configurations to set base directories to be used by Mopidy and Mopidy extensions: `core/cache_dir`, `core/config_dir`, and `core/data_dir`. (Fixes: #843, PR: #1232)

23.3.5 Extension support

- Add new methods to `Extension` class for getting cache, config and data directories specific to your extension:
  - `mopidy.ext.Extension.get_cache_dir()`
  - `mopidy.ext.Extension.get_config_dir()`
  - `mopidy.ext.Extension.get_data_dir()`

  Extensions should use these methods so that the correct directories are used both when Mopidy is run by a regular user and when run as a system service. (Fixes: #843, PR: #1232)
- Add `mopidy.httpclient.format_proxy()` and `mopidy.httpclient.format_user_agent()`. (Part of: #1156)
- It is now possible to import `mopidy.backends` without having GObject or GStreamer installed. In other words, a lot of backend extensions should now be able to run tests in a virtualenv with global site-packages disabled. This removes a lot of potential error sources. (Fixes: #1068, PR: #1115)

23.3.6 Local backend

- Filter out `None` from `get_distinct()` results. All returned results should be strings. (Fixes: #1202)

23.3.7 Stream backend

- Move stream playlist parsing from GStreamer to the stream backend. (Fixes: #671)

23.3.8 File backend

The *Mopidy-File* backend is a new bundled backend. It is similar to Mopidy-Local since it works with local files, but it differs in a few key ways:

- Mopidy-File lets you browse your media files by their file hierarchy.

- It supports multiple media directories, all exposed under the "Files" directory when you browse your library with e.g. an MPD client.
- There is no index of the media files, like the JSON or SQLite files used by Mopidy-Local. Thus no need to scan the music collection before starting Mopidy. Everything is read from the file system when needed and changes to the file system is thus immediately visible in Mopidy clients.
- Because there is no index, there is no support for search.

Our long term plan is to keep this very simple file backend in Mopidy, as it has a well defined and limited scope, while splitting the more feature rich Mopidy-Local extension out to an independent project. (Fixes: #1004, PR: #1207)

23.3.9 M3U backend

- Support loading UTF-8 encoded M3U files with the `.m3u8` file extension. (PR: #1193)

23.3.10 MPD frontend

- The MPD command `count` now ignores tracks with no length, which would previously cause a `TypeError`. (PR: #1192)
- Concatenate multiple artists, composers and performers using the "A;B" format instead of "A, B". This is a part of updating our protocol implementation to match MPD 0.19. (PR: #1213)
- Add "not implemented" skeletons of new commands in the MPD protocol version 0.19:
  - Current playlist:
    - `rangeid`
    - `addtagid`
    - `cleartagid`
  - Mounts and neighbors:
    - `mount`
    - `unmount`
    - `listmounts`
    - `listneighbors`
  - Music DB:
    - `listfiles`
- Track data now include the `Last-Modified` field if set on the track model. (Fixes: #1218, PR: #1219)
- Implement `tagtypes` MPD command. (PR: #1235)
- Exclude empty tags fields from metadata output. (Fixes: #1045, PR: #1235)
- Implement protocol extensions to output Album URIs and Album Images when outputting track data to clients. (PR: #1230)
- The MPD commands `lsinfo` and `listplaylists` are now implemented using the *`as_list()`* method, which retrieves a lot less data and is thus much faster than the deprecated *`get_playlists()`*. The drawback is that the `Last-Modified` timestamp is not available through this method, and the timestamps in the MPD command responses are now always set to the current time.

23.3.11 Internal changes

- Tests have been cleaned up to stop using deprecated APIs where feasible. (Partial fix: #1083, PR: #1090)

23.4 v1.0.8 (2015-07-22)

Bug fix release.

- Fix reversal of `Title` and `Name` in MPD protocol (Fixes: #1212 PR: #1214)
- Fix crash if an M3U file in the `m3u/playlist_dir` directory has a file name not decodable with the current file system encoding. (Fixes: #1209)

23.5 v1.0.7 (2015-06-26)

Bug fix release.

- Fix error in the MPD command `list title ...`. The error was introduced in v1.0.6.

23.6 v1.0.6 (2015-06-25)

Bug fix release.

- Core/MPD/Local: Add support for `title` in *`mopidy.core.LibraryController.get_distinct()`*. (Fixes: #1181, PR: #1183)
- Core: Make sure track changes make it to audio while paused. (Fixes: #1177, PR: #1185)

23.7 v1.0.5 (2015-05-19)

Bug fix release.

- Core: Add workaround for playlist providers that do not support creating playlists. (Fixes: #1162, PR #1165)
- M3U: Fix encoding error when saving playlists with non-ASCII track titles. (Fixes: #1175, PR #1176)

23.8 v1.0.4 (2015-04-30)

Bug fix release.

- Audio: Since all previous attempts at tweaking the queuing for #1097 seems to break things in subtle ways for different users. We are giving up on tweaking the defaults and just going to live with a bit more lag on software volume changes. (Fixes: #1147)

23.9 v1.0.3 (2015-04-28)

Bug fix release.

- HTTP: Another follow-up to the Tornado <3.0 fixing. Since the tests aren't run for Tornado 2.3 we didn't catch that our previous fix wasn't sufficient. (Fixes: #1153, PR: #1154)
- Audio: Follow-up fix for #1097 still exhibits issues for certain setups. We are giving this get an other go by setting the buffer size to maximum 100ms instead of a fixed number of buffers. (Addresses: #1147, PR: #1154)

23.10 v1.0.2 (2015-04-27)

Bug fix release.

- HTTP: Make event broadcasts work with Tornado 2.3 again. The threading fix in v1.0.1 broke this.
- Audio: Fix for #1097 tuned down the buffer size in the queue. Turns out this can cause distortions in certain cases. Give this an other go with a more generous buffer size. (Addresses: #1147, PR: #1152)
- Audio: Make sure mute events get emitted by software mixer. (Fixes: #1146, PR: #1152)

23.11 v1.0.1 (2015-04-23)

Bug fix release.

- Core: Make the new history controller available for use. (Fixes: mopidy.js#6)
- Audio: Software volume control has been reworked to greatly reduce the delay between changing the volume and the change taking effect. (Fixes: #1097, PR: #1101)
- Audio: As a side effect of the previous bug fix, software volume is no longer tied to the PulseAudio application volume when using `pulsesink`. This behavior was confusing for many users and doesn't work well with the plans for multiple outputs.
- Audio: Update scanner to decode all media it finds. This should fix cases where the scanner hangs on non-audio files like video. The scanner will now also let us know if we found any decodeable audio. (Fixes: #726, PR: issue:*1124*)
- HTTP: Fix threading bug that would cause duplicate delivery of WS messages. (PR: #1127)
- MPD: Fix case where a playlist that is present in both browse and as a listed playlist breaks the MPD frontend protocol output. (Fixes #1120, PR: #1142)

23.12 v1.0.0 (2015-03-25)

Three months after our fifth anniversary, Mopidy 1.0 is finally here!

Since the release of 0.19, we've closed or merged approximately 140 issues and pull requests through more than 600 commits by a record high 19 extraordinary people, including seven newcomers. Thanks to *everyone* who has *contributed*!

For the longest time, the focus of Mopidy 1.0 was to be another incremental improvement, to be numbered 0.20. The result is still very much an incremental improvement, with lots of small and larger improvements across Mopidy's functionality.

The major features of Mopidy 1.0 are:

- *Semantic Versioning*. We promise to not break APIs before Mopidy 2.0. A Mopidy extension working with Mopidy 1.0 should continue to work with all Mopidy 1.x releases.
- Preparation work to ease migration to a cleaned up and leaner core API in Mopidy 2.0, and to give us some of the benefits of the cleaned up core API right away.
- Preparation work to enable gapless playback in an upcoming 1.x release.

23.12.1 Dependencies

Since the previous release there are no changes to Mopidy's dependencies. However, porting from GStreamer 0.10 to 1.x and support for running Mopidy with Python 3.4+ is not far off on our roadmap.

23.12.2 Core API

In the API used by all frontends and web extensions there is lots of methods and arguments that are now deprecated in preparation for the next major release. With the exception of some internals that leaked out in the playback controller, no core APIs have been removed in this release. In other words, most clients should continue to work unchanged when upgrading to Mopidy 1.0. Though, it is strongly encouraged to review any use of the deprecated parts of the API as those parts will be removed in Mopidy 2.0.

- **Deprecated:** Deprecate all Python properties in the core API. The previously undocumented getter and setter methods are now the official API. This aligns the Python API with the WebSocket/JavaScript API. Python frontends needs to be updated. WebSocket/JavaScript API users are not affected. (Fixes: #952)
- Add `mopidy.core.HistoryController` which keeps track of what tracks have been played. (Fixes: #423, #1056, PR: #803, #1063)
- Add `mopidy.core.MixerController` which keeps track of volume and mute. (Fixes: #962)

Core library controller

- **Deprecated:** *`mopidy.core.LibraryController.find_exact()`*. Use *`mopidy.core.LibraryController.search()`* with the `exact` keyword argument set to `True`.
- **Deprecated:** The `uri` argument to *`mopidy.core.LibraryController.lookup()`*. Use new `uris` keyword argument instead.
- Add `exact` keyword argument to *`mopidy.core.LibraryController.search()`*.
- Add `uris` keyword argument to *`mopidy.core.LibraryController.lookup()`* which allows for simpler lookup of multiple URIs. (Fixes: #1008, PR: #1047)
- Updated *`mopidy.core.LibraryController.search()`* and *`mopidy.core.LibraryController.find_exact()`* to normalize and warn about malformed queries from clients. (Fixes: #1067, PR: #1073)
- Add *`mopidy.core.LibraryController.get_distinct()`* for getting unique values for a given field. (Fixes: #913, PR: #1022)
- Add *`mopidy.core.LibraryController.get_images()`* for looking up images for any URI that is known to the backends. (Fixes #973, PR: #981, #992 and #1013)

Core playlist controller

- **Deprecated:** `mopidy.core.PlaylistsController.get_playlists()`. Use `as_list()` and `get_items()` instead. (Fixes: #1057, PR: #1075)
- **Deprecated:** `mopidy.core.PlaylistsController.filter()`. Use `as_list()` and filter yourself.
- Add `mopidy.core.PlaylistsController.as_list()`. (Fixes: #1057, PR: #1075)
- Add `mopidy.core.PlaylistsController.get_items()`. (Fixes: #1057, PR: #1075)

Core tracklist controller

- **Removed:** The following methods were documented as internal. They are now fully private and unavailable outside the core actor. (Fixes: #1058, PR: #1062)
  - `mopidy.core.TracklistController.mark_played()`
  - `mopidy.core.TracklistController.mark_playing()`
  - `mopidy.core.TracklistController.mark_unplayable()`
- Add `uris` argument to `mopidy.core.TracklistController.add()` which allows for simpler addition of multiple URIs to the tracklist. (Fixes: #1060, PR: #1065)

Core playback controller

- **Removed:** Remove several internal parts that were leaking into the public API and was never intended to be used externally. (Fixes: #1070, PR: #1076)
  - `mopidy.core.PlaybackController.change_track()` is now internal.
  - Removed `on_error_step` keyword argument from `mopidy.core.PlaybackController.play()`
  - Removed `clear_current_track` keyword argument to `mopidy.core.PlaybackController.stop()`.
  - Made the following event triggers internal:
    * `mopidy.core.PlaybackController.on_end_of_track()`
    * `mopidy.core.PlaybackController.on_stream_changed()`
    * `mopidy.core.PlaybackController.on_tracklist_changed()`
  - `mopidy.core.PlaybackController.set_current_tl_track()` is now internal.
- **Deprecated:** The old methods on `mopidy.core.PlaybackController` for volume and mute management have been deprecated. Use `mopidy.core.MixerController` instead. (Fixes: #962)
- When seeking while paused, we no longer change to playing. (Fixes: #939, PR: #1018)
- Changed `mopidy.core.PlaybackController.play()` to take the return value from `mopidy.backend.PlaybackProvider.change_track()` into account when determining the success of the `play()` call. (PR: #1071)
- Add `mopidy.core.Listener.stream_title_changed()` and `mopidy.core.PlaybackController.get_stream_title()` for letting clients know about the current title in streams. (PR: #938, #1030)

23.12.3 Backend API

In the API implemented by all backends there have been way fewer but somewhat more drastic changes with some methods removed and new ones being required for certain functionality to continue working. Most backends were already updated to be compatible with Mopidy 1.0 before the release. New versions of the backends will be released shortly after Mopidy itself.

Backend library providers

- **Removed:** Remove `mopidy.backend.LibraryProvider.find_exact()`.
- Add an `exact` keyword argument to *`mopidy.backend.LibraryProvider.search()`* to replace the old `find_exact()` method.

Backend playlist providers

- **Removed:** Remove default implementation of `mopidy.backend.PlaylistsProvider.playlists`. This is potentially backwards incompatible. (PR: #1046)
- Changed the API for *`mopidy.backend.PlaylistsProvider`*. Note that this change is **not** backwards compatible. These changes are important to reduce the Mopidy startup time. (Fixes: #1057, PR: #1075)
  - Add *`mopidy.backend.PlaylistsProvider.as_list()`*.
  - Add *`mopidy.backend.PlaylistsProvider.get_items()`*.
  - Remove `mopidy.backend.PlaylistsProvider.playlists` property.

Backend playback providers

- Changed the API for *`mopidy.backend.PlaybackProvider`*. Note that this change is **not** backwards compatible for certain backends. These changes are crucial to adding gapless in one of the upcoming releases. (Fixes: #1052, PR: #1064)
  - *`mopidy.backend.PlaybackProvider.translate_uri()`* has been added. It is strongly recommended that all backends migrate to using this API for translating "Mopidy URIs" to real ones for playback.
  - The semantics and signature of *`mopidy.backend.PlaybackProvider.play()`* has changed. The method is now only used to set the playback state to playing, and no longer takes a track.

    Backends must migrate to *`mopidy.backend.PlaybackProvider.translate_uri()`* or *`mopidy.backend.PlaybackProvider.change_track()`* to continue working.
  - *`mopidy.backend.PlaybackProvider.prepare_change()`* has been added.

23.12.4 Models

- Add *`mopidy.models.Image`* model to be returned by *`mopidy.core.LibraryController.get_images()`*. (Part of #973)
- Change the semantics of *`mopidy.models.Track.last_modified`* to be milliseconds instead of seconds since Unix epoch, or a simple counter, depending on the source of the track. This makes it match the semantics of *`mopidy.models.Playlist.last_modified`*. (Fixes: #678, PR: #1036)

23.12.5 Commands

- Make the `mopidy` command print a friendly error message if the `gobject` Python module cannot be imported. (Fixes: #836)
- Add support for repeating the `-v` argument four times to set the log level for all loggers to the lowest possible value, including log records at levels lower than `DEBUG` too.
- Add path to the current `mopidy` executable to the output of `mopidy deps`. This make it easier to see that a user is using pip-installed Mopidy instead of APT-installed Mopidy without asking for `which mopidy` output.

23.12.6 Configuration

- Add support for the log level value `all` to the loglevels configurations. This can be used to show absolutely all log records, including those at custom levels below `DEBUG`.
- Add debug logging of unknown sections. (Fixes: #694, PR: #1002)

23.12.7 Logging

- Add custom log level `TRACE` (numerical level 5), which can be used by Mopidy and extensions to log at an even more detailed level than `DEBUG`.
- Add support for per logger color overrides. (Fixes: #808, PR: #1005)

23.12.8 Local backend

- Improve error logging for scanner. (Fixes: #856, PR: #874)
- Add symlink support with loop protection to file finder. (Fixes: #858, PR: #874)
- Add `--force` option for `mopidy local scan` for forcing a full rescan of the library. (Fixes: #910, PR: #1010)
- Stop ignoring `offset` and `limit` in searches when using the default JSON backed local library. (Fixes: #917, PR: #949)
- Removed double triggering of `playlists_loaded` event. (Fixes: #998, PR: #999)
- Cleanup and refactoring of local playlist code. Preserves playlist names better and fixes bug in deletion of playlists. (Fixes: #937, PR: #995 and rebased into #1000)
- Sort local playlists by name. (Fixes: #1026, PR: #1028)
- Moved playlist support out to a new extension, *Mopidy-M3U*.
- *Deprecated:* The config value `local/playlists_dir` is no longer in use and can be removed from your config.

Local library API

- Implementors of `mopidy.local.Library.lookup()` should now return a list of `Track` instead of a single track, just like the other `lookup()` methods in Mopidy. For now, returning a single track will continue to work. (PR: #840)
- Add support for giving local libraries direct access to tags and duration. (Fixes: #967)

- Add `mopidy.local.Library.get_images()` for looking up images for local URIs. (Fixes: #1031, PR: #1032 and #1037)

23.12.9 Stream backend

- Add support for HTTP proxies when doing initial metadata lookup for a stream. (Fixes #390, PR: #982)
- Add basic tests for the stream library provider.

23.12.10 M3U backend

- Mopidy-M3U is a new bundled backend. It provides the same M3U support as was previously part of the local backend. See *Migrating from Mopidy-Local playlists* for how to migrate your local playlists to work with the M3U backend. (Fixes: #1054, PR: #1066)
- In playlist names, replace "/", which are illegal in M3U file names, with "|". (PR: #1084)

23.12.11 MPD frontend

- Add support for blacklisting MPD commands. This is used to prevent clients from using `listall` and `listallinfo` which recursively lookup the entire "database". If you insist on using a client that needs these commands change `mpd/command_blacklist`.
- Start setting the `Name` field with the stream title when listening to radio streams. (Fixes: #944, PR: #1030)
- Enable browsing of artist references, in addition to albums and playlists. (PR: #884)
- Switch the `list` command over to using the new method `mopidy.core.LibraryController.get_distinct()` for increased performance. (Fixes: #913)
- In stored playlist names, replace "/", which are illegal, with "|" instead of a whitespace. Pipes are more similar to forward slash.
- Share a single mapping between names and URIs across all MPD sessions. (Fixes: #934, PR: #968)
- Add support for `toggleoutput` command. (PR: #1015)
- The `mixrampdb` and `mixrampdelay` commands are now known to Mopidy, but are not implemented. (PR: #1015)
- Fix crash on socket error when using a locale causing the exception's error message to contain characters not in ASCII. (Fixes: issue:*971*, PR: #1044)

23.12.12 HTTP frontend

- **Deprecated:** Deprecated the `http/static_dir` config. Please make your web clients pip-installable Mopidy extensions to make it easier to install for end users.
- Prevent a race condition in WebSocket event broadcasting from crashing the web server. (PR: #1020)

23.12.13 Mixers

- Add support for disabling volume control in Mopidy entirely by setting the configuration `audio/mixer` to `none`. (Fixes: #936, PR: #1015, #1035)

23.12.14 Audio

- **Removed:** Support for visualizers and the `audio/visualizer` config value. The feature was originally added as a workaround for all the people asking for ncmpcpp visualizer support, and since we could get it almost for free thanks to GStreamer. But, this feature did never make sense for a server such as Mopidy.
- **Deprecated:** Deprecated `mopidy.audio.Audio.emit_end_of_stream()`. Pass a `None` buffer to `mopidy.audio.Audio.emit_data()` to end the stream. This should only affect Mopidy-Spotify.
- Add `mopidy.audio.AudioListener.tags_changed()`. Notifies core when new tags are found.
- Add `mopidy.audio.Audio.get_current_tags()` for looking up the current tags of the playing media.
- Internal code cleanup within audio subsystem:
  - Started splitting audio code into smaller better defined pieces.
  - Improved GStreamer related debug logging.
  - Provide better error messages for missing plugins.
  - Add foundation for trying to re-add multiple output support.
  - Add internal helper for converting GStreamer data types to Python.
  - Reduce scope of audio scanner to just find tags and duration. Modification time, URI and minimum length handling are now outside of this class.
  - Update scanner to operate with milliseconds for duration.
  - Update scanner to use a custom source, typefind and decodebin. This allows us to detect playlists before we try to decode them.
  - Refactored scanner to create a new pipeline per track, this is needed as reseting decodebin is much slower than tearing it down and making a fresh one.
- Move and rename helper for converting tags to tracks.
- Ignore albums without a name when converting tags to tracks.
- Support UTF-8 in M3U playlists. (Fixes: #853)
- Add workaround for volume not persisting across tracks on OS X. (Issue: #886, PR: #958)
- Improved missing plugin error reporting in scanner. (PR: #1033)
- Introduced a new return type for the scanner, a named tuple with `uri`, `tags`, `duration`, `seekable` and `mime`. (PR: #1033)
- Added support for checking if the media is seekable, and getting the initial MIME type guess. (PR: #1033)

23.12.15 Mopidy.js client library

This version has been released to npm as Mopidy.js v0.5.0.

- Reexport When.js library as `Mopidy.when`, to make it easily available to users of Mopidy.js. (Fixes: mopidy.js#1)
- Default to `wss://` as the WebSocket protocol if the page is hosted on `https://`. This has no effect if the `webSocketUrl` setting is specified. (Pull request: mopidy.js#2)
- Upgrade dependencies.

23.12.16 Development

- Add new *contribution guidelines*.
- Add new *development guide*.
- Speed up event emitting.
- Changed test runner from nose to py.test. (PR: #1024)

23.13 v0.19.5 (2014-12-23)

Today is Mopidy's five year anniversary. We're celebrating with a bugfix release and are looking forward to the next five years!

- Config: Support UTF-8 in extension's default config. If an extension with non-ASCII characters in its default config was installed, and Mopidy didn't already have a config file, Mopidy would crashed when trying to create the initial config file based on the default config of all available extensions. (Fixes: discuss.mopidy.com/t/428)
- Extensions: Fix crash when unpacking data from `pkg_resources.VersionConflict` created with a single argument. (Fixes: #911)
- Models: Hide empty collections from `repr()` representations.
- Models: Field values are no longer stored on the model instance when the value matches the default value for the field. This makes two models equal when they have a field which in one case is implicitly set to the default value and in the other case explicitly set to the default value, but with otherwise equal fields. (Fixes: #837)
- Models: Changed the default value of `mopidy.models.Album.num_tracks`, `mopidy.models.Track.track_no`, and `mopidy.models.Track.last_modified` from `0` to `None`.
- Core: When skipping to the next track in consume mode, remove the skipped track from the tracklist. This is consistent with the original MPD server's behavior. (Fixes: #902)
- Local: Fix scanning of modified files. (PR: #904)
- MPD: Re-enable browsing of empty directories. (PR: #906)
- MPD: Remove track comments from responses. They are not included by the original MPD server, and this works around #881. (PR: #882)
- HTTP: Errors while starting HTTP apps are logged instead of crashing the HTTP server. (Fixes: #875)

23.14 v0.19.4 (2014-09-01)

Bug fix release.

- Configuration: `mopidy --config` now supports directories.
- Logging: Fix that some loggers would be disabled if `logging/config_file` was set. (Fixes: #740)
- Quit process with exit code 1 when stopping because of a backend, frontend, or mixer initialization error.
- Backend API: Update `mopidy.backend.LibraryProvider.browse()` signature and docs to match how the core use the backend's browse method. (Fixes: #833)
- Local library API: Add `mopidy.local.Library.ROOT_DIRECTORY_URI` constant for use by implementors of `mopidy.local.Library.browse()`. (Related to: #833)

- HTTP frontend: Guard against double close of WebSocket, which causes an `AttributeError` on Tornado < 3.2.
- MPD frontend: Make the `list` command return albums when sending 3 arguments. This was incorrectly returning artists after the MPD command changes in 0.19.0. (Fixes: #817)
- MPD frontend: Fix a race condition where two threads could try to free the same data simultaneously. (Fixes: #781)

23.15 v0.19.3 (2014-08-03)

Bug fix release.

- Audio: Fix negative track length for radio streams. (Fixes: #662, PR: #796)
- Audio: Tell GStreamer to not pick Jack sink. (Fixes: #604)
- Zeroconf: Fix discovery by adding `.local` to the announced hostname. (PR: #795)
- Zeroconf: Fix intermittent DBus/Avahi exception.
- Extensions: Fail early if trying to setup an extension which doesn't implement the *`mopidy.ext.Extension.setup()`* method. (Fixes: #813)

23.16 v0.19.2 (2014-07-26)

Bug fix release, directly from the Mopidy development sprint at EuroPython 2014 in Berlin.

- Audio: Make *`audio/mixer_volume`* work on the software mixer again. This was broken with the mixer changes in 0.19.0. (Fixes: #791)
- HTTP frontend: When using Tornado 4.0, allow WebSocket requests from other hosts. (Fixes: #788)
- MPD frontend: Fix crash when MPD commands are called with the wrong number of arguments. This was broken with the MPD command changes in 0.19.0. (Fixes: #789)

23.17 v0.19.1 (2014-07-23)

Bug fix release.

- Dependencies: Mopidy now requires Tornado >= 2.3, instead of >= 3.1. This should make Mopidy continue to work on Debian/Raspbian stable, where Tornado 2.3 is the newest version available.
- HTTP frontend: Add missing string interpolation placeholder.
- Development: `mopidy --version` and *`mopidy.core.Core.get_version()`* now returns the correct version when Mopidy is run from a Git repo other than Mopidy's own. (Related to #706)

23.18 v0.19.0 (2014-07-21)

The focus of 0.19 have been on improving the MPD implementation, replacing GStreamer mixers with our own mixer API, and on making web clients installable with `pip`, like any other Mopidy extension.

Since the release of 0.18, we've closed or merged 53 issues and pull requests through about 445 commits by *12 people*, including five new guys. Thanks to everyone that has contributed!

Dependencies

- Mopidy now requires Tornado >= 3.1.
- Mopidy no longer requires CherryPy or ws4py. Previously, these were optional dependencies required for the HTTP frontend to work.

Backend API

- *Breaking change:* Imports of the backend API from `mopidy.backends` no longer works. The new API introuced in v0.18 is now required. Most extensions already use the new API location.

Commands

- The `mopidy-convert-config` tool for migrating the `setings.py` configuration file used by Mopidy up until 0.14 to the new config file format has been removed after over a year of trusty service. If you still need to convert your old `settings.py` configuration file, do so using and older release, like Mopidy 0.18, or migrate the configuration to the new format by hand.

Configuration

- Add `optional=True` support to `mopidy.config.Boolean`.

Logging

- Fix proper decoding of exception messages that depends on the user's locale.
- Colorize logs depending on log level. This can be turned off with the new `logging/color` configuration. (Fixes: #772)

Extension support

- *Breaking change:* Removed the `Extension` methods that were deprecated in 0.18: `get_backend_classes()`, `get_frontend_classes()`, and `register_gstreamer_elements()`. Use `mopidy.ext.Extension.setup()` instead, as most extensions already do.

Audio

- *Breaking change:* Removed support for GStreamer mixers. GStreamer 1.x does not support volume control, so we changed to use software mixing by default in v0.17.0. Now, we're removing support for all other GStreamer mixers and are reintroducing mixers as something extensions can provide independently of GStreamer. (Fixes: #665, PR: #760)
- *Breaking change:* Changed the `audio/mixer` config value to refer to Mopidy mixer extensions instead of GStreamer mixers. The default value, `software`, still has the same behavior. All other values will either no longer work or will at the very least require you to install an additional extension.
- Changed the `audio/mixer_volume` config value behavior from affecting GStreamer mixers to affecting Mopidy mixer extensions instead. The end result should be the same without any changes to this config value.
- Deprecated the `audio/mixer_track` config value. This config value is no longer in use. Mixer extensions that need additional configuration handle this themselves.
- Use *Proxy configuration* when streaming media from the Internet. (Partly fixing #390)
- Fix proper decoding of exception messages that depends on the user's locale.
- Fix recognition of ASX and XSPF playlists with tags in all caps or with carriage return line endings. (Fixes: #687)
- Support simpler ASX playlist variant with `<ENTRY>` elements without children.

- Added `target_state` attribute to the audio layer's *`state_changed()`* event. Currently, it is `None` except when we're paused because of buffering. Then the new field exposes our target state after buffering has completed.

Mixers

- Added new *`mopidy.mixer.Mixer`* API which can be implemented by extensions.
- Created a bundled extension, *Mopidy-SoftwareMixer*, for controlling volume in software in GStreamer's pipeline. This is Mopidy's default mixer. To use this mixer, set the *`audio/mixer`* config value to `software`.
- Created an external extension, Mopidy-ALSAMixer, for controlling volume with hardware through ALSA. To use this mixer, install the extension, and set the *`audio/mixer`* config value to `alsamixer`.

HTTP frontend

- CherryPy and ws4py have been replaced with Tornado. This will hopefully reduce CPU usage on OS X (#445) and improve error handling in corner cases, like when returning from suspend (#718).
- Added support for packaging web clients as Mopidy extensions and installing them using pip. See the *HTTP server side API* for details. (Fixes: #440)
- Added web page at `/mopidy/` which lists all web clients installed as Mopidy extensions. (Fixes: #440)
- Added support for extending the HTTP frontend with additional server side functionality. See *HTTP server side API* for details.
- Exposed the core API using HTTP POST requests with JSON-RPC payloads at `/mopidy/rpc`. This is the same JSON-RPC interface as is exposed over the WebSocket at `/mopidy/ws`, so you can run any core API command.

  The HTTP POST interfaces does not give you access to events from Mopidy, like the WebSocket does. The WebSocket interface is still recommended for web clients. The HTTP POST interface may be easier to use for simpler programs, that just needs to query the currently playing track or similar. See *HTTP POST API* for details.
- If Zeroconf is enabled, we now announce the `_mopidy-http._tcp` service in addition to `_http._tcp`. This is to make it easier to automatically find Mopidy's HTTP server among other Zeroconf-published HTTP servers on the local network.

Mopidy.js client library

This version has been released to npm as Mopidy.js v0.4.0.

- Update Mopidy.js to use when.js 3. If you maintain a Mopidy client, you should review the differences between when.js 2 and 3 and the when.js debugging guide.
- All of Mopidy.js' promise rejection values are now of the Error type. This ensures that all JavaScript VMs will show a useful stack trace if a rejected promise's value is used to throw an exception. To allow catch clauses to handle different errors differently, server side errors are of the type `Mopidy.ServerError`, and connection related errors are of the type `Mopidy.ConnectionError`.
- Add support for method calls with by-name arguments. The old calling convention, `by-position-only`, is still the default, but this will change in the future. A warning is logged to the console if you don't explicitly select a calling convention. See the *Mopidy.js JavaScript library* docs for details.

MPD frontend

- Proper command tokenization for MPD requests. This replaces the old regex based system with an MPD protocol specific tokenizer responsible for breaking requests into pieces before the handlers have at them. (Fixes: #591 and #592)
- Updated command handler system. As part of the tokenizer cleanup we've updated how commands are registered and making it simpler to create new handlers.

- Simplified a bunch of handlers. All the "browse" type commands now use a common browse helper under the hood for less repetition. Likewise the query handling of "search" commands has been somewhat simplified.
- Adds placeholders for missing MPD commands, preparing the way for bumping the protocol version once they have been added.
- Respond to all pending requests before closing connection. (PR: #722)
- Stop incorrectly catching *LookupError* in command handling. (Fixes: #741)
- Browse support for playlists and albums has been added. (PR: #749, #754)
- The `lsinfo` command now returns browse results before local playlists. This is helpful as not all clients sort the returned items. (PR: #755)
- Browse now supports different entries with identical names. (PR: #762)
- Search terms that are empty or consists of only whitespace are no longer included in the search query sent to backends. (PR: #758)

Local backend

- The JSON local library backend now logs a friendly message telling you about `mopidy local scan` if you don't have a local library cache. (Fixes: #711)
- The `local scan` command now use multiple threads to walk the file system and check files' modification time. This speeds up scanning, escpecially when scanning remote file systems over e.g. NFS.
- the `local scan` command now creates necessary folders if they don't already exist. Previously, this was only done by the Mopidy server, so doing a `local scan` before running the server the first time resulted in a crash. (Fixes: #703)
- Fix proper decoding of exception messages that depends on the user's locale.

Stream backend

- Add config value *`stream/metadata_blacklist`* to blacklist certain URIs we should not open to read metadata from before they are opened for playback. This is typically needed for services that invalidate URIs after a single use. (Fixes: #660)

23.19 v0.18.3 (2014-02-16)

Bug fix release.

- Fix documentation build.

23.20 v0.18.2 (2014-02-16)

Bug fix release.

- We now log warnings for wrongly configured extensions, and clearly label them in `mopidy config`, but does no longer stop Mopidy from starting because of misconfigured extensions. (Fixes: #682)
- Fix a crash in the server side WebSocket handler caused by connection problems with clients. (Fixes: #428, #571)
- Fix the `time_position` field of the `track_playback_ended` event, which has been always 0 since v0.18.0. This made scrobbles by Mopidy-Scrobbler not be persisted by Last.fm, because Mopidy reported that you listened to 0 seconds of each track. (Fixes: #674)

- Fix the log setup so that it is possible to increase the amount of logging from a specific logger using the `loglevels` config section. (Fixes: #684)
- Serialization of *`Playlist`* models with the `last_modified` field set to a `datetime.datetime` instance did not work. The type of *`mopidy.models.Playlist.last_modified`* has been redefined from a `datetime.datetime` instance to the number of milliseconds since Unix epoch as an integer. This makes serialization of the time stamp simpler.
- Minor refactor of the MPD server context so that Mopidy's MPD protocol implementation can easier be reused. (Fixes: #646)
- Network and signal handling has been updated to play nice on Windows systems.

23.21 v0.18.1 (2014-01-23)

Bug fix release.

- Disable extension instead of crashing if a dependency has the wrong version. (Fixes: #657)
- Make logging work to both console, debug log file, and any custom logging setup from *`logging/config_file`* at the same time. (Fixes: #661)

23.22 v0.18.0 (2014-01-19)

The focus of 0.18 have been on two fronts: the local library and browsing.

First, the local library's old tag cache file used for storing the track metadata scanned from your music collection has been replaced with a far simpler implementation using JSON as the storage format. At the same time, the local library have been made replaceable by extensions, so you can now create extensions that use your favorite database to store the metadata.

Second, we've finally implemented the long awaited "file system" browsing feature that you know from MPD. It is supported by both the MPD frontend and the local and Spotify backends. It is also used by the new Mopidy-Dirble extension to provide you with a directory of Internet radio stations from all over the world.

Since the release of 0.17, we've closed or merged 49 issues and pull requests through about 285 commits by *11 people*, including six new guys. Thanks to everyone that has contributed!

Core API

- Add *`mopidy.core.Core.version()`* for HTTP clients to manage compatibility between API versions. (Fixes: #597)
- Add *`mopidy.models.Ref`* class for use as a lightweight reference to other model types, containing just an URI, a name, and an object type. It is barely used for now, but its use will be extended over time.
- Add *`mopidy.core.LibraryController.browse()`* method for browsing a virtual file system of tracks. Backends can implement support for this by implementing *`mopidy.backend.LibraryProvider.browse()`*.
- Events emitted on play/stop, pause/resume, next/previous and on end of track has been cleaned up to work consistently. See the message of commit 1d108752f6 for the full details. (Fixes: #629)

Backend API

- Move the backend API classes from `mopidy.backends.base` to *`mopidy.backend`* and remove the `Base` prefix from the class names:
  - From `mopidy.backends.base.Backend` to *`mopidy.backend.Backend`*

  - From `mopidy.backends.base.BaseLibraryProvider` to *mopidy.backend.LibraryProvider*
  - From `mopidy.backends.base.BasePlaybackProvider` to *mopidy.backend.PlaybackProvider*
  - From `mopidy.backends.base.BasePlaylistsProvider` to *mopidy.backend.PlaylistsProvider*
  - From `mopidy.backends.listener.BackendListener` to *mopidy.backend.BackendListener*

  Imports from the old locations still works, but are deprecated.

- Add *mopidy.backend.LibraryProvider.browse()*, which can be implemented by backends that wants to expose directories of tracks in Mopidy's virtual file system.

Frontend API

- The dummy backend used for testing many frontends have moved from `mopidy.backends.dummy` to `mopidy.backend.dummy`. (PR: #984)

Commands

- Reduce amount of logging from dependencies when using *mopidy -v*. (Fixes: #593)
- Add support for additional logging verbosity levels with `mopidy -vv` and `mopidy -vvv` which increases the amount of logging from dependencies. (Fixes: #593)

Configuration

- The default for the *mopidy --config* option has been updated to include `$XDG_CONFIG_DIRS` in addition to `$XDG_CONFIG_DIR`. (Fixes #431)
- Added support for deprecating config values in order to allow for graceful removal of the no longer used config value `local/tag_cache_file`.

Extension support

- Switched to using a registry model for classes provided by extension. This allows extensions to be extended by other extensions, as needed by for example pluggable libraries for the local backend. See *mopidy.ext.Registry* for details. (Fixes #601)
- Added the new method *mopidy.ext.Extension.setup()*. This method replaces the now deprecated `get_backend_classes()`, `get_frontend_classes()`, and `register_gstreamer_elements()`.

Audio

- Added *audio/mixer_volume* to set the initial volume of mixers. This is especially useful for setting the software mixer volume to something else than the default 100%. (Fixes: #633)

Local backend

Note: After upgrading to Mopidy 0.18 you must run `mopidy local scan` to reindex your local music collection. This is due to the change of storage format.

- Added support for browsing local directories in Mopidy's virtual file system.
- Finished the work on creating pluggable libraries. Users can now reconfigure Mopidy to use alternate library providers of their choosing for local files. (Fixes issue #44, partially resolves #397, and causes a temporary regression of #527.)

- Switched default local library provider from a "tag cache" file that closely resembled the one used by the original MPD server to a compressed JSON file. This greatly simplifies our library code and reuses our existing model serialization code, as used by the HTTP API and web clients.
- Removed our outdated and bug-ridden "tag cache" local library implementation.
- Added the config value *`local/library`* to select which library to use. It defaults to `json`, which is the only local library bundled with Mopidy.
- Added the config value *`local/data_dir`* to have a common config for where to store local library data. This is intended to avoid every single local library provider having to have it's own config value for this.
- Added the config value *`local/scan_flush_threshold`* to control how often to tell local libraries to store changes when scanning local music.

Streaming backend

- Add live lookup of URI metadata. (Fixes #540)
- Add support for extended M3U playlist, meaning that basic track metadata stored in playlists will be used by Mopidy.

HTTP frontend

- Upgrade Mopidy.js dependencies and add support for using Mopidy.js with Browserify. This version has been released to npm as Mopidy.js v0.2.0. (Fixes: #609)

MPD frontend

- Make the `lsinfo`, `listall`, and `listallinfo` commands support browsing of Mopidy's virtual file system. (Fixes: #145)
- Empty commands now return a `ACK [5@0] {} No command given` error instead of `OK`. This is consistent with the original MPD server implementation.

Internal changes

- Events from the audio actor, backends, and core actor are now emitted asyncronously through the GObject event loop. This should resolve the issue that has blocked the merge of the EOT-vs-EOS fix for a long time.

23.23 v0.17.0 (2013-11-23)

The focus of 0.17 has been on introducing subcommands to the `mopidy` command, making it possible for extensions to add subcommands of their own, and to improve the default config file when starting Mopidy the first time. In addition, we've grown support for Zeroconf publishing of the MPD and HTTP servers, and gotten a much faster scanner. The scanner now also scans some additional tags like composers and performers.

Since the release of 0.16, we've closed or merged 22 issues and pull requests through about 200 commits by *five people*, including one new contributor.

Commands

- Switched to subcommands for the `mopidy` command , this implies the following changes: (Fixes: #437)

Old command	New command
mopidy –show-deps	mopidy deps
mopidy –show-config	mopidy config
mopidy-scan	mopidy local scan

- Added hooks for extensions to create their own custom subcommands and converted `mopidy-scan` as a first user of the new API. (Fixes: #436)

Configuration

- When `mopidy` is started for the first time we create an empty *$XDG_CONFIG_DIR*`/mopidy/mopidy.conf` file. We now populate this file with the default config for all installed extensions so it'll be easier to set up Mopidy without looking through all the documentation for relevant config values. (Fixes: #467)

Core API

- The *`Track`* model has grown fields for `composers`, `performers`, `genre`, and `comment`.
- The search field `track` has been renamed to `track_name` to avoid confusion with `track_no`. (Fixes: #535)
- The signature of the tracklist's *`filter()`* and *`remove()`* methods have changed. Previously, they expected e.g. `tracklist.filter(tlid=17)`. Now, the value must always be a list, e.g. `tracklist.filter(tlid=[17])`. This change allows you to get or remove multiple tracks with a single call, e.g. `tracklist.remove(tlid=[1, 2, 7])`. This is especially useful for web clients, as requests can be batched. This also brings the interface closer to the library's *`find_exact()`* and *`search()`* methods.

Audio

- Change default volume mixer from `autoaudiomixer` to `software`. GStreamer 1.x does not support volume control, so we're changing to use software mixing by default, as that may be the only thing we'll support in the future when we upgrade to GStreamer 1.x.

Local backend

- Library scanning has been switched back from GStreamer's discoverer to our custom implementation due to various issues with GStreamer 0.10's built in scanner. This also fixes the scanner slowdown. (Fixes: #565)
- When scanning, we no longer default the album artist to be the same as the track artist. Album artist is now only populated if the scanned file got an explicit album artist set.
- The scanner will now extract multiple artists from files with multiple artist tags.
- The scanner will now extract composers and performers, as well as genre, bitrate, and comments. (Fixes: #577)
- Fix scanner so that time of last modification is respected when deciding which files can be skipped when scanning the music collection for changes.
- The scanner now ignores the capitalization of file extensions in *`local/excluded_file_extensions`*, so you no longer need to list both `.jpg` and `.JPG` to ignore JPEG files when scanning. (Fixes: #525)
- The scanner now by default ignores `*.nfo` and `*.html` files too.

MPD frontend

- The MPD service is now published as a Zeroconf service if avahi-daemon is running on the system. Some MPD clients will use this to present Mopidy as an available server on the local network without needing any configuration. See the *`mpd/zeroconf`* config value to change the service name or disable the service. (Fixes: #39)
- Add support for `composer`, `performer`, `comment`, `genre`, and `performer`. These tags can be used with `list ...`, `search ...`, and `find ...` and their variants, and are supported in the `any` tag also
- The `bitrate` field in the `status` response is now always an integer. This follows the behavior of the original MPD server. (Fixes: #577)

HTTP frontend

- The HTTP service is now published as a Zeroconf service if avahi-daemon is running on the system. Some browsers will present HTTP Zeroconf services on the local network as "local sites" bookmarks. See the *`http/zeroconf`* config value to change the service name or disable the service. (Fixes: #39)

DBUS/MPRIS

- The `mopidy` process now registers it's GObject event loop as the default eventloop for dbus-python. (Fixes: mopidy-mpris#2)

23.24 v0.16.1 (2013-11-02)

This is very small release to get Mopidy's Debian package ready for inclusion in Debian.

Commands

- Fix removal of last dir level in paths to dependencies in `mopidy --show-deps` output.
- Add manpages for all commands.

Local backend

- Fix search filtering by track number that was added in 0.16.0.

MPD frontend

- Add support for `list "albumartist" ...` which was missed when `find` and `search` learned to handle `albumartist` in 0.16.0. (Fixes: #553)

23.25 v0.16.0 (2013-10-27)

The goals for 0.16 were to add support for queuing playlists of e.g. radio streams directly to Mopidy, without manually extracting the stream URLs from the playlist first, and to move the Spotify, Last.fm, and MPRIS support out to independent Mopidy extensions, living outside the main Mopidy repo. In addition, we've seen some cleanup to the playback vs tracklist part of the core API, which will require some changes for users of the HTTP/JavaScript APIs, as well as the addition of audio muting to the core API. To speed up the *development of new extensions*, we've added a cookiecutter project to get the skeleton of a Mopidy extension up and running in a matter of minutes. Read below for all the details and for links to issues with even more details.

Since the release of 0.15, we've closed or merged 31 issues and pull requests through about 200 commits by *five people*, including three new contributors.

Dependencies

Parts of Mopidy have been moved to their own external extensions. If you want Mopidy to continue to work like it used to, you may have to install one or more of the following extensions as well:

- The Spotify backend has been moved to Mopidy-Spotify.
- The Last.fm scrobbler has been moved to Mopidy-Scrobbler.
- The MPRIS frontend has been moved to Mopidy-MPRIS.

Core

- Parts of the functionality in *`mopidy.core.PlaybackController`* have been moved to *`mopidy.core.TracklistController`*:

Old location	New location
playback.get_consume()	tracklist.get_consume()
playback.set_consume(v)	tracklist.set_consume(v)
playback.consume	tracklist.consume
playback.get_random()	tracklist.get_random()
playback.set_random(v)	tracklist.set_random(v)
playback.random	tracklist.random
playback.get_repeat()	tracklist.get_repeat()
playback.set_repeat(v)	tracklist.set_repeat(v)
playback.repeat	tracklist.repeat
playback.get_single()	tracklist.get_single()
playback.set_single(v)	tracklist.set_single(v)
playback.single	tracklist.single
playback.get_tracklist_position()	tracklist.index(tl_track)
playback.tracklist_position	tracklist.index(tl_track)
playback.get_tl_track_at_eot()	tracklist.eot_track(tl_track)
playback.tl_track_at_eot	tracklist.eot_track(tl_track)
playback.get_tl_track_at_next()	tracklist.next_track(tl_track)
playback.tl_track_at_next	tracklist.next_track(tl_track)
playback.get_tl_track_at_previous()	tracklist.previous_track(tl_track)
playback.tl_track_at_previous	tracklist.previous_track(tl_track)

The `tl_track` argument to the last four new functions are used as the reference `tl_track` in the tracklist to find e.g. the next track. Usually, this will be *`current_tl_track`*.

- Added *`mopidy.core.PlaybackController.mute`* for muting and unmuting audio. (Fixes: #186)
- Added *`mopidy.core.CoreListener.mute_changed()`* event that is triggered when the mute state changes.
- In "random" mode, after a full playthrough of the tracklist, playback continued from the last track played to the end of the playlist in non-random order. It now stops when all tracks have been played once, unless "repeat" mode is enabled. (Fixes: #453)
- In "single" mode, after a track ended, playback continued with the next track in the tracklist. It now stops after playing a single track, unless "repeat" mode is enabled. (Fixes: #496)

Audio

- Added support for parsing and playback of playlists in GStreamer. For end users this basically means that you can now add a radio playlist to Mopidy and we will automatically download it and play the stream inside it. Currently we support M3U, PLS, XSPF and ASX files. Also note that we can currently only play the first stream in the playlist.
- We now handle the rare case where an audio track has max volume equal to min. This was causing divide by zero errors when scaling volumes to a zero to hundred scale. (Fixes: #525)
- Added support for muting audio without setting the volume to 0. This works both for the software and hardware mixers. (Fixes: #186)

Local backend

- Replaced our custom media library scanner with GStreamer's builtin scanner. This should make scanning less error prone and faster as timeouts should be infrequent. (Fixes: #198)
- Media files with less than 100ms duration are now excluded from the library.

- Media files with the file extensions `.jpeg`, `.jpg`, `.png`, `.txt`, and `.log` are now skipped by the media library scanner. You can change the list of excluded file extensions by setting the *`local/excluded_file_extensions`* config value. (Fixes: #516)
- Unknown URIs found in playlists are now made into track objects with the URI set instead of being ignored. This makes it possible to have playlists with e.g. HTTP radio streams and not just `local:track:...` URIs. This used to work, but was broken in Mopidy 0.15.0. (Fixes: #527)
- Fixed crash when playing `local:track:...` URIs which contained non-ASCII chars after uridecode.
- Removed media files are now also removed from the in-memory media library when the media library is reloaded from disk. (Fixes: #500)

MPD frontend

- Made the formerly unused commands `outputs`, `enableoutput`, and `disableoutput` mute/unmute audio. (Related to: #186)
- The MPD command `list` now works with `"albumartist"` as its second argument, e.g. `list "album" "albumartist" "anartist"`. (Fixes: #468)
- The MPD commands `find` and `search` now accepts `albumartist` and `track` (this is the track number, not the track name) as field types to limit the search result with.
- The MPD command `count` is now implemented. It accepts the same type of arguments as `find` and `search`, but returns the number of tracks and their total playtime instead.

Extension support

- A cookiecutter project for quickly creating new Mopidy extensions have been created. You can find it at cookiecutter-mopidy-ext. (Fixes: #522)

23.26 v0.15.0 (2013-09-19)

A release with a number of small and medium fixes, with no specific focus.

Dependencies

- Mopidy no longer supports Python 2.6. Currently, the only Python version supported by Mopidy is Python 2.7. We're continuously working towards running Mopidy on Python 3. (Fixes: #344)

Command line options

- Converted from the optparse to the argparse library for handling command line options.
- `mopidy --show-config` will now take into consideration any *`mopidy --option`* arguments appearing later on the command line. This helps you see the effective configuration for runs with the same `mopidy --options` arguments.

Audio

- Added support for audio visualization. `audio/visualizer` can now be set to GStreamer visualizers.
- Properly encode localized mixer names before logging.

Local backend

- An album's number of discs and a track's disc number are now extracted when scanning your music collection.
- The scanner now gives up scanning a file after a second, and continues with the next file. This fixes some hangs on non-media files, like logs. (Fixes: #476, #483)

- Added support for pluggable library updaters. This allows extension writers to start providing their own custom libraries instead of being stuck with just our tag cache as the only option.
- Converted local backend to use new `local:playlist:path` and `local:track:path` URI scheme. Also moves support of `file://` to streaming backend.

Spotify backend

- Prepend playlist folder names to the playlist name, so that the playlist hierarchy from your Spotify account is available in Mopidy. (Fixes: #62)
- Fix proxy config values that was broken with the config system change in 0.14. (Fixes: #472)

MPD frontend

- Replace newline, carriage return and forward slash in playlist names. (Fixes: #474, #480)
- Accept `listall` and `listallinfo` commands without the URI parameter. The methods are still not implemented, but now the commands are accepted as valid.

HTTP frontend

- Fix too broad truth test that caused *`mopidy.models.TlTrack`* objects with `tlid` set to `0` to be sent to the HTTP client without the `tlid` field. (Fixes: #501)
- Upgrade Mopidy.js dependencies. This version has been released to npm as Mopidy.js v0.1.1.

Extension support

- `mopidy.config.Secret` is now deserialized to unicode instead of bytes. This may require modifications to extensions.

23.27 v0.14.2 (2013-07-01)

This is a maintenance release to make Mopidy 0.14 work with pyspotify 1.11.

Dependencies

- pyspotify >= 1.9, < 2 is now required for Spotify support. In other words, you're free to upgrade to pyspotify 1.11, but it isn't a requirement.

23.28 v0.14.1 (2013-04-28)

This release addresses an issue in v0.14.0 where the new `mopidy-convert-config` tool and the new *`mopidy --option`* command line option was broken because some string operations inadvertently converted some byte strings to unicode.

23.29 v0.14.0 (2013-04-28)

The 0.14 release has a clear focus on two things: the new configuration system and extension support. Mopidy's documentation has also been greatly extended and improved.

Since the last release a month ago, we've closed or merged 53 issues and pull requests. A total of seven *authors* have contributed, including one new.

Dependencies

- setuptools or distribute is now required. We've introduced this dependency to use setuptools' entry points functionality to find installed Mopidy extensions.

New configuration system

- Mopidy has a new configuration system based on ini-style files instead of a Python file. This makes configuration easier for users, and also makes it possible for Mopidy extensions to have their own config sections.

  As part of this change we have cleaned up the naming of our config values.

  To ease migration we've made a tool named `mopidy-convert-config` for automatically converting the old `settings.py` to a new `mopidy.conf` file. This tool takes care of all the renamed config values as well. See `mopidy-convert-config` for details on how to use it.

- A long wanted feature: You can now enable or disable specific frontends or backends without having to redefine `FRONTENDS` or `BACKENDS` in your config. Those config values are gone completely.

Extension support

- Mopidy now supports extensions. This means that any developer now easily can create a Mopidy extension to add new control interfaces or music backends. This helps spread the maintenance burden across more developers, and also makes it possible to extend Mopidy with new backends the core developers are unable to create and/or maintain because of geo restrictions, etc. If you're interested in creating an extension for Mopidy, read up on *Extension development*.

- All of Mopidy's existing frontends and backends are now plugged into Mopidy as extensions, but they are still distributed together with Mopidy and are enabled by default.

- The NAD mixer have been moved out of Mopidy core to its own project, Mopidy-NAD. See *Extensions* for more information.

- Janez Troha has made the first two external extensions for Mopidy: a backend for playing music from Soundcloud, and a backend for playing music from a Beets music library. See *Extensions* for more information.

Command line options

- The command option `mopidy --list-settings` is now named `mopidy --show-config`.

- The command option `mopidy --list-deps` is now named `mopidy --show-deps`.

- What configuration files to use can now be specified through the command option `mopidy --config`, multiple files can be specified using colon as a separator.

- Configuration values can now be overridden through the command option `mopidy --option`. For example: `mopidy --option spotify/enabled=false`.

- The GStreamer command line options, `mopidy --gst-*` and `mopidy --help-gst` are no longer supported. To set GStreamer debug flags, you can use environment variables such as `GST_DEBUG`. Refer to GStreamer's documentation for details.

Spotify backend

- Add support for starred playlists, both your own and those owned by other users. (Fixes: #326)

- Fix crash when a new playlist is added by another Spotify client. (Fixes: #387, #425)

MPD frontend

- Playlists with identical names are now handled properly by the MPD frontend by suffixing the duplicate names with e.g. `[2]`. This is needed because MPD identify playlists by name only, while Mopidy and Spotify supports multiple playlists with the same name, and identify them using an URI. (Fixes: #114)

MPRIS frontend

- The frontend is now disabled if the `DISPLAY` environment variable is unset. This avoids some harmless error messages, that have been known to confuse new users debugging other problems.

Development

- Developers running Mopidy from a Git clone now need to run `python setup.py develop` to register the bundled extensions. If you don't do this, Mopidy will not find any frontends or backends. Note that we highly recomend you do this in a virtualenv, not system wide. As a bonus, the command also gives you a `mopidy` executable in your search path.

23.30 v0.13.0 (2013-03-31)

The 0.13 release brings small improvements and bugfixes throughout Mopidy. There are no major new features, just incremental improvement of what we already have.

Dependencies

- Pykka >= 1.1 is now required.

Core

- Removed the `mopidy.settings.DEBUG_THREAD` setting and the `--debug-thread` command line option. Sending SIGUSR1 to the Mopidy process will now always make it log tracebacks for all alive threads.
- Log a warning if a track isn't playable to make it more obvious that backend X needs backend Y to be present for playback to work.
- *`mopidy.core.TracklistController.add()`* now accepts an `uri` which it will lookup in the library and then add to the tracklist. This is helpful for e.g. web clients that doesn't want to transfer all track meta data back to the server just to add it to the tracklist when the server already got all the needed information easily available. (Fixes: #325)
- Change the following methods to accept an `uris` keyword argument:
  - *`mopidy.core.LibraryController.find_exact()`*
  - *`mopidy.core.LibraryController.search()`*

  Search queries will only be forwarded to backends handling the given URI roots, and the backends may use the URI roots to further limit what results are returned. For example, a search with `uris=['file:']` will only be processed by the local backend. A search with `uris=['file:///media/music']` will only be processed by the local backend, and, if such filtering is supported by the backend, will only return results with URIs within the given URI root.

Audio sub-system

- Make audio error logging handle log messages with non-ASCII chars. (Fixes: #347)

Local backend

- Make `mopidy-scan` work with Ogg Vorbis files. (Fixes: #275)
- Fix playback of files with non-ASCII chars in their file path. (Fixes: #353)

Spotify backend

- Let GStreamer handle time position tracking and seeks. (Fixes: #191)
- For all playlists owned by other Spotify users, we now append the owner's username to the playlist name. (Partly fixes: #114)

HTTP frontend

- Mopidy.js now works both from browsers and from Node.js environments. This means that you now can make Mopidy clients in Node.js. Mopidy.js has been published to the npm registry for easy installation in Node.js projects.
- Upgrade Mopidy.js' build system Grunt from 0.3 to 0.4.
- Upgrade Mopidy.js' dependencies when.js from 1.6.1 to 2.0.0.
- Expose *`mopidy.core.Core.get_uri_schemes()`* to HTTP clients. It is available through Mopidy.js as `mopidy.getUriSchemes()`.

MPRIS frontend

- Publish album art URIs if available.
- Publish disc number of track if available.

23.31 v0.12.0 (2013-03-12)

The 0.12 release has been delayed for a while because of some issues related some ongoing GStreamer cleanup we didn't invest enough time to finish. Finally, we've come to our senses and have now cherry-picked the good parts to bring you a new release, while postponing the GStreamer changes to 0.13. The release adds a new backend for playing audio streams, as well as various minor improvements throughout Mopidy.

- Make Mopidy work on early Python 2.6 versions. (Fixes: #302)
  - `optparse` fails if the first argument to `add_option` is a unicode string on Python < 2.6.2rc1.
  - `foo(**data)` fails if the keys in `data` is unicode strings on Python < 2.6.5rc1.

Audio sub-system

- Improve selection of mixer tracks for volume control. (Fixes: #307)

Local backend

- Make `mopidy-scan` support symlinks.

Stream backend

We've added a new backend for playing audio streams, the `stream backend`. It is activated by default. The stream backend supports the intersection of what your GStreamer installation supports and what protocols are included in the `mopidy.settings.STREAM_PROTOCOLS` setting.

Current limitations:

- No metadata about the current track in the stream is available.
- Playlists are not parsed, so you can't play e.g. a M3U or PLS file which contains stream URIs. You need to extract the stream URL from the playlist yourself. See #303 for progress on this.

Core API

- *`mopidy.core.PlaylistsController.get_playlists()`* now accepts an argument `include_tracks`. This defaults to `True`, which has the same old behavior. If set to `False`, the tracks are stripped from the playlists before they are returned. This can be used to limit the amount of data returned if the response is to be passed out of the application, e.g. to a web client. (Fixes: #297)

Models

- Add *`mopidy.models.Album.images`* field for including album art URIs. (Partly fixes #263)
- Add *`mopidy.models.Track.disc_no`* field. (Partly fixes: #286)

- Add *mopidy.models.Album.num_discs* field. (Partly fixes: #286)

23.32 v0.11.1 (2012-12-24)

Spotify search was broken in 0.11.0 for users of Python 2.6. This release fixes it. If you're using Python 2.7, v0.11.0 and v0.11.1 should be equivalent.

23.33 v0.11.0 (2012-12-24)

In celebration of Mopidy's three year anniversary December 23, we're releasing Mopidy 0.11. This release brings several improvements, most notably better search which now includes matching artists and albums from Spotify in the search results.

Settings

- The settings validator now complains if a setting which expects a tuple of values (e.g. `mopidy.settings.BACKENDS`, `mopidy.settings.FRONTENDS`) has a non-iterable value. This typically happens because the setting value contains a single value and one has forgotten to add a comma after the string, making the value a tuple. (Fixes: #278)

Spotify backend

- Add `mopidy.settings.SPOTIFY_TIMEOUT` setting which allows you to control how long we should wait before giving up on Spotify searches, etc.
- Add support for looking up albums, artists, and playlists by URI in addition to tracks. (Fixes: #67)

  As an example of how this can be used, you can try the the following MPD commands which now all adds one or more tracks to your tracklist:

  ```
  add "spotify:track:1mwt9hzaH7idmC5UCoOUkz"
  add "spotify:album:3gpHG5MGwnipnap32lFYvI"
  add "spotify:artist:5TgQ66WuWkoQ2xYxaSTnVP"
  add "spotify:user:p3.no:playlist:0XX6tamRiqEgh3t6FPFEkw"
  ```

- Increase max number of tracks returned by searches from 100 to 200, which seems to be Spotify's current max limit.

Local backend

- Load track dates from tag cache.
- Add support for searching by track date.

MPD frontend

- Add `mopidy.settings.MPD_SERVER_CONNECTION_TIMEOUT` setting which controls how long an MPD client can stay inactive before the connection is closed by the server.
- Add support for the `findadd` command.
- Updated to match the MPD 0.17 protocol (Fixes: #228):
  - Add support for `seekcur` command.
  - Add support for `config` command.
  - Add support for loading a range of tracks from a playlist to the `load` command.
  - Add support for `searchadd` command.

  - Add support for `searchaddpl` command.
  - Add empty stubs for channel commands for client to client communication.
- Add support for search by date.
- Make `seek` and `seekid` not restart the current track before seeking in it.
- Include fake tracks representing albums and artists in the search results. When these are added to the tracklist, they expand to either all tracks in the album or all tracks by the artist. This makes it easy to play full albums in proper order, which is a feature that have been frequently requested. (Fixes: #67, #148)

Internal changes

Models:

- Specified that `mopidy.models.Playlist.last_modified` should be in UTC.
- Added `mopidy.models.SearchResult` model to encapsulate search results consisting of more than just tracks.

Core API:

- Change the following methods to return `mopidy.models.SearchResult` objects which can include both track results and other results:
  - `mopidy.core.LibraryController.find_exact()`
  - `mopidy.core.LibraryController.search()`
- Change the following methods to accept either a dict with filters or kwargs. Previously they only accepted kwargs, which made them impossible to use from the Mopidy.js through JSON-RPC, which doesn't support kwargs.
  - `mopidy.core.LibraryController.find_exact()`
  - `mopidy.core.LibraryController.search()`
  - `mopidy.core.PlaylistsController.filter()`
  - `mopidy.core.TracklistController.filter()`
  - `mopidy.core.TracklistController.remove()`
- Actually trigger the `mopidy.core.CoreListener.volume_changed()` event.
- Include the new volume level in the `mopidy.core.CoreListener.volume_changed()` event.
- The `track_playback_{paused,resumed,started,ended}` events now include a `mopidy.models.TlTrack` instead of a `mopidy.models.Track`.

Audio:

- Mixers with fewer than 100 volume levels could report another volume level than what you just set due to the conversion between Mopidy's 0-100 range and the mixer's range. Now Mopidy returns the recently set volume if the mixer reports a volume level that matches the recently set volume, otherwise the mixer's volume level is rescaled to the 1-100 range and returned.

23.34 v0.10.0 (2012-12-12)

We've added an HTTP frontend for those wanting to build web clients for Mopidy!

Dependencies

- pyspotify >= 1.9, < 1.11 is now required for Spotify support. In other words, you're free to upgrade to pyspotify 1.10, but it isn't a requirement.

Documentation

- Added installation instructions for Fedora.

Spotify backend

- Save a lot of memory by reusing artist, album, and track models.
- Make sure the playlist loading hack only runs once.

Local backend

- Change log level from error to warning on messages emitted when the tag cache isn't found and a couple of similar cases.
- Make `mopidy-scan` ignore invalid dates, e.g. dates in years outside the range 1-9999.
- Make `mopidy-scan` accept `-q`/`--quiet` and `-v`/`--verbose` options to control the amount of logging output when scanning.
- The scanner can now handle files with other encodings than UTF-8. Rebuild your tag cache with `mopidy-scan` to include tracks that may have been ignored previously.

HTTP frontend

- Added new optional HTTP frontend which exposes Mopidy's core API through JSON-RPC 2.0 messages over a WebSocket. See *HTTP JSON-RPC API* for further details.
- Added a JavaScript library, Mopidy.js, to make it easier to develop web based Mopidy clients using the new HTTP frontend.

Bug fixes

- #256: Fix crash caused by non-ASCII characters in paths returned from `glib`. The bug can be worked around by overriding the settings that includes offending `$XDG_` variables.

23.35 v0.9.0 (2012-11-21)

Support for using the local and Spotify backends simultaneously have for a very long time been our most requested feature. Finally, it's here!

Dependencies

- pyspotify >= 1.9, < 1.10 is now required for Spotify support.

Documentation

- New *Installation* guides, organized by OS and distribution so that you can follow one concise list of instructions instead of jumping around the docs to look for instructions for each dependency.
- Moved *Raspberry Pi* howto from the wiki to the docs.
- Updated *MPD clients* overview.
- Added *MPRIS clients* and *UPnP clients* overview.

Multiple backends support

- Both the local backend and the Spotify backend are now turned on by default. The local backend is listed first in the `mopidy.settings.BACKENDS` setting, and are thus given the highest priority in e.g. search results, meaning that we're listing search hits from the local backend first. If you want to prioritize the backends in another way, simply set `BACKENDS` in your own settings file and reorder the backends.

There are no other setting changes related to the local and Spotify backends. As always, see `mopidy.settings` for the full list of available settings.

Spotify backend

- The Spotify backend now includes release year and artist on albums.
- #233: The Spotify backend now returns the track if you search for the Spotify track URI.
- Added support for connecting to the Spotify service through an HTTP or SOCKS proxy, which is supported by pyspotify >= 1.9.
- Subscriptions to other Spotify user's "starred" playlists are ignored, as they currently isn't fully supported by pyspotify.

Local backend

- #236: The `mopidy-scan` command failed to include tags from ALAC files (Apple lossless) because it didn't support multiple tag messages from GStreamer per track it scanned.
- Added support for search by filename to local backend.

MPD frontend

- #218: The MPD commands `listplaylist` and `listplaylistinfo` now accepts unquoted playlist names if they don't contain spaces.
- #246: The MPD command `list album artist ""` and similar `search`, `find`, and `list` commands with empty filter values caused a `LookupError`, but should have been ignored by the MPD server.
- The MPD frontend no longer lowercases search queries. This broke e.g. search by URI, where casing may be essential.
- The MPD command `plchanges` always returned the entire playlist. It now returns an empty response when the client has seen the latest version.
- The MPD commands `search` and `find` now allows the key `file`, which is used by ncmpcpp instead of `filename`.
- The MPD commands `search` and `find` now allow search query values to be empty strings.
- The MPD command `listplaylists` will no longer return playlists without a name. This could crash ncmpcpp.
- The MPD command `list` will no longer return artist names, album names, or dates that are blank.
- The MPD command `decoders` will now return an empty response instead of a "not implemented" error to make the ncmpcpp browse view work the first time it is opened.

MPRIS frontend

- The MPRIS playlists interface is now supported by our MPRIS frontend. This means that you now can select playlists to queue and play from the Ubuntu Sound Menu.

Audio mixers

- Made the `NAD mixer` responsive to interrupts during amplifier calibration. It will now quit immediately, while previously it completed the calibration first, and then quit, which could take more than 15 seconds.

Developer support

- Added optional background thread for debugging deadlocks. When the feature is enabled via the `--debug-thread` option or `mopidy.settings.DEBUG_THREAD` setting a `SIGUSR1` signal will dump the traceback for all running threads.
- The settings validator will now allow any setting prefixed with `CUSTOM_` to exist in the settings file.

Internal changes

Internally, Mopidy have seen a lot of changes to pave the way for multiple backends and the future HTTP frontend.

- A new layer and actor, "core", has been added to our stack, inbetween the frontends and the backends. The responsibility of the core layer and actor is to take requests from the frontends, pass them on to one or more backends, and combining the response from the backends into a single response to the requesting frontend.

  Frontends no longer know anything about the backends. They just use the *mopidy.core — Core API*.
- The dependency graph between the core controllers and the backend providers have been straightened out, so that we don't have any circular dependencies. The frontend, core, backend, and audio layers are now strictly separate. The frontend layer calls on the core layer, and the core layer calls on the backend layer. Both the core layer and the backends are allowed to call on the audio layer. Any data flow in the opposite direction is done by broadcasting of events to listeners, through e.g. `mopidy.core.CoreListener` and `mopidy.audio.AudioListener`.

  See *Architecture* for more details and illustrations of all the relations.
- All dependencies are now explicitly passed to the constructors of the frontends, core, and the backends. This makes testing each layer with dummy/mocked lower layers easier than with the old variant, where dependencies where looked up in Pykka's actor registry.
- All properties in the core API now got getters, and setters if setting them is allowed. They are not explictly listed in the docs as they have the same behavior as the documented properties, but they are available and may be used. This is useful for the future HTTP frontend.

Models:

- Added `mopidy.models.Album.date` attribute. It has the same format as the existing `mopidy.models.Track.date`.
- Added `mopidy.models.ModelJSONEncoder` and `mopidy.models.model_json_decoder()` for automatic JSON serialization and deserialization of data structures which contains Mopidy models. This is useful for the future HTTP frontend.

Library:

- `mopidy.core.LibraryController.find_exact()` and `mopidy.core.LibraryController.search()` now returns plain lists of tracks instead of playlist objects.
- `mopidy.core.LibraryController.lookup()` now returns a list of tracks instead of a single track. This makes it possible to support lookup of artist or album URIs which then can expand to a list of tracks.

Playback:

- The base playback provider has been updated with sane default behavior instead of empty functions. By default, the playback provider now lets GStreamer keep track of the current track's time position. The local backend simply uses the base playback provider without any changes. Any future backend that just feeds URIs to GStreamer to play can also use the base playback provider without any changes.
- Removed `mopidy.core.PlaybackController.track_at_previous`. Use `mopidy.core.PlaybackController.tl_track_at_previous` instead.
- Removed `mopidy.core.PlaybackController.track_at_next`. Use `mopidy.core.PlaybackController.tl_track_at_next` instead.

- Removed `mopidy.core.PlaybackController.track_at_eot`. Use `mopidy.core.PlaybackController.tl_track_at_eot` instead.
- Removed `mopidy.core.PlaybackController.current_tlid`. Use `mopidy.core.PlaybackController.current_tl_track` instead.

Playlists:

The playlists part of the core API has been revised to be more focused around the playlist URI, and some redundant functionality has been removed:

- Renamed "stored playlists" to "playlists" everywhere, including the core API used by frontends.
- `mopidy.core.PlaylistsController.playlists` no longer supports assignment to it. The *playlists* property on the backend layer still does, and all functionality is maintained by assigning to the playlists collections at the backend level.
- `mopidy.core.PlaylistsController.delete()` now accepts an URI, and not a playlist object.
- `mopidy.core.PlaylistsController.save()` now returns the saved playlist. The returned playlist may differ from the saved playlist, and should thus be used instead of the playlist passed to `mopidy.core.PlaylistsController.save()`.
- `mopidy.core.PlaylistsController.rename()` has been removed, since renaming can be done with `mopidy.core.PlaylistsController.save()`.
- `mopidy.core.PlaylistsController.get()` has been replaced by `mopidy.core.PlaylistsController.filter()`.
- The event `mopidy.core.CoreListener.playlist_changed()` has been changed to include the playlist that was changed.

Tracklist:

- Renamed "current playlist" to "tracklist" everywhere, including the core API used by frontends.
- Removed `mopidy.core.TracklistController.append()`. Use `mopidy.core.TracklistController.add()` instead, which is now capable of adding multiple tracks.
- `mopidy.core.TracklistController.get()` has been replaced by `mopidy.core.TracklistController.filter()`.
- `mopidy.core.TracklistController.remove()` can now remove multiple tracks, and returns the tracks it removed.
- When the tracklist is changed, we now trigger the new `mopidy.core.CoreListener.tracklist_changed()` event. Previously we triggered `mopidy.core.CoreListener.playlist_changed()`, which is intended for stored playlists, not the tracklist.

Towards Python 3 support:

- Make the entire code base use unicode strings by default, and only fall back to bytestrings where it is required. Another step closer to Python 3.

23.36 v0.8.1 (2012-10-30)

A small maintenance release to fix a bug introduced in 0.8.0 and update Mopidy to work with Pykka 1.0.

Dependencies

- Pykka >= 1.0 is now required.

Bug fixes

- #213: Fix "streaming task paused, reason not-negotiated" errors observed by some users on some Spotify tracks due to a change introduced in 0.8.0. See the issue for a patch that applies to 0.8.0.
- #216: Volume returned by the MPD command *status* contained a floating point `.0` suffix. This bug was introduced with the large audio output and mixer changes in v0.8.0 and broke the MPDroid Android client. It now returns an integer again.

23.37 v0.8.0 (2012-09-20)

This release does not include any major new features. We've done a major cleanup of how audio outputs and audio mixers work, and on the way we've resolved a bunch of related issues.

Audio output and mixer changes

- Removed multiple outputs support. Having this feature currently seems to be more trouble than what it is worth. The `mopidy.settings.OUTPUTS` setting is no longer supported, and has been replaced with `mopidy.settings.OUTPUT` which is a GStreamer bin description string in the same format as `gst-launch` expects. Default value is `autoaudiosink`. (Fixes: #81, #115, #121, #159)
- Switch to pure GStreamer based mixing. This implies that users setup a GStreamer bin with a mixer in it in `mopidy.settings.MIXER`. The default value is `autoaudiomixer`, a custom mixer that attempts to find a mixer that will work on your system. If this picks the wrong mixer you can of course override it. Setting the mixer to `None` is also supported. MPD protocol support for volume has also been updated to return -1 when we have no mixer set. `software` can be used to force software mixing.
- Removed the Denon hardware mixer, as it is not maintained.
- Updated the NAD hardware mixer to work in the new GStreamer based mixing regime. Settings are now passed as GStreamer element properties. In practice that means that the following old-style config:

  ```
  MIXER = u'mopidy.mixers.nad.NadMixer'
  MIXER_EXT_PORT = u'/dev/ttyUSB0'
  MIXER_EXT_SOURCE = u'Aux'
  MIXER_EXT_SPEAKERS_A = u'On'
  MIXER_EXT_SPEAKERS_B = u'Off'
  ```

  Now is reduced to simply:

  ```
  MIXER = u'nadmixer port=/dev/ttyUSB0 source=aux speakers-a=on speakers-b=off'
  ```

  The `port` property defaults to `/dev/ttyUSB0`, and the rest of the properties may be left out if you don't want the mixer to adjust the settings on your NAD amplifier when Mopidy is started.

Changes

- When unknown settings are encountered, we now check if it's similar to a known setting, and suggests to the user what we think the setting should have been.
- Added `--list-deps` option to the `mopidy` command that lists required and optional dependencies, their current versions, and some other information useful for debugging. (Fixes: #74)
- Added `tools/debug-proxy.py` to tee client requests to two backends and diff responses. Intended as a developer tool for checking for MPD protocol changes and various client support. Requires gevent, which currently is not a dependency of Mopidy.
- Support tracks with only release year, and not a full release date, like e.g. Spotify tracks.

- Default value of `LOCAL_MUSIC_PATH` has been updated to be `$XDG_MUSIC_DIR`, which on most systems this is set to `$HOME`. Users of local backend that relied on the old default `~/music` need to update their settings. Note that the code responsible for finding this music now also ignores UNIX hidden files and folders.
- File and path settings now support `$XDG_CACHE_DIR`, `$XDG_DATA_DIR` and `$XDG_MUSIC_DIR` substitution. Defaults for such settings have been updated to use this instead of hidden away defaults.
- Playback is now done using `playbin2` from GStreamer instead of rolling our own. This is the first step towards resolving #171.

Bug fixes

- #72: Created a Spotify track proxy that will switch to using loaded data as soon as it becomes available.
- #150: Fix bug which caused some clients to block Mopidy completely. The bug was caused by some clients sending `close` and then shutting down the connection right away. This trigged a situation in which the connection cleanup code would wait for an response that would never come inside the event loop, blocking everything else.
- #162: Fixed bug when the MPD command `playlistinfo` is used with a track position. Track position and CPID was intermixed, so it would cause a crash if a CPID matching the track position didn't exist.
- Fixed crash on lookup of unknown path when using local backend.
- #189: `LOCAL_MUSIC_PATH` and path handling in rest of settings has been updated so all of the code now uses the correct value.
- Fixed incorrect track URIs generated by M3U playlist parsing code. Generated tracks are now relative to `LOCAL_MUSIC_PATH`.
- #203: Re-add support for software mixing.

23.38 v0.7.3 (2012-08-11)

A small maintenance release to fix a crash affecting a few users, and a couple of small adjustments to the Spotify backend.

Changes

- Fixed crash when logging `IOError` exceptions on systems using languages with non-ASCII characters, like French.
- Move the default location of the Spotify cache from *~/.cache/mopidy* to *~/.cache/mopidy/spotify*. You can change this by setting `mopidy.settings.SPOTIFY_CACHE_PATH`.
- Reduce time required to update the Spotify cache on startup. One one system/Spotify account, the time from clean cache to ready for use was reduced from 35s to 12s.

23.39 v0.7.2 (2012-05-07)

This is a maintenance release to make Mopidy 0.7 build on systems without all of Mopidy's runtime dependencies, like Launchpad PPAs.

Changes

- Change from version tuple at `mopidy.VERSION` to **PEP 386** compliant version string at `mopidy.__version__` to conform to **PEP 396**.

23.40 v0.7.1 (2012-04-22)

This is a maintenance release to make Mopidy 0.7 work with pyspotify >= 1.7.

Changes

- Don't override pyspotify's `notify_main_thread` callback. The default implementation is sensible, while our override did nothing.

23.41 v0.7.0 (2012-02-25)

Not a big release with regard to features, but this release got some performance improvements over v0.6, especially for slower Atom systems. It also fixes a couple of other bugs, including one which made Mopidy crash when using GStreamer from the prereleases of Ubuntu 12.04.

Changes

- The MPD command `playlistinfo` is now faster, thanks to John Bäckstrand.
- Added the method `mopidy.backends.base.CurrentPlaylistController.length()`, `mopidy.backends.base.CurrentPlaylistController.index()`, and `mopidy.backends.base.CurrentPlaylistController.slice()` to reduce the need for copying the entire current playlist from one thread to another. Thanks to John Bäckstrand for pinpointing the issue.
- Fix crash on creation of config and cache directories if intermediate directories does not exist. This was especially the case on OS X, where `~/.config` doesn't exist for most users.
- Fix `gst.LinkError` which appeared when using newer versions of GStreamer, e.g. on Ubuntu 12.04 Alpha. (Fixes: #144)
- Fix crash on mismatching quotation in `list` MPD queries. (Fixes: #137)
- Volume is now reported to be the same as the volume was set to, also when internal rounding have been done due to `mopidy.settings.MIXER_MAX_VOLUME` has been set to cap the volume. This should make it possible to manage capped volume from clients that only increase volume with one step at a time, like ncmpcpp does.

23.42 v0.6.1 (2011-12-28)

This is a maintenance release to make Mopidy 0.6 work with pyspotify >= 1.5, which Mopidy's develop branch have supported for a long time. This should also make the Debian packages work out of the box again.

Important changes

- pyspotify 1.5 or greater is required.

Changes

- Spotify playlist folder boundaries are now properly detected. In other words, if you use playlist folders, you will no longer get lots of log messages about bad playlists.

23.43 v0.6.0 (2011-10-09)

The development of Mopidy have been quite slow for the last couple of months, but we do have some goodies to release which have been idling in the develop branch since the warmer days of the summer. This release brings support for the MPD `idle` command, which makes it possible for a client wait for updates from the server instead of polling every second. Also, we've added support for the MPRIS standard, so that Mopidy can be controlled over D-Bus from e.g. the Ubuntu Sound Menu.

Please note that 0.6.0 requires some updated dependencies, as listed under *Important changes* below.

Important changes

- Pykka 0.12.3 or greater is required.
- pyspotify 1.4 or greater is required.
- All config, data, and cache locations are now based on the XDG spec.
  - This means that your settings file will need to be moved from `~/.mopidy/settings.py` to `~/.config/mopidy/settings.py`.
  - Your Spotify cache will now be stored in `~/.cache/mopidy` instead of `~/.mopidy/spotify_cache`.
  - The local backend's `tag_cache` should now be in `~/.local/share/mopidy/tag_cache`, likewise your playlists will be in `~/.local/share/mopidy/playlists`.
  - The local client now tries to lookup where your music is via XDG, it will fall-back to `~/music` or use whatever setting you set manually.
- The MPD command `idle` is now supported by Mopidy for the following subsystems: player, playlist, options, and mixer. (Fixes: #32)
- A new frontend `mopidy.frontends.mpris` have been added. It exposes Mopidy through the MPRIS interface over D-Bus. In practice, this makes it possible to control Mopidy through the Ubuntu Sound Menu.

Changes

- Replace `mopidy.backends.base.Backend.uri_handlers` with `mopidy.backends.base.Backend.uri_schemes`, which just takes the part up to the colon of an URI, and not any prefix.
- Add Listener API, `mopidy.listeners`, to be implemented by actors wanting to receive events from the backend. This is a formalization of the ad hoc events the Last.fm scrobbler has already been using for some time.
- Replaced all of the MPD network code that was provided by asyncore with custom stack. This change was made to facilitate support for the `idle` command, and to reduce the number of event loops being used.
- Fix metadata update in Shoutcast streaming. (Fixes: #122)
- Unescape all incoming MPD requests. (Fixes: #113)
- Increase the maximum number of results returned by Spotify searches from 32 to 100.
- Send Spotify search queries to pyspotify as unicode objects, as required by pyspotify 1.4. (Fixes: #129)
- Add setting `mopidy.settings.MPD_SERVER_MAX_CONNECTIONS`. (Fixes: #134)
- Remove *destroy()* methods from backend controller and provider APIs, as it was not in use and actually not called by any code. Will reintroduce when needed.

23.44 v0.5.0 (2011-06-15)

Since last time we've added support for audio streaming to SHOUTcast servers and fixed the longstanding playlist loading issue in the Spotify backend. As always the release has a bunch of bug fixes and minor improvements.

Please note that 0.5.0 requires some updated dependencies, as listed under *Important changes* below.

Important changes

- If you use the Spotify backend, you *must* upgrade to libspotify 0.0.8 and pyspotify 1.3. If you install from APT, libspotify and pyspotify will automatically be upgraded. If you are not installing from APT, follow the instructions at *Installation*.
- If you have explicitly set the `mopidy.settings.SPOTIFY_HIGH_BITRATE` setting, you must update your settings file. The new setting is named `mopidy.settings.SPOTIFY_BITRATE` and accepts the integer values 96, 160, and 320.
- Mopidy now supports running with 1 to N outputs at the same time. This feature was mainly added to facilitate SHOUTcast support, which Mopidy has also gained. In its current state outputs can not be toggled during runtime.

Changes

- Local backend:
  - Fix local backend time query errors that where coming from stopped pipeline. (Fixes: #87)
- Spotify backend:
  - Thanks to Antoine Pierlot-Garcin's recent work on updating and improving pyspotify, stored playlists will again load when Mopidy starts. The workaround of searching and reconnecting to make the playlists appear are no longer necessary. (Fixes: #59)
  - Track's that are no longer available in Spotify's archives are now "autolinked" to corresponding tracks in other albums, just like the official Spotify clients do. (Fixes: #34)
- MPD frontend:
  - Refactoring and cleanup. Most notably, all request handlers now get an instance of `mopidy.frontends.mpd.dispatcher.MpdContext` as the first argument. The new class contains reference to any object in Mopidy the MPD protocol implementation should need access to.
  - Close the client connection when the command `close` is received.
  - Do not allow access to the command `kill`.
  - `commands` and `notcommands` now have correct output if password authentication is turned on, but the connected user has not been authenticated yet.
- Command line usage:
  - Support passing options to GStreamer. See `--help-gst` for a list of available options. (Fixes: #95)
  - Improve `--list-settings` output. (Fixes: #91)
  - Added `--interactive` for reading missing local settings from `stdin`. (Fixes: #96)
  - Improve shutdown procedure at CTRL+C. Add signal handler for `SIGTERM`, which initiates the same shutdown procedure as CTRL+C does.
- Tag cache generator:
  - Made it possible to abort **`mopidy-scan`** with CTRL+C.
  - Fixed bug regarding handling of bad dates.
  - Use `logging` instead of `print` statements.
  - Found and worked around strange WMA metadata behaviour.
- Backend API:

  - Calling on `mopidy.backends.base.playback.PlaybackController.next()` and `mopidy.backends.base.playback.PlaybackController.previous()` no longer implies that playback should be started. The playback state–whether playing, paused or stopped–will now be kept.
  - The method `mopidy.backends.base.playback.PlaybackController.change_track()` has been added. Like `next()`, and `prev()`, it changes the current track without changing the playback state.

23.45 v0.4.1 (2011-05-06)

This is a bug fix release fixing audio problems on older GStreamer and some minor bugs.

Bug fixes

- Fix broken audio on at least GStreamer 0.10.30, which affects Ubuntu 10.10. The GStreamer *appsrc* bin wasn't being linked due to lack of default caps. (Fixes: #85)
- Fix crash in `mopidy.mixers.nad` that occures at startup when the `io` module is available. We used an *eol* keyword argument which is supported by `serial.FileLike.readline()`, but not by `io.RawBaseIO.readline()`. When the `io` module is available, it is used by PySerial instead of the *FileLike* implementation.
- Fix UnicodeDecodeError in MPD frontend on non-english locale. Thanks to Antoine Pierlot-Garcin for the patch. (Fixes: #88)
- Do not create Pykka proxies that are not going to be used in *`mopidy.core`*. The underlying actor may already intentionally be dead, and thus the program may crash on creating a proxy it doesn't need. Combined with the Pykka 0.12.2 release this fixes a crash in the Last.fm frontend which may occur when all dependencies are installed, but the frontend isn't configured. (Fixes: #84)

23.46 v0.4.0 (2011-04-27)

Mopidy 0.4.0 is another release without major feature additions. In 0.4.0 we've fixed a bunch of issues and bugs, with the help of several new contributors who are credited in the changelog below. The major change of 0.4.0 is an internal refactoring which clears way for future features, and which also make Mopidy work on Python 2.7. In other words, Mopidy 0.4.0 works on Ubuntu 11.04 and Arch Linux.

Please note that 0.4.0 requires some updated dependencies, as listed under *Important changes* below. Also, the known bug in the Spotify playlist loading from Mopidy 0.3.0 is still present.

Warning: Known bug in Spotify playlist loading

There is a known bug in the loading of Spotify playlists. To avoid the bug, follow the simple workaround described at #59.

Important changes

- Mopidy now depends on Pykka >=0.12. If you install from APT, Pykka will automatically be installed. If you are not installing from APT, you may install Pykka from PyPI:

```
sudo pip install -U Pykka
```

- If you use the Spotify backend, you *should* upgrade to libspotify 0.0.7 and the latest pyspotify from the Mopidy developers. If you install from APT, libspotify and pyspotify will automatically be upgraded. If you are not installing from APT, follow the instructions at *Installation*.

Changes

- Mopidy now use Pykka actors for thread management and inter-thread communication. The immediate advantage of this is that Mopidy now works on Python 2.7, which is the default on e.g. Ubuntu 11.04. (Fixes: #66)
- Spotify backend:
  - Fixed multiple segmentation faults due to bugs in Pyspotify. Thanks to Antoine Pierlot-Garcin and Jamie Kirkpatrick for patches to Pyspotify.
  - Better error messages on wrong login or network problems. Thanks to Antoine Pierlot-Garcin for patches to Mopidy and Pyspotify. (Fixes: #77)
  - Reduce log level for trivial log messages from warning to info. (Fixes: #71)
  - Pause playback on network connection errors. (Fixes: #65)
- Local backend:
  - Fix crash in **`mopidy-scan`** if a track has no artist name. Thanks to Martins Grunskis for test and patch and "octe" for patch.
  - Fix crash in *tag_cache* parsing if a track has no total number of tracks in the album. Thanks to Martins Grunskis for the patch.
- MPD frontend:
  - Add support for "date" queries to both the `find` and `search` commands. This makes media library browsing in ncmpcpp work, though very slow due to all the meta data requests to Spotify.
  - Add support for `play "-1"` when in playing or paused state, which fixes resume and addition of tracks to the current playlist while playing for the MPoD client.
  - Fix bug where `status` returned `song: None`, which caused MPDroid to crash. (Fixes: #69)
  - Gracefully fallback to IPv4 sockets on systems that supports IPv6, but has turned it off. (Fixes: #75)
- GStreamer output:
  - Use `uridecodebin` for playing audio from both Spotify and the local backend. This contributes to support for multiple backends simultaneously.
- Settings:
  - Fix crash on `--list-settings` on clean installation. Thanks to Martins Grunskis for the bug report and patch. (Fixes: #63)
- Packaging:
  - Replace test data symlinks with real files to avoid symlink issues when installing with pip. (Fixes: #68)
- Debugging:
  - Include platform, architecture, Linux distribution, and Python version in the debug log, to ease debugging of issues with attached debug logs.

23.47 v0.3.1 (2011-01-22)

A couple of fixes to the 0.3.0 release is needed to get a smooth installation.

Bug fixes

- The Spotify application key was missing from the Python package.
- Installation of the Python package as a normal user failed because it did not have permissions to install `mopidy.desktop`. The file is now only installed if the installation is executed as the root user.

23.48 v0.3.0 (2011-01-22)

Mopidy 0.3.0 brings a bunch of small changes all over the place, but no large changes. The main features are support for high bitrate audio from Spotify, and MPD password authentication.

Regarding the docs, we've improved the *installation instructions* and done a bit of testing of the available *Android* and *iOS clients* for MPD.

Please note that 0.3.0 requires some updated dependencies, as listed under *Important changes* below. Also, there is a known bug in the Spotify playlist loading, as described below. As the bug will take some time to fix and has a known workaround, we did not want to delay the release while waiting for a fix to this problem.

Warning: Known bug in Spotify playlist loading

There is a known bug in the loading of Spotify playlists. This bug affects both Mopidy 0.2.1 and 0.3.0, given that you use libspotify 0.0.6. To avoid the bug, either use Mopidy 0.2.1 with libspotify 0.0.4, or use either Mopidy version with libspotify 0.0.6 and follow the simple workaround described at #59.

Important changes

- If you use the Spotify backend, you need to upgrade to libspotify 0.0.6 and the latest pyspotify from the Mopidy developers. Follow the instructions at *Installation*.
- If you use the Last.fm frontend, you need to upgrade to pylast 0.5.7. Run `sudo pip install --upgrade pylast` or install Mopidy from APT.

Changes

- Spotify backend:
  - Support high bitrate (320k) audio. Set the new setting `mopidy.settings.SPOTIFY_HIGH_BITRATE` to `True` to switch to high bitrate audio.
  - Rename `mopidy.backends.libspotify` to `mopidy.backends.spotify`. If you have set `mopidy.settings.BACKENDS` explicitly, you may need to update the setting's value.
  - Catch and log error caused by playlist folder boundaries being threated as normal playlists. More permanent fix requires support for checking playlist types in pyspotify (see #62).
  - Fix crash on failed lookup of track by URI. (Fixes: #60)
- Local backend:
  - Add **`mopidy-scan`** command to generate `tag_cache` files without any help from the original MPD server. See *Generating a local library* for instructions on how to use it.
  - Fix support for UTF-8 encoding in tag caches.
- MPD frontend:
  - Add support for password authentication. See `mopidy.settings.MPD_SERVER_PASSWORD` for details on how to use it. (Fixes: #41)

  - Support `setvol 50` without quotes around the argument. Fixes volume control in Droid MPD.
  - Support `seek 1 120` without quotes around the arguments. Fixes seek in Droid MPD.

- Last.fm frontend:
  - Update to use Last.fm's new Scrobbling 2.0 API, as the old Submissions Protocol 1.2.1 is deprecated. (Fixes: #33)
  - Fix crash when track object does not contain all the expected meta data.
  - Fix crash when response from Last.fm cannot be decoded as UTF-8. (Fixes: #37)
  - Fix crash when response from Last.fm contains invalid XML.
  - Fix crash when response from Last.fm has an invalid HTTP status line.
- Mixers:
  - Support use of unicode strings for settings specific to `mopidy.mixers.nad`.
- Settings:
  - Automatically expand the "~" characted to the user's home directory and make the path absolute for settings with names ending in `_PATH` or `_FILE`.
  - Rename the following settings. The settings validator will warn you if you need to change your local settings.
    * `LOCAL_MUSIC_FOLDER` to `mopidy.settings.LOCAL_MUSIC_PATH`
    * `LOCAL_PLAYLIST_FOLDER` to `mopidy.settings.LOCAL_PLAYLIST_PATH`
    * `LOCAL_TAG_CACHE` to `mopidy.settings.LOCAL_TAG_CACHE_FILE`
    * `SPOTIFY_LIB_CACHE` to `mopidy.settings.SPOTIFY_CACHE_PATH`
  - Fix bug which made settings set to `None` or 0 cause a `mopidy.SettingsError` to be raised.
- Packaging and distribution:
  - Setup APT repository and create Debian packages of Mopidy. See *Installation* for instructions for how to install Mopidy, including all dependencies, from APT.
  - Install `mopidy.desktop` file that makes Mopidy available from e.g. Gnome application menus.
- API:
  - Rename and generalize `Playlist._with(**kwargs)` to *`mopidy.models.ImmutableObject.copy()`*.
  - Add `musicbrainz_id` field to *`mopidy.models.Artist`*, *`mopidy.models.Album`*, and *`mopidy.models.Track`*.
  - Prepare for multi-backend support (see #40) by introducing the *provider concept*. Split the backend API into a *backend controller API* (for frontend use) and a *backend provider API* (for backend implementation use), which includes the following changes:
    * Rename `BaseBackend` to `mopidy.backends.base.Backend`.
    * Rename `BaseCurrentPlaylistController` to `mopidy.backends.base.CurrentPlaylistController`.
    * Split `BaseLibraryController` to `mopidy.backends.base.LibraryController` and `mopidy.backends.base.BaseLibraryProvider`.
    * Split `BasePlaybackController` to `mopidy.backends.base.PlaybackController` and `mopidy.backends.base.BasePlaybackProvider`.

    * Split `BaseStoredPlaylistsController` to `mopidy.backends.base.StoredPlaylistsController` and `mopidy.backends.base.BaseStoredPlaylistsProvider`.

  - Move `BaseMixer` to `mopidy.mixers.base.BaseMixer`.
  - Add docs for the current non-stable output API, `mopidy.outputs.base.BaseOutput`.

23.49 v0.2.1 (2011-01-07)

This is a maintenance release without any new features.

Bug fixes

- Fix crash in `mopidy.frontends.lastfm` which occurred at playback if either `pylast` was not installed or the Last.fm scrobbling was not correctly configured. The scrobbling thread now shuts properly down at failure.

23.50 v0.2.0 (2010-10-24)

In Mopidy 0.2.0 we've added a Last.fm scrobbling support, which means that Mopidy now can submit meta data about the tracks you play to your Last.fm profile. See `mopidy.frontends.lastfm` for details on new dependencies and settings. If you use Mopidy's Last.fm support, please join the Mopidy group at Last.fm.

With the exception of the work on the Last.fm scrobbler, there has been a couple of quiet months in the Mopidy camp. About the only thing going on, has been stabilization work and bug fixing. All bugs reported on GitHub, plus some, have been fixed in 0.2.0. Thus, we hope this will be a great release!

We've worked a bit on OS X support, but not all issues are completely solved yet. #25 is the one that is currently blocking OS X support. Any help solving it will be greatly appreciated!

Finally, please *update your pyspotify installation* when upgrading to Mopidy 0.2.0. The latest pyspotify got a fix for the segmentation fault that occurred when playing music and searching at the same time, thanks to Valentin David.

Important changes

- Added a Last.fm scrobbler. See `mopidy.frontends.lastfm` for details.

Changes

- Logging and command line options:
  - Simplify the default log format, `mopidy.settings.CONSOLE_LOG_FORMAT`. From a user's point of view: Less noise, more information.
  - Rename the `--dump` command line option to `--save-debug-log`.
  - Rename setting `mopidy.settings.DUMP_LOG_FORMAT` to `mopidy.settings.DEBUG_LOG_FORMAT` and use it for `--verbose` too.
  - Rename setting `mopidy.settings.DUMP_LOG_FILENAME` to `mopidy.settings.DEBUG_LOG_FILENAME`.
- MPD frontend:
  - MPD command `list` now supports queries by artist, album name, and date, as used by e.g. the Ario client. (Fixes: #20)
  - MPD command `add ""` and `addid ""` now behaves as expected. (Fixes #16)

  - MPD command `playid "-1"` now correctly resumes playback if paused.
- Random mode:
  - Fix wrong behavior on end of track and next after random mode has been used. (Fixes: #18)
  - Fix infinite recursion loop crash on playback of non-playable tracks when in random mode. (Fixes #17)
  - Fix assertion error that happened if one removed tracks from the current playlist, while in random mode. (Fixes #22)
- Switched from using subprocesses to threads. (Fixes: #14)
- `mopidy.outputs.gstreamer`: Set `caps` on the `appsrc` bin before use. This makes sound output work with GStreamer >= 0.10.29, which includes the versions used in Ubuntu 10.10 and on OS X if using Homebrew. (Fixes: #21, #24, contributes to #14)
- Improved handling of uncaught exceptions in threads. The entire process should now exit immediately.

23.51 v0.1.0 (2010-08-23)

After three weeks of long nights and sprints we're finally pleased enough with the state of Mopidy to remove the alpha label, and do a regular release.

Mopidy 0.1.0 got important improvements in search functionality, working track position seeking, no known stability issues, and greatly improved MPD client support. There are lots of changes since 0.1.0a3, and we urge you to at least read the *important changes* below.

This release does not support OS X. We're sorry about that, and are working on fixing the OS X issues for a future release. You can track the progress at #14.

Important changes

- License changed from GPLv2 to Apache License, version 2.0.
- GStreamer is now a required dependency. See our *GStreamer installation docs*.
- `mopidy.backends.libspotify` is now the default backend. `mopidy.backends.despotify` is no longer available. This means that you need to install the *dependencies for libspotify*.
- If you used `mopidy.backends.libspotify` previously, pyspotify must be updated when updating to this release, to get working seek functionality.
- `mopidy.settings.SERVER_HOSTNAME` and `mopidy.settings.SERVER_PORT` has been renamed to `mopidy.settings.MPD_SERVER_HOSTNAME` and `mopidy.settings.MPD_SERVER_PORT` to allow for multiple frontends in the future.

Changes

- Exit early if not Python >= 2.6, < 3.
- Validate settings at startup and print useful error messages if the settings has not been updated or anything is misspelled.
- Add command line option `--list-settings` to print the currently active settings.
- Include Sphinx scripts for building docs, pylintrc, tests and test data in the packages created by `setup.py` for i.e. PyPI.
- MPD frontend:
  - Search improvements, including support for multi-word search.
  - Fixed `play "-1"` and `playid "-1"` behaviour when playlist is empty or when a current track is set.

- Support `plchanges "-1"` to work better with MPDroid.
- Support `pause` without arguments to work better with MPDroid.
- Support `plchanges`, `play`, `consume`, `random`, `repeat`, and `single` without quotes to work better with BitMPC.
- Fixed deletion of the currently playing track from the current playlist, which crashed several clients.
- Implement `seek` and `seekid`.
- Fix `playlistfind` output so the correct song is played when playing songs directly from search results in GMPC.
- Fix `load` so that one can append a playlist to the current playlist, and make it return the correct error message if the playlist is not found.
- Support for single track repeat added. (Fixes: #4)
- Relocate from *`mopidy.mpd`* to `mopidy.frontends.mpd`.
- Split gigantic protocol implementation into eleven modules.
- Rename `mopidy.frontends.mpd.{serializer => translator}` to match naming in backends.
- Remove setting `mopidy.settings.SERVER` and `mopidy.settings.FRONTEND` in favour of the new `mopidy.settings.FRONTENDS`.
- Run MPD server in its own process.

• Backends:

- Rename `mopidy.backends.gstreamer` to `mopidy.backends.local`.
- Remove `mopidy.backends.despotify`, as Despotify is little maintained and the Libspotify backend is working much better. (Fixes: #9, #10, #13)
- A Spotify application key is now bundled with the source. `mopidy.settings.SPOTIFY_LIB_APPKEY` is thus removed.
- If failing to play a track, playback will skip to the next track.
- Both `mopidy.backends.libspotify` and `mopidy.backends.local` have been rewritten to use the new common GStreamer audio output module, `mopidy.outputs.gstreamer`.

• Mixers:

- Added new `mopidy.mixers.gstreamer_software.GStreamerSoftwareMixer` which now is the default mixer on all platforms.
- New setting `mopidy.settings.MIXER_MAX_VOLUME` for capping the maximum output volume.

• Backend API:

- Relocate from `mopidy.backends` to `mopidy.backends.base`.
- The `id` field of *`mopidy.models.Track`* has been removed, as it is no longer needed after the CPID refactoring.
- `mopidy.backends.base.BaseBackend()` now accepts an `output_queue` which it can use to send messages (i.e. audio data) to the output process.
- `mopidy.backends.base.BaseLibraryController.find_exact()` now accepts keyword arguments of the form `find_exact(artist=['foo'], album=['bar'])`.

- `mopidy.backends.base.BaseLibraryController.search()` now accepts keyword arguments of the form `search(artist=['foo', 'fighters'], album=['bar', 'grooves'])`.
- `mopidy.backends.base.BaseCurrentPlaylistController.append()` replaces `mopidy.backends.base.BaseCurrentPlaylistController.load()`. Use `mopidy.backends.base.BaseCurrentPlaylistController.clear()` if you want to clear the current playlist.
- The following fields in `mopidy.backends.base.BasePlaybackController` has been renamed to reflect their relation to methods called on the controller:
  * `next_track` to `track_at_next`
  * `next_cp_track` to `cp_track_at_next`
  * `previous_track` to `track_at_previous`
  * `previous_cp_track` to `cp_track_at_previous`
- `mopidy.backends.base.BasePlaybackController.track_at_eot` and `mopidy.backends.base.BasePlaybackController.cp_track_at_eot` has been added to better handle the difference between the user pressing next and the current track ending.
- Rename `mopidy.backends.base.BasePlaybackController.new_playlist_loaded_callback()` to `mopidy.backends.base.BasePlaybackController.on_current_playlist_change()`.
- Rename `mopidy.backends.base.BasePlaybackController.end_of_track_callback()` to `mopidy.backends.base.BasePlaybackController.on_end_of_track()`.
- Remove `mopidy.backends.base.BaseStoredPlaylistsController.search()` since it was barely used, untested, and we got no use case for non-exact search in stored playlists yet. Use `mopidy.backends.base.BaseStoredPlaylistsController.get()` instead.

23.52 v0.1.0a3 (2010-08-03)

In the last two months, Mopidy's MPD frontend has gotten lots of stability fixes and error handling improvements, proper support for having the same track multiple times in a playlist, and support for IPv6. We have also fixed the choppy playback on the libspotify backend. For the road ahead of us, we got an updated release roadmap with our goals for the 0.1 to 0.3 releases.

Enjoy the best alpha relase of Mopidy ever :-)

Changes

- MPD frontend:
  - Support IPv6.
  - `addid` responds properly on errors instead of crashing.
  - `commands` support, which makes RelaXXPlayer work with Mopidy. (Fixes: #6)
  - Does no longer crash on invalid data, i.e. non-UTF-8 data.
  - `ACK` error messages are now MPD-compliant, which should make clients handle errors from Mopidy better.
  - Requests to existing commands with wrong arguments are no longer reported as unknown commands.

  - `command_list_end` before `command_list_start` now returns unknown command error instead of crashing.
  - `list` accepts field argument without quotes and capitalized, to work with GMPC and ncmpc.
  - `noidle` command now returns `OK` instead of an error. Should make some clients work a bit better.
  - Having multiple identical tracks in a playlist is now working properly. (CPID refactoring)
- Despotify backend:
  - Catch and log `spytify.SpytifyError`. (Fixes: #11)
- Libspotify backend:
  - Fix choppy playback using the Libspotify backend by using blocking ALSA mode. (Fixes: #7)
- Backend API:
  - A new data structure called `cp_track` is now used in the current playlist controller and the playback controller. A `cp_track` is a two-tuple of (CPID integer, *`mopidy.models.Track`*), identifying an instance of a track uniquely within the current playlist.
  - `mopidy.backends.BaseCurrentPlaylistController.load()` now accepts lists of *`mopidy.models.Track`* instead of *`mopidy.models.Playlist`*, as none of the other fields on the `Playlist` model was in use.
  - `mopidy.backends.BaseCurrentPlaylistController.add()` now returns the `cp_track` added to the current playlist.
  - `mopidy.backends.BaseCurrentPlaylistController.remove()` now takes criterias, just like `mopidy.backends.BaseCurrentPlaylistController.get()`.
  - `mopidy.backends.BaseCurrentPlaylistController.get()` now returns a `cp_track`.
  - `mopidy.backends.BaseCurrentPlaylistController.tracks` is now read-only. Use the methods to change its contents.
  - `mopidy.backends.BaseCurrentPlaylistController.cp_tracks` is a read-only list of `cp_track`. Use the methods to change its contents.
  - `mopidy.backends.BasePlaybackController.current_track` is now just for convenience and read-only. To set the current track, assign a `cp_track` to `mopidy.backends.BasePlaybackController.current_cp_track`.
  - `mopidy.backends.BasePlaybackController.current_cpid` is the read-only CPID of the current track.
  - `mopidy.backends.BasePlaybackController.next_cp_track` is the next `cp_track` in the playlist.
  - `mopidy.backends.BasePlaybackController.previous_cp_track` is the previous `cp_track` in the playlist.
  - `mopidy.backends.BasePlaybackController.play()` now takes a `cp_track`.

23.53 v0.1.0a2 (2010-06-02)

It has been a rather slow month for Mopidy, but we would like to keep up with the established pace of at least a release per month.

Changes

- Improvements to MPD protocol handling, making Mopidy work much better with a group of clients, including ncmpc, MPoD, and Theremin.
- New command line flag `--dump` for dumping debug log to `dump.log` in the current directory.
- New setting `mopidy.settings.MIXER_ALSA_CONTROL` for forcing what ALSA control `mopidy.mixers.alsa.AlsaMixer` should use.

23.54 v0.1.0a1 (2010-05-04)

Since the previous release Mopidy has seen about 300 commits, more than 200 new tests, a libspotify release, and major feature additions to Spotify. The new releases from Spotify have lead to updates to our dependencies, and also to new bugs in Mopidy. Thus, this is primarily a bugfix release, even though the not yet finished work on a GStreamer backend have been merged.

All users are recommended to upgrade to 0.1.0a1, and should at the same time ensure that they have the latest versions of our dependencies: Despotify r508 if you are using DespotifyBackend, and pyspotify 1.1 with libspotify 0.0.4 if you are using LibspotifyBackend.

As always, report problems at our IRC channel or our issue tracker. Thanks!

Changes

- Backend API changes:
  - Removed `backend.playback.volume` wrapper. Use `backend.mixer.volume` directly.
  - Renamed `backend.playback.playlist_position` to `current_playlist_position` to match naming of `current_track`.
  - Replaced `get_by_id()` with a more flexible `get(**criteria)`.
- Merged the `gstreamer` branch from Thomas Adamcik:
  - More than 200 new tests, and thus several bug fixes to existing code.
  - Several new generic features, like shuffle, consume, and playlist repeat. (Fixes: #3)
  - **[Work in Progress]** A new backend for playing music from a local music archive using the GStreamer library.
- Made `mopidy.mixers.alsa.AlsaMixer` work on machines without a mixer named "Master".
- Make `mopidy.backends.DespotifyBackend` ignore local files in playlists (feature added in Spotify 0.4.3). Reported by Richard Haugen Olsen.
- And much more.

23.55 v0.1.0a0 (2010-03-27)

"*Release early. Release often. Listen to your customers.*" wrote Eric S. Raymond in *The Cathedral and the Bazaar*.

Three months of development should be more than enough. We have more to do, but Mopidy is working and usable. 0.1.0a0 is an alpha release, which basicly means we will still change APIs, add features, etc. before the final 0.1.0 release. But the software is usable as is, so we release it. Please give it a try and give us feedback, either at our IRC channel or through the issue tracker. Thanks!

Changes

- Initial version. No changelog available.

CHAPTER 24

Versioning

Mopidy follows Semantic Versioning. In summary this means that our version numbers have three parts, MAJOR.MINOR.PATCH, which change according to the following rules:

- When we *make incompatible API changes*, we increase the MAJOR number.
- When we *add features* in a backwards-compatible manner, we increase the MINOR number.
- When we *fix bugs* in a backwards-compatible manner, we increase the PATCH number.

The promise is that if you make a Mopidy extension for Mopidy 1.0, it should work unchanged with any Mopidy 1.x release, but probably not with 2.0. When a new major version is released, you must review the incompatible changes and update your extension accordingly.

24.1 Release schedule

We intend to have about one feature release every month in periods of active development. The features added is a mix of what we feel is most important/requested of the missing features, and features we develop just because we find them fun to make, even though they may be useful for very few users or for a limited use case.

Bugfix releases will be released whenever we discover bugs that are too serious to wait for the next feature release. We will only release bugfix releases for the last feature release. E.g. when 1.2.0 is released, we will no longer provide bugfix releases for the 1.1.x series. In other words, there will be just a single supported release at any point in time. This is to not spread our limited resources too thin.

CHAPTER 25

Contributing

If you want to contribute to Mopidy, here are some tips to get you started.

25.1 Asking questions

Please get in touch with us in one of these ways when requesting help with Mopidy and its extensions:

- Our discussion forum: discuss.mopidy.com. Just sign in and fire away.
- Our IRC channel: #mopidy on irc.freenode.net, with public searchable logs. Be prepared to hang around for a while, as we're not always around to answer straight away.

Before asking for help, it might be worth your time to read the *Troubleshooting* page, both so you might find a solution to your problem but also to be able to provide useful details when asking for help.

25.2 Helping users

If you want to contribute to Mopidy, a great place to start is by helping other users on IRC and in the discussion forum. This is a contribution we value highly. As more people help with user support, new users get faster and better help. For your own benefit, you'll quickly learn what users find confusing, difficult or lacking, giving you some ideas for where you may contribute improvements, either to code or documentation. Lastly, this may also free up time for other contributors to spend more time on fixing bugs or implementing new features.

25.3 Issue guidelines

1. If you need help, see *Asking questions* above. The GitHub issue tracker is not a support forum.
2. If you are not sure if what you're experiencing is a bug or not, post in the discussion forum first to verify that it's a bug.

3. If you are sure that you've found a bug or have a feature request, check if there's already an issue in the issue tracker. If there is, see if there is anything you can add to help reproduce or fix the issue.
4. If there is no exising issue matching your bug or feature request, create a new issue. Please include as much relevant information as possible. If it's a bug, including how to reproduce the bug and any relevant logs or error messages.

25.4 Pull request guidelines

1. Before spending any time on making a pull request:
   - If it's a bug, *file an issue*.
   - If it's an enhancement, discuss it with other Mopidy developers first, either in a GitHub issue, on the discussion forum, or on IRC. Making sure your ideas and solutions are aligned with other contributors greatly increases the odds of your pull request being quickly accepted.
2. Create a new branch, based on the `develop` branch, for every feature or bug fix. Keep branches small and on topic, as that makes them far easier to review. We often use the following naming convention for branches:
   - Features get the prefix `feature/`, e.g. `feature/track-last-modified-as-ms`.
   - Bug fixes get the prefix `fix/`, e.g. `fix/902-consume-track-on-next`.
   - Improvements to the documentation get the prefix `docs/`, e.g. `docs/add-ext-mopidy-spotify-tunigo`.
3. Follow the *code style*, especially make sure the `flake8` linter does not complain about anything. Travis CI will check that your pull request is "flake8 clean". See *Style checking and linting*.
4. Include tests for any new feature or substantial bug fix. See *Running tests*.
5. Include documentation for any new feature. See *Writing documentation*.
6. Feel free to include a changelog entry in your pull request. The changelog is in `docs/changelog.rst`.
7. Write good commit messages.
   - Follow the template "topic: description" for the first line of the commit message, e.g. "mpd: Switch list command to using list_distinct". See the commit history for inspiration.
   - Use the rest of the commit message to explain anything you feel isn't obvious. It's better to have the details here than in the pull request description, since the commit message will live forever.
   - Write in the imperative, present tense: "add" not "added".

   For more inspiration, feel free to read these blog posts:
   - Writing Git commit messages
   - A Note About Git Commit Messages
   - On commit messages
8. Send a pull request to the `develop` branch. See the GitHub pull request docs for help.

Note: If you are contributing a bug fix for a specific minor version of Mopidy you should create the branch based on `release-x.y` instead of `develop`. When the release is done the changes will be merged back into `develop` automatically as part of the normal release process. See *Creating releases*.

CHAPTER 26

Development environment

This page describes a common development setup for working with Mopidy and Mopidy extensions. Of course, there may be other ways that work better for you and the tools you use, but here's one recommended way to do it.

26.1 Initial setup

The following steps help you get a good initial setup. They build on each other to some degree, so if you're not very familiar with Python development it might be wise to proceed in the order laid out here.

26.1.1 Install Mopidy the regular way

Install Mopidy the regular way. Mopidy has some non-Python dependencies which may be tricky to install. Thus we recommend to always start with a full regular Mopidy install, as described in *Installation*. That is, if you're running e.g. Debian, start with installing Mopidy from Debian packages.

26.1.2 Make a development workspace

Make a directory to be used as a workspace for all your Mopidy development:

```
mkdir ~/mopidy-dev
```

It will contain all the Git repositories you'll check out when working on Mopidy and extensions.

26.1.3 Make a virtualenv

Make a Python virtualenv for Mopidy development. The virtualenv will wall off Mopidy and its dependencies from the rest of your system. All development and installation of Python dependencies, versions of Mopidy, and extensions are done inside the virtualenv. This way your regular Mopidy install, which you set up in the first step, is unaffected by your hacking and will always be working.

Most of us use the virtualenvwrapper to ease working with virtualenvs, so that's what we'll be using for the examples here. First, install and setup virtualenvwrapper as described in their docs.

To create a virtualenv named `mopidy` which uses Python 2.7, allows access to system-wide packages like GStreamer, and uses the Mopidy workspace directory as the "project path", run:

```
mkvirtualenv -a ~/mopidy-dev --python `which python2.7` \
  --system-site-packages mopidy
```

Now, each time you open a terminal and want to activate the `mopidy` virtualenv, run:

```
workon mopidy
```

This will both activate the `mopidy` virtualenv, and change the current working directory to `~/mopidy-dev`.

26.1.4 Clone the repo from GitHub

Once inside the virtualenv, it's time to clone the `mopidy/mopidy` Git repo from GitHub:

```
git clone https://github.com/mopidy/mopidy.git
```

When you've cloned the `mopidy` Git repo, `cd` into it:

```
cd ~/mopidy-dev/mopidy/
```

With a fresh clone of the Git repo, you should start out on the `develop` branch. This is where all features for the next feature release land. To confirm that you're on the right branch, run:

```
git branch
```

26.1.5 Install development tools

We use a number of Python development tools. The `dev-requirements.txt` file has comments describing what we use each dependency for, so we might just as well include the file verbatim here:

```
# Automate tasks
invoke

# Build documentation
sphinx

# Check code style, errors, etc
flake8
flake8-import-order

# Mock dependencies in tests
mock
responses

# Test runners
pytest
pytest-capturelog
pytest-cov
pytest-xdist
tox

# Check that MANIFEST.in matches Git repo contents before making a release
check-manifest

# To make wheel packages
```

(continues on next page)

(continued from previous page)

```
wheel

# Securely upload packages to PyPI
twine
```

Install them all into the active virtualenv by running pip:

```
pip install --upgrade -r dev-requirements.txt
```

To upgrade the tools in the future, just rerun the exact same command.

26.1.6 Install Mopidy from the Git repo

Next up, we'll want to run Mopidy from the Git repo. There's two reasons for this: first of all, it lets you easily change the source code, restart Mopidy, and see the change take effect. Second, it's a convenient way to keep at the bleeding edge, testing the latest developments in Mopidy itself or test some extension against the latest Mopidy changes.

Assuming you're still inside the Git repo, use pip to install Mopidy from the Git repo in an "editable" form:

```
pip install --editable .
```

This will not copy the source code into the virtualenv's `site-packages` directory, but instead create a link there pointing to the Git repo. Using `cdsitepackages` from virtualenvwrapper, we can quickly show that the installed `Mopidy.egg-link` file points back to the Git repo:

```
$ cdsitepackages
$ cat Mopidy.egg-link
/home/user/mopidy-dev/mopidy
.%
$
```

It will also create a `mopidy` executable inside the virtualenv that will always run the latest code from the Git repo. Using another virtualenvwrapper command, `cdvirtualenv`, we can show that too:

```
$ cdvirtualenv
$ cat bin/mopidy
...
```

The executable should contain something like this, using `pkg_resources` to look up Mopidy's "console script" entry point:

```
#!/home/user/virtualenvs/mopidy/bin/python2
# EASY-INSTALL-ENTRY-SCRIPT: 'Mopidy==0.19.5','console_scripts','mopidy'
__requires__ = 'Mopidy==0.19.5'
import sys
from pkg_resources import load_entry_point

if __name__ == '__main__':
    sys.exit(
        load_entry_point('Mopidy==0.19.5', 'console_scripts', 'mopidy')()
    )
```

Note: It still works to run `python mopidy` directly on the `~/mopidy-dev/mopidy/mopidy/` Python package directory, but if you don't run the `pip install` command above, the extensions bundled with Mopidy will not be registered with pkg_resources, making Mopidy quite useless.

Third, the pip install command will register the bundled Mopidy extensions so that Mopidy may find them through pkg_resources. The result of this can be seen in the Git repo, in a new directory called Mopidy.egg-info, which is ignored by Git. The Mopidy.egg-info/entry_points.txt file is of special interest as it shows both how the above executable and the bundled extensions are connected to the Mopidy source code:

```
[console_scripts]
mopidy = mopidy.__main__:main

[mopidy.ext]
http = mopidy.http:Extension
local = mopidy.local:Extension
mpd = mopidy.mpd:Extension
softwaremixer = mopidy.softwaremixer:Extension
stream = mopidy.stream:Extension
```

Warning: It's not uncommon to clean up in the Git repo now and then, e.g. by running git clean.

If you do this, then the Mopidy.egg-info directory will be removed, and pkg_resources will no longer know how to locate the "console script" entry point or the bundled Mopidy extensions.

The fix is simply to run the install command again:

```
pip install --editable .
```

Finally, we can go back to the workspace, again using a virtualenvwrapper tool:

```
cdproject
```

26.2 Running Mopidy from Git

As long as the virtualenv is activated, you can start Mopidy from any directory. Simply run:

```
mopidy
```

To stop it again, press Ctrl+C.

Every time you change code in Mopidy or an extension and want to see it live, you must restart Mopidy.

If you want to iterate quickly while developing, it may sound a bit tedious to restart Mopidy for every minor change. Then it's useful to have tests to exercise your code...

26.3 Running tests

Mopidy has quite good test coverage, and we would like all new code going into Mopidy to come with tests.

26.3.1 Test it all

You need to know at least one command; the one that runs all the tests:

```
tox
```

This will run exactly the same tests as Travis CI runs for all our branches and pull requests. If this command turns green, you can be quite confident that your pull request will get the green flag from Travis as well, which is a requirement for it to be merged.

As this is the ultimate test command, it's also the one taking the most time to run; up to a minute, depending on your system. But, if you have patience, this is all you need to know. Always run this command before pushing your changes to GitHub.

If you take a look at the tox config file, `tox.ini`, you'll see that tox runs tests in multiple environments, including a `flake8` environment that lints the source code for issues and a `docs` environment that tests that the documentation can be built. You can also limit tox to just test specific environments using the `-e` option, e.g. to run just unit tests:

```
tox -e py27
```

To learn more, see the tox documentation .

26.3.2 Running unit tests

Under the hood, `tox -e py27` will use pytest as the test runner. We can also use it directly to run all tests:

```
py.test
```

py.test has lots of possibilities, so you'll have to dive into their docs and plugins to get full benefit from it. To get you interested, here are some examples.

We can limit to just tests in a single directory to save time:

```
py.test tests/http/
```

With the help of the pytest-xdist plugin, we can run tests with four Python processes in parallel, which usually cuts the test time in half or more:

```
py.test -n 4
```

Another useful feature from pytest-xdist, is the possiblity to stop on the first test failure, watch the file system for changes, and then rerun the tests. This makes for a very quick code-test cycle:

```
py.test -f     # or --looponfail
```

With the help of the pytest-cov plugin, we can get a report on what parts of the given module, `mopidy` in this example, are covered by the test suite:

```
py.test --cov=mopidy --cov-report=term-missing
```

Note: Up to date test coverage statistics can also be viewed online at coveralls.io.

If we want to speed up the test suite, we can even get a list of the ten slowest tests:

```
py.test --durations=10
```

By now, you should be convinced that running py.test directly during development can be very useful.

26.3.3 Continuous integration

Mopidy uses the free service Travis CI for automatically running the test suite when code is pushed to GitHub. This works both for the main Mopidy repo, but also for any forks. This way, any contributions to Mopidy through GitHub will automatically be tested by Travis CI, and the build status will be visible in the GitHub pull request interface, making it easier to evaluate the quality of pull requests.

For each successful build, Travis submits code coverage data to coveralls.io. If you're out of work, coveralls might help you find areas in the code which could need better test coverage.

26.3.4 Style checking and linting

We're quite pedantic about *Code style* and try hard to keep the Mopidy code base a very clean and nice place to work in.

Luckily, you can get very far by using the flake8 linter to check your code for issues before submitting a pull request. Mopidy passes all of flake8's checks, with only a very few exceptions configured in `setup.cfg`. You can either run the `flake8` tox environment, like Travis CI will do on your pull request:

```
tox -e flake8
```

Or you can run flake8 directly:

```
flake8
```

If successful, the command will not print anything at all.

Note: In some rare cases it doesn't make sense to listen to flake8's warnings. In those cases, ignore the check by appending `# noqa: <warning code>` to the source line that triggers the warning. The `# noqa` part will make flake8 skip all checks on the line, while the warning code will help other developers lookup what you are ignoring.

26.4 Writing documentation

To write documentation, we use Sphinx. See their site for lots of documentation on how to use Sphinx.

Note: To generate a few graphs which are part of the documentation, you need some additional dependencies. You can install them from APT with:

```
sudo apt-get install python-pygraphviz graphviz
```

To build the documentation, go into the `docs/` directory:

```
cd ~/mopidy-dev/mopidy/docs/
```

Then, to see all available build targets, run:

```
make
```

To generate an HTML version of the documentation, run:

```
make html
```

The generated HTML will be available at `_build/html/index.html`. To open it in a browser you can run either of the following commands, depending on your OS:

```
xdg-open _build/html/index.html     # Linux
open _build/html/index.html         # OS X
```

The documentation at https://docs.mopidy.com/ is hosted by Read the Docs, which automatically updates the documentation when a change is pushed to the `mopidy/mopidy` repo at GitHub.

26.5 Working on extensions

Much of the above also applies to Mopidy extensions, though they're often a bit simpler. They don't have documentation sites and their test suites are either small and fast, or sadly missing entirely. Most of them use tox and flake8, and py.test can be used to run their test suites.

26.5.1 Installing extensions

As always, the `mopidy` virtualenv should be active when working on extensions:

```
workon mopidy
```

Just like with non-development Mopidy installations, you can install extensions using pip:

```
pip install Mopidy-Scrobbler
```

Installing an extension from its Git repo works the same way as with Mopidy itself. First, go to the Mopidy workspace:

```
cdproject     # or cd ~/mopidy-dev/
```

Clone the desired Mopidy extension:

```
git clone https://github.com/mopidy/mopidy-spotify.git
```

Change to the newly created extension directory:

```
cd mopidy-spotify/
```

Then, install the extension in “editable” mode, so that it can be imported from anywhere inside the virtualenv and the extension is registered and discoverable through `pkg_resources`:

```
pip install --editable .
```

Every extension will have a `README.rst` file. It may contain information about extra dependencies required, development process, etc. Extensions usually have a changelog in the readme file.

26.5.2 Upgrading extensions

Extensions often have a much quicker life cycle than Mopidy itself, often with daily releases in periods of active development. To find outdated extensions in your virtualenv, you can run:

```
pip search mopidy
```

This will list all available Mopidy extensions and compare the installed versions with the latest available ones.

To upgrade an extension installed with pip, simply use pip:

```
pip install --upgrade Mopidy-Scrobbler
```

To upgrade an extension installed from a Git repo, it’s usually enough to pull the new changes in:

```
cd ~/mopidy-dev/mopidy-spotify/
git pull
```

Of course, if you have local modifications, you’ll need to stash these away on a branch or similar first.

Depending on the changes to the extension, it may be necessary to update the metadata about the extension package by installing it in “editable” mode again:

```
pip install --editable .
```

26.6 Contribution workflow

Before you being, make sure you’ve read the *Contributing* page and the guidelines there. This section will focus more on the practical workflow.

For the examples, we’re making a change to Mopidy. Approximately the same workflow should work for most Mopidy extensions too.

26.6.1 Setting up Git remotes

Assuming we already have a local Git clone of the upstream Git repo in `~/mopidy-dev/mopidy/`, we can run `git remote -v` to list the configured remotes of the repo:

```
$ git remote -v
origin  https://github.com/mopidy/mopidy.git (fetch)
origin  https://github.com/mopidy/mopidy.git (push)
```

For clarity, we can rename the `origin` remote to `upstream`:

```
$ git remote rename origin upstream
$ git remote -v
upstream        https://github.com/mopidy/mopidy.git (fetch)
upstream        https://github.com/mopidy/mopidy.git (push)
```

If you haven't already, fork the repository to your own GitHub account.

Then, add the new fork as a remote to your local clone:

```
git remote add myuser git@github.com:myuser/mopidy.git
```

The end result is that you have both the upstream repo and your own fork as remotes:

```
$ git remote -v
myuser  git@github.com:myuser/mopidy.git (fetch)
myuser  git@github.com:myuser/mopidy.git (push)
upstream        https://github.com/mopidy/mopidy.git (fetch)
upstream        https://github.com/mopidy/mopidy.git (push)
```

26.6.2 Creating a branch

Fetch the latest data from all remotes without affecting your working directory:

```
git remote update
```

Now, we are ready to create and checkout a new branch off of the upstream `develop` branch for our work:

```
git checkout -b fix/666-crash-on-foo upstream/develop
```

Do the work, while remembering to adhere to code style, test the changes, make necessary updates to the documentation, and making small commits with good commit messages. All as described in *Contributing* and elsewhere in the *Development environment* guide.

26.6.3 Creating a pull request

When everything is done and committed, push the branch to your fork on GitHub:

```
git push myuser fix/666-crash-on-foo
```

Go to the repository on GitHub where you want the change merged, in this case https://github.com/mopidy/mopidy, and create a pull request.

26.6.4 Updating a pull request

When the pull request is created, Travis CI will run all tests on it. If something fails, you'll get notified by email. You might as well just fix the issues right away, as we won't merge a pull request without a green Travis build. See *Running tests* on how to run the same tests locally as Travis CI runs on your pull request.

When you've fixed the issues, you can update the pull request simply by pushing more commits to the same branch in your fork:

```
git push myuser fix/666-crash-on-foo
```

Likewise, when you get review comments from other developers on your pull request, you're expected to create additional commits which addresses the comments. Push them to your branch so that the pull request is updated.

Note: Setup the remote as the default push target for your branch:

```
git branch --set-upstream-to myuser/fix/666-crash-on-foo
```

Then you can push more commits without specifying the remote:

```
git push
```

CHAPTER 27

Release procedures

Here we try to keep an up to date record of how Mopidy releases are made. This documentation serves both as a checklist, to reduce the project's dependency on key individuals, and as a stepping stone to more automation.

27.1 Creating releases

1. Update changelog and commit it.
2. Bump the version number in `mopidy/__init__.py`. Remember to update the test case in `tests/test_version.py`.
3. Merge the release branch (`develop` in the example) into master:

```
git checkout master
git merge --no-ff -m "Release v0.16.0" develop
```

4. Install/upgrade tools used for packaging:

```
pip install -U twine wheel
```

5. Build package and test it manually in a new virtualenv. The following assumes the use of virtualenvwrapper:

```
python setup.py sdist bdist_wheel

mktmpenv
pip install path/to/dist/Mopidy-0.16.0.tar.gz
toggleglobalsitepackages
# do manual test
deactivate

mktmpenv
pip install path/to/dist/Mopidy-0.16.0-py27-none-any.whl
toggleglobalsitepackages
```

(continues on next page)

(continued from previous page)

```
# do manual test
deactivate
```

6. Tag the release:

```
git tag -a -m "Release v0.16.0" v0.16.0
```

7. Push to GitHub:

```
git push --follow-tags
```

8. Upload the previously built and tested sdist and bdist_wheel packages to PyPI:

```
twine upload dist/Mopidy-0.16.0*
```

9. Merge `master` back into `develop` and push the branch to GitHub.
10. Make sure the new tag is built by Read the Docs, and that the `latest` version shows the newly released version.
11. Spread the word through the topic on #mopidy on IRC, @mopidy on Twitter, and on the mailing list.
12. Update the Debian package.

27.2 Updating Debian packages

This howto is not intended to learn you all the details, just to give someone already familiar with Debian packaging an overview of how Mopidy's Debian packages is maintained.

1. Install the basic packaging tools:

```
sudo apt-get install build-essential git-buildpackage
```

2. Create a Wheezy pbuilder env if running on Ubuntu and this the first time. See #561 for details about why this is needed:

```
DIST=wheezy sudo git-pbuilder update --mirror=http://mirror.rackspace.com/debian/␣
↪--debootstrapopts --keyring=/usr/share/keyrings/debian-archive-keyring.gpg
```

3. Check out the `debian` branch of the repo:

```
git checkout -t origin/debian
git pull
```

4. Merge the latest release tag into the `debian` branch:

```
git merge v0.16.0
```

5. Update the `debian/changelog` with a "New upstream release" entry:

```
dch -v 0.16.0-0mopidy1
git add debian/changelog
git commit -m "debian: New upstream release"
```

6. Check if any dependencies in `debian/control` or similar needs updating.
7. Install any Build-Deps listed in `debian/control`.

8. Build the package and fix any issues repeatedly until the build succeeds and the Lintian check at the end of the build is satisfactory:

   ```
   git buildpackage -uc -us
   ```

   If you are using the pbuilder make sure this command is:

   ```
   sudo git buildpackage -uc -us --git-ignore-new --git-pbuilder --git-dist=wheezy --
   ↪git-no-pbuilder-autoconf
   ```

9. Install and test newly built package:

   ```
   sudo debi
   ```

   Again for pbuilder use:

   ```
   sudo debi --debs-dir /var/cache/pbuilder/result/
   ```

10. If everything is OK, build the package a final time to tag the package version:

    ```
    git buildpackage -uc -us --git-tag
    ```

    Pbuilder:

    ```
    sudo git buildpackage -uc -us --git-ignore-new --git-pbuilder --git-dist=wheezy --
    ↪git-no-pbuilder-autoconf --git-tag
    ```

11. Push the changes you've done to the `debian` branch and the new tag:

    ```
    git push
    git push --tags
    ```

12. If you're building for multiple architectures, checkout the `debian` branch on the other builders and run:

    ```
    git buildpackage -uc -us
    ```

    Modify as above to use the pbuilder as needed.

13. Copy files to the APT server. Make sure to select the correct part of the repo, e.g. main, contrib, or non-free:

    ```
    scp ../mopidy*_0.16* bonobo.mopidy.com:/srv/apt.mopidy.com/app/incoming/stable/
    ↪main
    ```

14. Update the APT repo:

    ```
    ssh bonobo.mopidy.com
    /srv/apt.mopidy.com/app/update.sh
    ```

15. Test installation from apt.mopidy.com:

    ```
    sudo apt-get update
    sudo apt-get dist-upgrade
    ```

CHAPTER 28

Code style

- Always import unicode_literals and use unicode literals for everything except where you're explicitly working with bytes, which are marked with the b prefix.

  Do this:

```
from __future__ import unicode_literals

foo = 'I am a unicode string, which is a sane default'
bar = b'I am a bytestring'
```

  Not this:

```
foo = u'I am a unicode string'
bar = 'I am a bytestring, but was it intentional?'
```

- Follow **PEP 8** unless otherwise noted. flake8 should be used to check your code against the guidelines.
- Use four spaces for indentation, *never* tabs.
- Use CamelCase with initial caps for class names:

```
ClassNameWithCamelCase
```

- Use underscore to split variable, function and method names for readability. Don't use CamelCase.

```
lower_case_with_underscores
```

- Use the fact that empty strings, lists and tuples are False and don't compare boolean values using == and !=.
- Follow whitespace rules as described in **PEP 8**. Good examples:

```
spam(ham[1], {eggs: 2})
spam(1)
dict['key'] = list[index]
```

- Limit lines to 80 characters and avoid trailing whitespace. However note that wrapped lines should be *one* indentation level in from level above, except for `if`, `for`, `with`, and `while` lines which should have two levels of indentation:

```
if (foo and bar ...
        baz and foobar):
    a = 1

from foobar import (foo, bar, ...
    baz)
```

- For consistency, prefer `'` over `"` for strings, unless the string contains `'`.
- Take a look at **PEP 20** for a nice peek into a general mindset useful for Python coding.

CHAPTER 29

Extension development

Mopidy started as simply an MPD server that could play music from Spotify. Early on, Mopidy got multiple "frontends" to expose Mopidy to more than just MPD clients: for example the scrobbler frontend that scrobbles your listening history to your Last.fm account, the MPRIS frontend that integrates Mopidy into the Ubuntu Sound Menu, and the HTTP server and JavaScript player API making web based Mopidy clients possible. In Mopidy 0.9 we added support for multiple music sources without stopping and reconfiguring Mopidy: for example the local backend for playing music from your disk, the stream backend for playing Internet radio streams, and the Spotify and SoundCloud backends, for playing music directly from those services.

All of these are examples of what you can accomplish by creating a Mopidy extension. If you want to create your own Mopidy extension for something that does not exist yet, this guide to extension development will help you get your extension running in no time, and make it feel the way users would expect your extension to behave.

29.1 Anatomy of an extension

Extensions are located in a Python package called `mopidy_something` where "something" is the name of the application, library or web service you want to integrate with Mopidy. So, for example, if you plan to add support for a service named Soundspot to Mopidy, you would name your extension's Python package `mopidy_soundspot`.

The extension must be shipped with a `setup.py` file and be registered on PyPI. The name of the distribution on PyPI would be something like "Mopidy-Soundspot". Make sure to include the name "Mopidy" somewhere in that name and that you check the capitalization. This is the name users will use when they install your extension from PyPI.

Mopidy extensions must be licensed under an Apache 2.0 (like Mopidy itself), BSD, MIT or more liberal license to be able to be enlisted in the Mopidy documentation. The license text should be included in the `LICENSE` file in the root of the extension's Git repo.

Combining this together, we get the following folder structure for our extension, Mopidy-Soundspot:

```
mopidy-soundspot/           # The Git repo root
    LICENSE                 # The license text
    MANIFEST.in             # List of data files to include in PyPI package
    README.rst              # Document what it is and how to use it
```

(continues on next page)

(continued from previous page)

```
    mopidy_soundspot/        # Your code
        __init__.py
        ext.conf             # Default config for the extension
        ...
    setup.py                 # Installation script
```

Example content for the most important files follows below.

29.2 cookiecutter project template

We've also made a cookiecutter project template for creating new Mopidy extensions. If you install cookiecutter and run a single command, you're asked a few questions about the name of your extension, etc. This is used to create a folder structure similar to the above, with all the needed files and most of the details filled in for you. This saves you a lot of tedious work and copy-pasting from this howto. See the readme of cookiecutter-mopidy-ext for further details.

29.3 Example README.rst

The README file should quickly explain what the extension does, how to install it, and how to configure it. It should also contain a link to a tarball of the latest development version of the extension. It's important that this link ends with `#egg=Mopidy-Something-dev` for installation using `pip install Mopidy-Something==dev` to work.

```
****************
Mopidy-Soundspot
****************

`Mopidy <http://www.mopidy.com/>`_ extension for playing music from
`Soundspot <http://soundspot.example.com/>`_.

Requires a Soundspot Platina subscription and the pysoundspot library.


Installation
============

Install by running::

    sudo pip install Mopidy-Soundspot

Or, if available, install the Debian/Ubuntu package from `apt.mopidy.com
<http://apt.mopidy.com/>`_.


Configuration
=============

Before starting Mopidy, you must add your Soundspot username and password
to the Mopidy configuration file::

    [soundspot]
    username = alice
    password = secret
```

(continues on next page)

(continued from previous page)

```
Project resources
=================

- `Source code <https://github.com/mopidy/mopidy-soundspot>`_
- `Issue tracker <https://github.com/mopidy/mopidy-soundspot/issues>`_
- `Development branch tarball <https://github.com/mopidy/mopidy-soundspot/tarball/
↪master#egg=Mopidy-Soundspot-dev>`_


Changelog
=========

v0.1.0 (2013-09-17)
-------------------

- Initial release.
```

29.4 Example setup.py

The `setup.py` file must use setuptools, and not distutils. This is because Mopidy extensions use setuptools' entry point functionality to register themselves as available Mopidy extensions when they are installed on your system.

The example below also includes a couple of convenient tricks for reading the package version from the source code so that it is defined in a single place, and to reuse the README file as the long description of the package for the PyPI registration.

The package must have `install_requires` on `setuptools` and `Mopidy >= 0.14` (or a newer version, if your extension requires it), in addition to any other dependencies required by your extension. If you implement a Mopidy frontend or backend, you'll need to include `Pykka >= 1.1` in the requirements. The `entry_points` part must be included. The `mopidy.ext` part cannot be changed, but the innermost string should be changed. It's format is `ext_name = package_name:Extension`. `ext_name` should be a short name for your extension, typically the part after "Mopidy-" in lowercase. This name is used e.g. to name the config section for your extension. The `package_name:Extension` part is simply the Python path to the extension class that will connect the rest of the dots.

```
from __future__ import absolute_import, unicode_literals

import re
from setuptools import setup, find_packages


def get_version(filename):
    content = open(filename).read()
    metadata = dict(re.findall("__([a-z]+)__ = '([^']+)'", content))
    return metadata['version']


setup(
    name='Mopidy-Soundspot',
    version=get_version('mopidy_soundspot/__init__.py'),
    url='https://github.com/your-account/mopidy-soundspot',
    license='Apache License, Version 2.0',
    author='Your Name',
```

(continues on next page)

(continued from previous page)

```
    author_email='your-email@example.com',
    description='Very short description',
    long_description=open('README.rst').read(),
    packages=find_packages(exclude=['tests', 'tests.*']),
    zip_safe=False,
    include_package_data=True,
    install_requires=[
        'setuptools',
        'Mopidy >= 0.14',
        'Pykka >= 1.1',
        'pysoundspot',
    ],
    entry_points={
        'mopidy.ext': [
            'soundspot = mopidy_soundspot:Extension',
        ],
    },
    classifiers=[
        'Environment :: No Input/Output (Daemon)',
        'Intended Audience :: End Users/Desktop',
        'License :: OSI Approved :: Apache Software License',
        'Operating System :: OS Independent',
        'Programming Language :: Python :: 2',
        'Topic :: Multimedia :: Sound/Audio :: Players',
    ],
)
```

To make sure your README, license file and default config file is included in the package that is uploaded to PyPI, we'll also need to add a `MANIFEST.in` file:

```
include LICENSE
include MANIFEST.in
include README.rst
include mopidy_soundspot/ext.conf
```

For details on the `MANIFEST.in` file format, check out the distutils docs. check-manifest is a very useful tool to check your `MANIFEST.in` file for completeness.

29.5 Example __init__.py

The `__init__.py` file should be placed inside the `mopidy_soundspot` Python package.

The root of your Python package should have an `__version__` attribute with a **PEP 386** compliant version number, for example "0.1". Next, it should have a class named `Extension` which inherits from Mopidy's extension base class, *`mopidy.ext.Extension`*. This is the class referred to in the `entry_points` part of `setup.py`. Any imports of other files in your extension, outside of Mopidy and it's core requirements, should be kept inside methods. This ensures that this file can be imported without raising `ImportError` exceptions for missing dependencies, etc.

The default configuration for the extension is defined by the `get_default_config()` method in the `Extension` class which returns a `ConfigParser` compatible config section. The config section's name must be the same as the extension's short name, as defined in the `entry_points` part of `setup.py`, for example `soundspot`. All extensions must include an `enabled` config which normally should default to `true`. Provide good defaults for all config values so that as few users as possible will need to change them. The exception is if the config value has security implications; in that case you should default to the most secure configuration. Leave any configurations that don't have meaningful defaults blank, like `username` and `password`. In the example below, we've chosen to maintain the default config as a separate file named `ext.conf`. This makes it easy to include the default config in documentation without duplicating it.

This is `mopidy_soundspot/__init__.py`:

```
from __future__ import absolute_import, unicode_literals

import logging
import os

from mopidy import config, exceptions, ext


__version__ = '0.1'

# If you need to log, use loggers named after the current Python module
logger = logging.getLogger(__name__)


class Extension(ext.Extension):

    dist_name = 'Mopidy-Soundspot'
    ext_name = 'soundspot'
    version = __version__

    def get_default_config(self):
        conf_file = os.path.join(os.path.dirname(__file__), 'ext.conf')
        return config.read(conf_file)

    def get_config_schema(self):
        schema = super(Extension, self).get_config_schema()
        schema['username'] = config.String()
        schema['password'] = config.Secret()
        return schema

    def get_command(self):
        from .commands import SoundspotCommand
        return SoundspotCommand()

    def validate_environment(self):
        # Any manual checks of the environment to fail early.
        # Dependencies described by setup.py are checked by Mopidy, so you
        # should not check their presence here.
        pass

    def setup(self, registry):
        # You will typically only do one of the following things in a
        # single extension.

        # Register a frontend
        from .frontend import SoundspotFrontend
        registry.add('frontend', SoundspotFrontend)

        # Register a backend
        from .backend import SoundspotBackend
        registry.add('backend', SoundspotBackend)

        # Or nothing to register e.g. command extension
        pass
```

And this is `mopidy_soundspot/ext.conf`:

```
[soundspot]
enabled = true
username =
password =
```

For more detailed documentation on the extension class, see the *mopidy.ext – Extension API*.

29.6 Example frontend

If you want to *use* Mopidy's core API from your extension, then you want to implement a frontend.

The skeleton of a frontend would look like this. Notice that the frontend gets passed a reference to the core API when it's created. See the *Frontend API* for more details.

```
import pykka

from mopidy import core


class SoundspotFrontend(pykka.ThreadingActor, core.CoreListener):
    def __init__(self, config, core):
        super(SoundspotFrontend, self).__init__()
        self.core = core

    # Your frontend implementation
```

29.7 Example backend

If you want to extend Mopidy to support new music and playlist sources, you want to implement a backend. A backend does not have access to Mopidy's core API at all, but it does have a bunch of interfaces it can implement to extend Mopidy.

The skeleton of a backend would look like this. See *mopidy.backend — Backend API* for more details.

```
import pykka

from mopidy import backend


class SoundspotBackend(pykka.ThreadingActor, backend.Backend):
    def __init__(self, config, audio):
        super(SoundspotBackend, self).__init__()
        self.audio = audio

    # Your backend implementation
```

29.8 Example command

If you want to extend the Mopidy with a new helper not run from the server, such as scanning for media, adding a command is the way to go. Your top level command name will always match your extension name, but you are free to add sub-commands with names of your choosing.

The skeleton of a command would look like this. See *mopidy.commands — Commands API* for more details.

```
from mopidy import commands


class SoundspotCommand(commands.Command):
    help = 'Some text that will show up in --help'

    def __init__(self):
        super(SoundspotCommand, self).__init__()
        self.add_argument('--foo')

    def run(self, args, config, extensions):
        # Your command implementation
        return 0
```

29.9 Example web application

As of Mopidy 0.19, extensions can use Mopidy's built-in web server to host static web clients as well as Tornado and WSGI web applications. For several examples, see the *HTTP server side API* docs or explore with *Mopidy-API-Explorer* extension.

29.10 Running an extension

Once your extension is ready to go, to see it in action you'll need to register it with Mopidy. Typically this is done by running `python setup.py install` from your extension's Git repo root directory. While developing your extension and to avoid doing this every time you make a change, you can instead run `python setup.py develop` to effectively link Mopidy directly with your development files.

29.11 Python conventions

In general, it would be nice if Mopidy extensions followed the same *Code style* as Mopidy itself, as they're part of the same ecosystem. Among other things, the code style guide explains why all the above examples start with `from __future__ import absolute_import, unicode_literals`.

29.12 Use of Mopidy APIs

When writing an extension, you should only use APIs documented at *API reference*. Other parts of Mopidy, like `mopidy.internal`, may change at any time and are not something extensions should use.

Mopidy performs type checking to help catch extension bugs. This applies to both frontend calls into core and return values from backends. Additionally model fields always get validated to further guard against bad data.

29.13 Logging in extensions

For servers like Mopidy, logging is essential for understanding what's going on. We use the `logging` module from Python's standard library. When creating a logger, always namespace the logger using your Python package name as this will be visible in Mopidy's debug log:

```
import logging

logger = logging.getLogger('mopidy_soundspot')

# Or even better, use the Python module name as the logger name:
logger = logging.getLogger(__name__)
```

When logging at logging level `info` or higher (i.e. `warning`, `error`, and `critical`, but not `debug`) the log message will be displayed to all Mopidy users. Thus, the log messages at those levels should be well written and easy to understand.

As the logger name is not included in Mopidy's default logging format, you should make it obvious from the log message who is the source of the log message. For example:

```
Loaded 17 Soundspot playlists
```

Is much better than:

```
Loaded 17 playlists
```

If you want to turn on debug logging for your own extension, but not for everything else due to the amount of noise, see the docs for the `loglevels/*` config section.

29.14 Making HTTP requests from extensions

Many Mopidy extensions need to make HTTP requests to use some web API. Here's a few recommendations to those extensions.

29.14.1 Proxies

If you make HTTP requests please make sure to respect the *proxy configs*, so that all the requests you make go through the proxy configured by the Mopidy user. To make this easier for extension developers, the helper function `mopidy.httpclient.format_proxy()` was added in Mopidy 1.1. This function returns the proxy settings formatted the way Requests expects.

29.14.2 User-Agent strings

When you make HTTP requests, it's helpful for debugging and usage analysis if the client identifies itself with a proper User-Agent string. In Mopidy 1.1, we added the helper function `mopidy.httpclient.format_user_agent()`. Here's an example of how to use it:

```
>>> from mopidy import httpclient
>>> import mopidy_soundspot
>>> httpclient.format_user_agent('%s/%s' % (
...     mopidy_soundspot.Extension.dist_name, mopidy_soundspot.__version__))
u'Mopidy-SoundSpot/2.0.0 Mopidy/1.0.7 Python/2.7.10'
```

29.14.3 Example using Requests sessions

Most Mopidy extensions that make HTTP requests use the Requests library to do so. When using Requests, the most convenient way to make sure the proxy and User-Agent header is set properly is to create a Requests session object and use that object to make all your HTTP requests:

```
from mopidy import httpclient

import requests

import mopidy_soundspot


def get_requests_session(proxy_config, user_agent):
    proxy = httpclient.format_proxy(proxy_config)
    full_user_agent = httpclient.format_user_agent(user_agent)

    session = requests.Session()
    session.proxies.update({'http': proxy, 'https': proxy})
    session.headers.update({'user-agent': full_user_agent})

    return session


# ``mopidy_config`` is the config object passed to your frontend/backend
# constructor
session = get_requests_session(
    proxy_config=mopidy_config['proxy'],
    user_agent='%s/%s' % (
        mopidy_soundspot.Extension.dist_name,
        mopidy_soundspot.__version__))

response = session.get('http://example.com')
# Now do something with ``response`` and/or make further requests using the
# ``session`` object.
```

For further details, see Requests' docs on session objects.

CHAPTER 30

Glossary

backend A part of Mopidy providing music library, playlist storage and/or playback capability to the *core*. Mopidy has a backend for each music store or music service it supports. See *mopidy.backend — Backend API* for details.

core The part of Mopidy that makes multiple frontends capable of using multiple backends. The core module is also the owner of the *tracklist*. To use the core module, see *mopidy.core — Core API*.

extension A Python package that can extend Mopidy with one or more *backends*, *frontends*, or GStreamer elements like *mixers*. See *Extensions* for a list of existing extensions and *Extension development* for how to make a new extension.

frontend A part of Mopidy *using* the *core* API. Existing frontends include the *MPD server*, the MPRIS/D-Bus integration, the Last.fm scrobbler, and the *HTTP server* with JavaScript API. See *Frontend API* for details.

mixer A GStreamer element that controls audio volume.

tracklist Mopidy's name for the play queue or current playlist. The name is inspired by the MPRIS specification.

CHAPTER 31

mopidy command

31.1 Synopsis

mopidy [-h] [–version] [-q] [-v] [–save-debug-log] [–config CONFIG_FILES] [-o CONFIG_OVERRIDES] [COMMAND] . . .

31.2 Description

Mopidy is a music server which can play music both from multiple sources, like your local hard drive, radio streams, and from Spotify and SoundCloud. Searches combines results from all music sources, and you can mix tracks from all sources in your play queue. Your playlists from Spotify or SoundCloud are also available for use.

The `mopidy` command is used to start the server.

31.3 Options

`--help, -h`
Show help message and exit.

`--version`
Show Mopidy's version number and exit.

`--quiet, -q`
Show less output: warning level and higher.

`--verbose, -v`
Show more output. Repeat up to four times for even more.

`--save-debug-log`
Save debug log to the file specified in the *`logging/debug_file`* config value, typically `./mopidy.log`.

`--config` `<file|directory>`
Specify config files and directories to use. To use multiple config files or directories, separate them with a colon. The later files override the earlier ones if there's a conflict. When specifying a directory, all files ending in .conf in the directory are used.

`--option` `<option>`**`, -o`** `<option>`
Specify additional config values in the `section/key=value` format. Can be provided multiple times.

31.4 Built in commands

`config`
Show the current effective config. All configuration sources are merged together to show the effective document. Secret values like passwords are masked out. Config for disabled extensions are not included.

`deps`
Show dependencies, their versions and installation location.

31.5 Extension commands

Additionally, extensions can provide extra commands. Run *mopidy –help* for a list of what is available on your system and command-specific help. Commands for disabled extensions will be listed, but can not be run.

`local clear`
Clear local media files from the local library.

`local scan`
Scan local media files present in your library.

31.6 Files

`/etc/mopidy/mopidy.conf` System wide Mopidy configuration file.

`~/.config/mopidy/mopidy.conf` Your personal Mopidy configuration file. Overrides any configs from the system wide configuration file.

31.7 Examples

To start the music server, run:

```
mopidy
```

To start the server with an additional config file, that can override configs set in the default config files, run:

```
mopidy --config ./my-config.conf
```

To start the server and change a config value directly on the command line, run:

```
mopidy --option mpd/enabled=false
```

The `--option` flag may be repeated multiple times to change multiple configs:

```
mopidy -o mpd/enabled=false -o spotify/bitrate=320
```

The `mopidy config` output shows the effect of the *`--option`* flags:

```
mopidy -o mpd/enabled=false -o spotify/bitrate=320 config
```

31.8 Reporting bugs

Report bugs to Mopidy's issue tracker at <https://github.com/mopidy/mopidy/issues>

CHAPTER 32

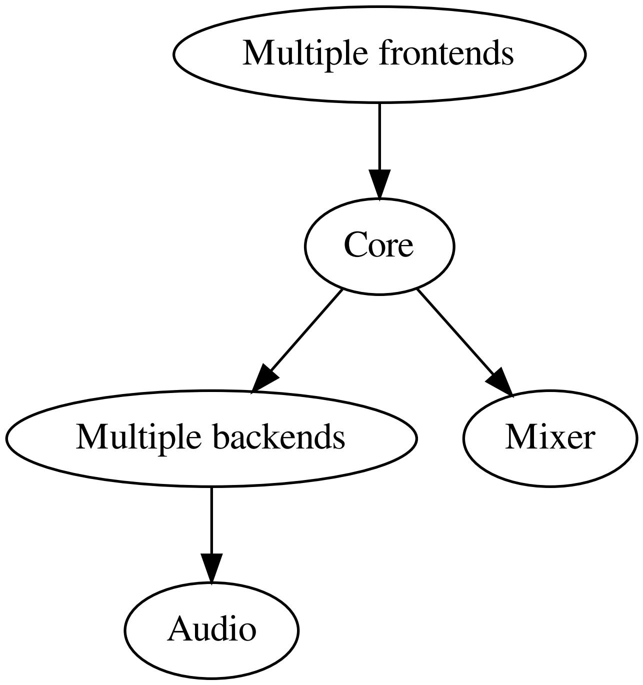
API reference

Note: Only APIs documented here are public and open for use by Mopidy extensions.

32.1 Concepts

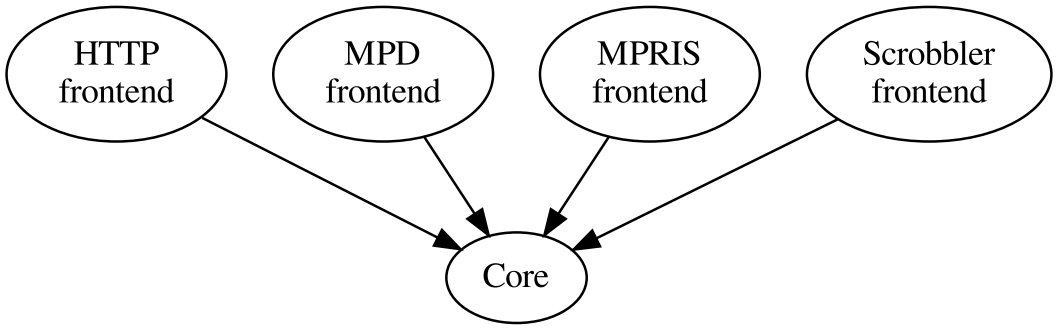
32.1.1 Architecture

The overall architecture of Mopidy is organized around multiple frontends and backends. The frontends use the core API. The core actor makes multiple backends work as one. The backends connect to various music sources. The core actor use the mixer actor to control volume, while the backends use the audio actor to play audio.



Frontends

Frontends expose Mopidy to the external world. They can implement servers for protocols like HTTP, MPD and MPRIS, and they can be used to update other services when something happens in Mopidy, like the Last.fm scrobbler frontend does. See *Frontend API* for more details.



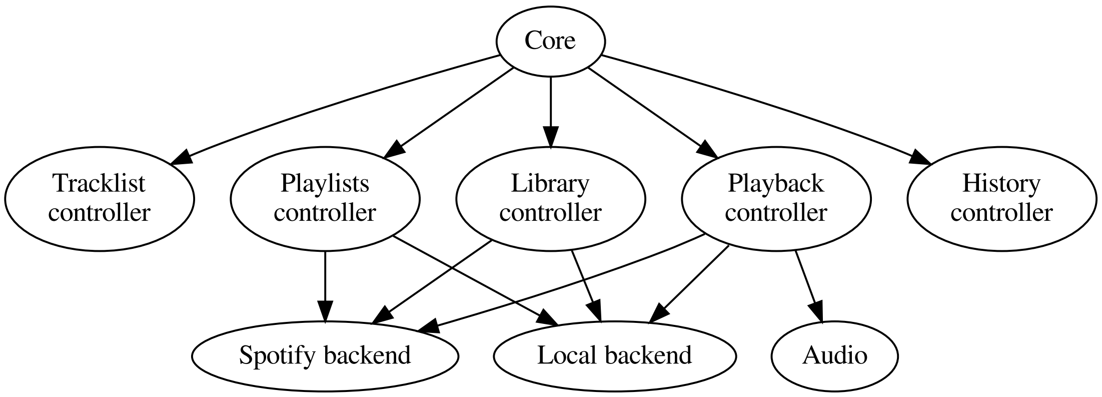
Core

The core is organized as a set of controllers with responsiblity for separate sets of functionality.

The core is the single actor that the frontends send their requests to. For every request from a frontend it calls out to one or more backends which does the real work, and when the backends respond, the core actor is responsible for combining the responses into a single response to the requesting frontend.

The core actor also keeps track of the tracklist, since it doesn't belong to a specific backend.

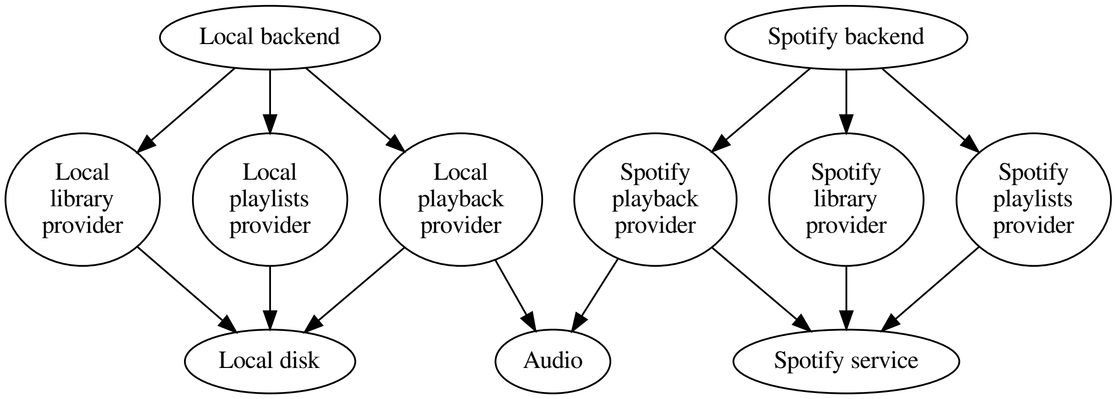
See *mopidy.core — Core API* for more details.



Backends

The backends are organized as a set of providers with responsiblity for separate sets of functionality, similar to the core actor.

Anything specific to i.e. Spotify integration or local storage is contained in the backends. To integrate with new music sources, you just add a new backend. See *mopidy.backend — Backend API* for more details.



Audio

The audio actor is a thin wrapper around the parts of the GStreamer library we use. If you implement an advanced backend, you may need to implement your own playback provider using the *mopidy.audio — Audio API*, but most backends can use the default playback provider without any changes.

Mixer

The mixer actor is responsible for volume control and muting. The default mixer use the audio actor to control volume in software. The alternative implementations are typically independent of the audio actor, but instead use some third party Python library or a serial interface to control other forms of volume controls.

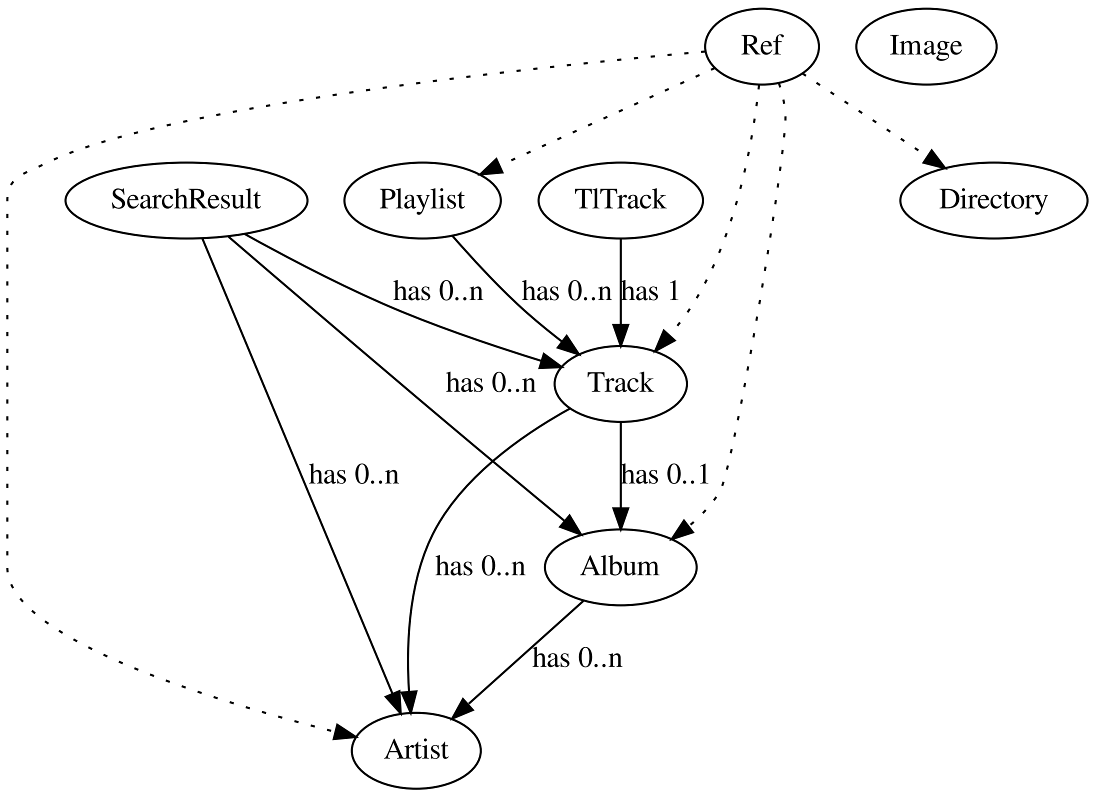
32.1.2 `mopidy.models` — Data models

These immutable data models are used for all data transfer within the Mopidy backends and between the backends and the MPD frontend. All fields are optional and immutable. In other words, they can only be set through the class constructor during instance creation. Additionally fields are type checked.

If you want to modify a model, use the `replace()` method. It accepts keyword arguments for the parts of the model you want to change, and copies the rest of the data from the model you call it on. Example:

```
>>> from mopidy.models import Track
>>> track1 = Track(name='Christmas Carol', length=171)
>>> track1
Track(artists=[], length=171, name='Christmas Carol')
>>> track2 = track1.replace(length=37)
>>> track2
Track(artists=[], length=37, name='Christmas Carol')
>>> track1
Track(artists=[], length=171, name='Christmas Carol')
```

Data model relations



Data model API

class `mopidy.models.`**`Ref`**(**args*, ***kwargs*)

Model to represent URI references with a human friendly name and type attached. This is intended for use a lightweight object "free" of metadata that can be passed around instead of using full blown models.

Parameters

- **`uri`** (*`string`*) – object URI
- **`name`** (*`string`*) – object name
- **`type`** (*`string`*) – object type

`ALBUM = u'album'`

Constant used for comparison with the *`type`* field.

`ARTIST = u'artist'`

Constant used for comparison with the *`type`* field.

`DIRECTORY = u'directory'`

Constant used for comparison with the *`type`* field.

PLAYLIST = u'playlist'
Constant used for comparison with the `type` field.

TRACK = u'track'
Constant used for comparison with the `type` field.

classmethod album(***kwargs*)
Create a `Ref` with `type` `ALBUM`.

classmethod artist(***kwargs*)
Create a `Ref` with `type` `ARTIST`.

classmethod directory(***kwargs*)
Create a `Ref` with `type` `DIRECTORY`.

name
The object name. Read-only.

classmethod playlist(***kwargs*)
Create a `Ref` with `type` `PLAYLIST`.

classmethod track(***kwargs*)
Create a `Ref` with `type` `TRACK`.

type
The object type, e.g. “artist”, “album”, “track”, “playlist”, “directory”. Read-only.

uri
The object URI. Read-only.

class mopidy.models.**Track**(**args*, ***kwargs*)

Parameters

- **uri** (*string*) – track URI
- **name** (*string*) – track name
- **artists** (list of `Artist`) – track artists
- **album** (`Album`) – track album
- **composers** (*string*) – track composers
- **performers** (*string*) – track performers
- **genre** (*string*) – track genre
- **track_no** (integer or `None` if unknown) – track number in album
- **disc_no** (integer or `None` if unknown) – disc number in album
- **date** (*string*) – track release date (YYYY or YYYY-MM-DD)
- **length** (integer or `None` if there is no duration) – track length in milliseconds
- **bitrate** (*integer*) – bitrate in kbit/s
- **comment** (*string*) – track comment
- **musicbrainz_id** (*string*) – MusicBrainz ID
- **last_modified** (integer or `None` if unknown) – Represents last modification time

album
The track `Album`. Read-only.

artists
A set of track artists. Read-only.

bitrate
The track's bitrate in kbit/s. Read-only.

comment
The track comment. Read-only.

composers
A set of track composers. Read-only.

date
The track release date. Read-only.

disc_no
The disc number in the album. Read-only.

genre
The track genre. Read-only.

last_modified
Integer representing when the track was last modified. Exact meaning depends on source of track. For local files this is the modification time in milliseconds since Unix epoch. For other backends it could be an equivalent timestamp or simply a version counter.

length
The track length in milliseconds. Read-only.

musicbrainz_id
The MusicBrainz ID of the track. Read-only.

name
The track name. Read-only.

performers
A set of track performers'. Read-only.

track_no
The track number in the album. Read-only.

uri
The track URI. Read-only.

class mopidy.models.**Album**(**args*, ***kwargs*)

Parameters

- **uri** (*string*) – album URI
- **name** (*string*) – album name
- **artists** (list of *Artist*) – album artists
- **num_tracks** (integer or None if unknown) – number of tracks in album
- **num_discs** (integer or None if unknown) – number of discs in album
- **date** (*string*) – album release date (YYYY or YYYY-MM-DD)
- **musicbrainz_id** (*string*) – MusicBrainz ID
- **images** (*list of strings*) – album image URIs

artists
A set of album artists. Read-only.

date
The album release date. Read-only.

images
The album image URIs. Read-only.

musicbrainz_id
The MusicBrainz ID of the album. Read-only.

name
The album name. Read-only.

num_discs
The number of discs in the album. Read-only.

num_tracks
The number of tracks in the album. Read-only.

uri
The album URI. Read-only.

class `mopidy.models.`**Artist**(**args*, ***kwargs*)

Parameters

- **uri** (*string*) – artist URI
- **name** (*string*) – artist name
- **sortname** (*string*) – artist name for sorting
- **musicbrainz_id** (*string*) – MusicBrainz ID

musicbrainz_id
The MusicBrainz ID of the artist. Read-only.

name
The artist name. Read-only.

sortname
Artist name for better sorting, e.g. with articles stripped

uri
The artist URI. Read-only.

class `mopidy.models.`**Playlist**(**args*, ***kwargs*)

Parameters

- **uri** (*string*) – playlist URI
- **name** (*string*) – playlist name
- **tracks** (list of *Track* elements) – playlist's tracks
- **last_modified** (*int*) – playlist's modification time in milliseconds since Unix epoch

last_modified
The playlist modification time in milliseconds since Unix epoch. Read-only.

Integer, or `None` if unknown.

length
The number of tracks in the playlist. Read-only.

name
The playlist name. Read-only.

tracks
The playlist's tracks. Read-only.

uri
The playlist URI. Read-only.

class mopidy.models.**Image**(**args*, ***kwargs*)

Parameters

- **uri** (*string*) – URI of the image
- **width** (*int*) – Optional width of image or None
- **height** (*int*) – Optional height of image or None

height
Optional height of the image or None. Read-only.

uri
The image URI. Read-only.

width
Optional width of the image or None. Read-only.

class mopidy.models.**TlTrack**(**args*, ***kwargs*)
A tracklist track. Wraps a regular track and it's tracklist ID.

The use of *TlTrack* allows the same track to appear multiple times in the tracklist.

This class also accepts it's parameters as positional arguments. Both arguments must be provided, and they must appear in the order they are listed here.

This class also supports iteration, so your extract its values like this:

```
(tlid, track) = tl_track
```

Parameters

- **tlid** (*int*) – tracklist ID
- **track** (*Track*) – the track

tlid
The tracklist ID. Read-only.

track
The track. Read-only.

class mopidy.models.**SearchResult**(**args*, ***kwargs*)

Parameters

- **uri** (*string*) – search result URI
- **tracks** (list of *Track* elements) – matching tracks
- **artists** (list of *Artist* elements) – matching artists
- **albums** (list of *Album* elements) – matching albums

albums
The albums matching the search query. Read-only.

artists
The artists matching the search query. Read-only.

tracks
The tracks matching the search query. Read-only.

uri
The search result URI. Read-only.

Data model helpers

class mopidy.models.**ImmutableObject**(**args*, ***kwargs*)
Superclass for immutable objects whose fields can only be modified via the constructor.

This version of this class has been retained to avoid breaking any clients relying on it's behavior. Internally in Mopidy we now use `ValidatedImmutableObject` for type safety and it's much smaller memory footprint.

Parameters **kwargs** (*any*) – kwargs to set as fields on the object

copy(***values*)
Deprecated since version 1.1: Use `replace()` instead.

replace(***kwargs*)
Replace the fields in the model and return a new instance

Examples:

```
# Returns a track with a new name
Track(name='foo').replace(name='bar')
# Return an album with a new number of tracks
Album(num_tracks=2).replace(num_tracks=5)
```

Parameters **kwargs** (*any*) – kwargs to set as fields on the object

Return type instance of the model with replaced fields

class mopidy.models.**ValidatedImmutableObject**(**args*, ***kwargs*)
Superclass for immutable objects whose fields can only be modified via the constructor. Fields should be `Field` instances to ensure type safety in our models.

Note that since these models can not be changed, we heavily memoize them to save memory. So constructing a class with the same arguments twice will give you the same instance twice.

replace(***kwargs*)
Replace the fields in the model and return a new instance

Examples:

```
# Returns a track with a new name
Track(name='foo').replace(name='bar')
# Return an album with a new number of tracks
Album(num_tracks=2).replace(num_tracks=5)
```

Note that internally we memoize heavily to keep memory usage down given our overly repetitive data structures. So you might get an existing instance if it contains the same values.

Parameters **kwargs** (*any*) – kwargs to set as fields on the object

Return type instance of the model with replaced fields

Data model (de)serialization

`mopidy.models.`**`model_json_decoder`**`(`*dct*`)`

Automatically deserialize Mopidy models from JSON.

Usage:

```
>>> import json
>>> json.loads(
...     '{"a_track": {"__model__": "Track", "name": "name"}}',
...     object_hook=model_json_decoder)
{u'a_track': Track(artists=[], name=u'name')}
```

`class` `mopidy.models.`**`ModelJSONEncoder`**`(`*skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, encoding='utf-8', default=None*`)`

Automatically serialize Mopidy models to JSON.

Usage:

```
>>> import json
>>> json.dumps({'a_track': Track(name='name')}, cls=ModelJSONEncoder)
'{"a_track": {"__model__": "Track", "name": "name"}}'
```

Data model field types

`class` `mopidy.models.fields.`**`Field`**`(`*default=None, type=None, choices=None*`)`

Base field for use in `ValidatedImmutableObject`. These fields are responsible for type checking and other data sanitation in our models.

For simplicity fields use the Python descriptor protocol to store the values in the instance dictionary. Also note that fields are mutable if the object they are attached to allow it.

Default values will be validated with the exception of `None`.

Parameters

- **`default`** – default value for field
- **`type`** – if set the field value must be of this type
- **`choices`** – if set the field value must be one of these

`class` `mopidy.models.fields.`**`String`**`(`*default=None*`)`

Specialized *`Field`* which is wired up for bytes and unicode.

Parameters **`default`** – default value for field

`class` `mopidy.models.fields.`**`Identifier`**`(`*default=None*`)`

`Field` for storing ASCII values such as GUIDs or other identifiers.

Values will be interned.

Parameters **`default`** – default value for field

`class` `mopidy.models.fields.`**`URI`**`(`*default=None*`)`

`Field` for storing URIs

Values will be interned, currently not validated.

Parameters **`default`** – default value for field

class `mopidy.models.fields.`**`Date`**(*default=None*)

`Field` for storing ISO 8601 dates as a string.

Supported formats are `YYYY-MM-DD`, `YYYY-MM` and `YYYY`, currently not validated.

Parameters **`default`** – default value for field

class `mopidy.models.fields.`**`Integer`**(*default=None*, *min=None*, *max=None*)

`Field` for storing integer numbers.

Parameters

- **`default`** – default value for field
- **`min`** – field value must be larger or equal to this value when set
- **`max`** – field value must be smaller or equal to this value when set

class `mopidy.models.fields.`**`Collection`**(*type*, *container=<type 'tuple'>*)

`Field` for storing collections of a given type.

Parameters

- **`type`** – all items stored in the collection must be of this type
- **`container`** – the type to store the items in

32.2 Basics

32.2.1 `mopidy.core` — Core API

The core API is the interface that is used by frontends like `mopidy.http` and `mopidy.mpd`. The core layer is in between the frontends and the backends. Don't forget that you will be accessing core as a Pykka actor. If you are only interested in being notified about changes in core see `CoreListener`.

Changed in version 1.1: All core API calls are now type checked.

Changed in version 1.1: All backend return values are now type checked.

class `mopidy.core.`**`Core`**(*config=None*, *mixer=None*, *backends=None*, *audio=None*)

`tracklist`
Manages everything related to the list of tracks we will play. See `TracklistController`.

`playback`
Manages playback state and the current playing track. See `PlaybackController`.

`library`
Manages the music library, e.g. searching and browsing for music. See `LibraryController`.

`playlists`
Manages stored playlists. See `PlaylistsController`.

`mixer`
Manages volume and muting. See `MixerController`.

`history`
Keeps record of what tracks have been played. See `HistoryController`.

get_uri_schemes()
Get list of URI schemes we can handle

get_version()
Get version of the Mopidy core API

Tracklist controller

class mopidy.core.**TracklistController**(*core*)

Manipulating

TracklistController.**add**(*tracks=None*, *at_position=None*, *uri=None*, *uris=None*)
Add tracks to the tracklist.

If `uri` is given instead of `tracks`, the URI is looked up in the library and the resulting tracks are added to the tracklist.

If `uris` is given instead of `uri` or `tracks`, the URIs are looked up in the library and the resulting tracks are added to the tracklist.

If `at_position` is given, the tracks are inserted at the given position in the tracklist. If `at_position` is not given, the tracks are appended to the end of the tracklist.

Triggers the `mopidy.core.CoreListener.tracklist_changed()` event.

Parameters

- **tracks** (list of `mopidy.models.Track` or `None`) – tracks to add
- **at_position** (int or `None`) – position in tracklist to add tracks
- **uri** (string or `None`) – URI for tracks to add
- **uris** (list of string or `None`) – list of URIs for tracks to add

Return type list of `mopidy.models.TlTrack`

New in version 1.0: The `uris` argument.

Deprecated since version 1.0: The `tracks` and `uri` arguments. Use `uris`.

TracklistController.**remove**(*criteria=None*, ***kwargs*)
Remove the matching tracks from the tracklist.

Uses `filter()` to lookup the tracks to remove.

Triggers the `mopidy.core.CoreListener.tracklist_changed()` event.

Parameters **criteria** (`dict`) – on or more criteria to match by

Return type list of `mopidy.models.TlTrack` that was removed

Deprecated since version 1.1: Providing the criteria via `kwargs`.

TracklistController.**clear**()
Clear the tracklist.

Triggers the `mopidy.core.CoreListener.tracklist_changed()` event.

TracklistController.**move**(*start*, *end*, *to_position*)
Move the tracks in the slice `[start:end]` to `to_position`.

Triggers the `mopidy.core.CoreListener.tracklist_changed()` event.

Parameters

- **start** (*int*) – position of first track to move
- **end** (*int*) – position after last track to move
- **to_position** (*int*) – new position for the tracks

`TracklistController.`**`shuffle`**`(`*start=None*, *end=None*`)`

Shuffles the entire tracklist. If `start` and `end` is given only shuffles the slice `[start:end]`.

Triggers the *`mopidy.core.CoreListener.tracklist_changed()`* event.

Parameters

- **start** (int or `None`) – position of first track to shuffle
- **end** (int or `None`) – position after last track to shuffle

Current state

`TracklistController.`**`get_tl_tracks`**`()`

Get tracklist as list of *`mopidy.models.TlTrack`*.

`TracklistController.`**`index`**`(`*tl_track=None*, *tlid=None*`)`

The position of the given track in the tracklist.

If neither *tl_track* or *tlid* is given we return the index of the currently playing track.

Parameters

- **tl_track** (*`mopidy.models.TlTrack`* or `None`) – the track to find the index of
- **tlid** (`int` or `None`) – TLID of the track to find the index of

Return type `int` or `None`

New in version 1.1: The *tlid* parameter

`TracklistController.`**`get_version`**`()`

Get the tracklist version.

Integer which is increased every time the tracklist is changed. Is not reset before Mopidy is restarted.

`TracklistController.`**`get_length`**`()`

Get length of the tracklist.

`TracklistController.`**`get_tracks`**`()`

Get tracklist as list of *`mopidy.models.Track`*.

`TracklistController.`**`slice`**`(`*start*, *end*`)`

Returns a slice of the tracklist, limited by the given start and end positions.

Parameters

- **start** (*int*) – position of first track to include in slice
- **end** (*int*) – position after last track to include in slice

Return type *`mopidy.models.TlTrack`*

`TracklistController.`**`filter`**`(`*criteria=None*, ***kwargs*`)`

Filter the tracklist by the given criterias.

A criteria consists of a model field to check and a list of values to compare it against. If the model field matches one of the values, it may be returned.

Only tracks that matches all the given criterias are returned.

Examples:

```
# Returns tracks with TLIDs 1, 2, 3, or 4 (tracklist ID)
filter({'tlid': [1, 2, 3, 4]})

# Returns track with URIs 'xyz' or 'abc'
filter({'uri': ['xyz', 'abc']})

# Returns track with a matching TLIDs (1, 3 or 6) and a
# matching URI ('xyz' or 'abc')
filter({'tlid': [1, 3, 6], 'uri': ['xyz', 'abc']})
```

Parameters **`criteria`** (*`dict, of (string, list) pairs`*) – on or more criteria to match by

Return type list of *`mopidy.models.TlTrack`*

Deprecated since version 1.1: Providing the criteria via `kwargs`.

Future state

`TracklistController.`**`get_eot_tlid`**`()`

The TLID of the track that will be played after the current track.

Not necessarily the same TLID as returned by *`get_next_tlid()`*.

Return type `int` or `None`

New in version 1.1.

`TracklistController.`**`get_next_tlid`**`()`

The tlid of the track that will be played if calling *`mopidy.core.PlaybackController.next()`*.

For normal playback this is the next track in the tracklist. If repeat is enabled the next track can loop around the tracklist. When random is enabled this should be a random track, all tracks should be played once before the tracklist repeats.

Return type `int` or `None`

New in version 1.1.

`TracklistController.`**`get_previous_tlid`**`()`

Returns the TLID of the track that will be played if calling *`mopidy.core.PlaybackController.previous()`*.

For normal playback this is the previous track in the tracklist. If random and/or consume is enabled it should return the current track instead.

Return type `int` or `None`

New in version 1.1.

`TracklistController.`**`eot_track`**`(`*tl_track*`)`

The track that will be played after the given track.

Not necessarily the same track as *`next_track()`*.

Parameters **`tl_track`** (*`mopidy.models.TlTrack`* or `None`) – the reference track

Return type *`mopidy.models.TlTrack`* or `None`

`TracklistController.`**`next_track`**(*tl_track*)

The track that will be played if calling *`mopidy.core.PlaybackController.next()`*.

For normal playback this is the next track in the tracklist. If repeat is enabled the next track can loop around the tracklist. When random is enabled this should be a random track, all tracks should be played once before the tracklist repeats.

Parameters **`tl_track`** (*`mopidy.models.TlTrack`* or `None`) – the reference track

Return type *`mopidy.models.TlTrack`* or `None`

`TracklistController.`**`previous_track`**(*tl_track*)

Returns the track that will be played if calling *`mopidy.core.PlaybackController.previous()`*.

For normal playback this is the previous track in the tracklist. If random and/or consume is enabled it should return the current track instead.

Parameters **`tl_track`** (*`mopidy.models.TlTrack`* or `None`) – the reference track

Return type *`mopidy.models.TlTrack`* or `None`

Options

`TracklistController.`**`get_consume`**()

Get consume mode.

`True` Tracks are removed from the tracklist when they have been played.

`False` Tracks are not removed from the tracklist.

`TracklistController.`**`set_consume`**(*value*)

Set consume mode.

`True` Tracks are removed from the tracklist when they have been played.

`False` Tracks are not removed from the tracklist.

`TracklistController.`**`get_random`**()

Get random mode.

`True` Tracks are selected at random from the tracklist.

`False` Tracks are played in the order of the tracklist.

`TracklistController.`**`set_random`**(*value*)

Set random mode.

`True` Tracks are selected at random from the tracklist.

`False` Tracks are played in the order of the tracklist.

`TracklistController.`**`get_repeat`**()

Get repeat mode.

`True` The tracklist is played repeatedly.

`False` The tracklist is played once.

`TracklistController.`**`set_repeat`**(*value*)

Set repeat mode.

To repeat a single track, set both `repeat` and `single`.

`True` The tracklist is played repeatedly.

False The tracklist is played once.

`TracklistController.`**`get_single`**`()`

Get single mode.

True Playback is stopped after current song, unless in `repeat` mode.

False Playback continues after current song.

`TracklistController.`**`set_single`**`(`*value*`)`

Set single mode.

True Playback is stopped after current song, unless in `repeat` mode.

False Playback continues after current song.

Playback controller

class `mopidy.core.`**`PlaybackController`**`(`*backends*, *core*`)`

Playback control

`PlaybackController.`**`play`**`(`*tl_track=None*, *tlid=None*`)`

Play the given track, or if the given tl_track and tlid is `None`, play the currently active track.

Note that the track **must** already be in the tracklist.

Parameters

- **tl_track** (*mopidy.models.TlTrack* or `None`) – track to play
- **tlid** (`int` or `None`) – TLID of the track to play

`PlaybackController.`**`next`**`()`

Change to the next track.

The current playback state will be kept. If it was playing, playing will continue. If it was paused, it will still be paused, etc.

`PlaybackController.`**`previous`**`()`

Change to the previous track.

The current playback state will be kept. If it was playing, playing will continue. If it was paused, it will still be paused, etc.

`PlaybackController.`**`stop`**`()`

Stop playing.

`PlaybackController.`**`pause`**`()`

Pause playback.

`PlaybackController.`**`resume`**`()`

If paused, resume playing the current track.

`PlaybackController.`**`seek`**`(`*time_position*`)`

Seeks to time position given in milliseconds.

Parameters **time_position** (*int*) – time position in milliseconds

Return type `True` if successful, else `False`

Current track

`PlaybackController.`**`get_current_tl_track`**`()`

Get the currently playing or selected track.

Returns a *`mopidy.models.TlTrack`* or `None`.

`PlaybackController.`**`get_current_track`**`()`

Get the currently playing or selected track.

Extracted from *`get_current_tl_track()`* for convenience.

Returns a *`mopidy.models.Track`* or `None`.

`PlaybackController.`**`get_stream_title`**`()`

Get the current stream title or `None`.

`PlaybackController.`**`get_time_position`**`()`

Get time position in milliseconds.

Playback states

`PlaybackController.`**`get_state`**`()`

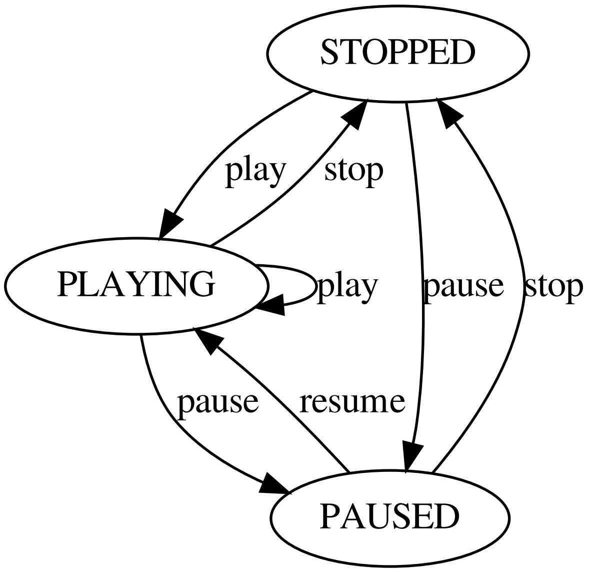
Get The playback state.

`PlaybackController.`**`set_state`**`(`*new_state*`)`

Set the playback state.

Must be `PLAYING`, `PAUSED`, or `STOPPED`.

Possible states and transitions:



class `mopidy.core.`**`PlaybackState`**

`STOPPED = 'stopped'`

`PLAYING = 'playing'`

`PAUSED = 'paused'`

Library controller

class `mopidy.core.`**`LibraryController`**

`LibraryController.`**`browse`**(*uri*)

Browse directories and tracks at the given `uri`.

`uri` is a string which represents some directory belonging to a backend. To get the intial root directories for backends pass `None` as the URI.

Returns a list of *`mopidy.models.Ref`* objects for the directories and tracks at the given `uri`.

The *`Ref`* objects representing tracks keep the track's original URI. A matching pair of objects can look like this:

```
Track(uri='dummy:/foo.mp3', name='foo', artists=..., album=...)
Ref.track(uri='dummy:/foo.mp3', name='foo')
```

The *`Ref`* objects representing directories have backend specific URIs. These are opaque values, so no one but the backend that created them should try and derive any meaning from them. The only valid exception to this is checking the scheme, as it is used to route browse requests to the correct backend.

For example, the dummy library's `/bar` directory could be returned like this:

```
Ref.directory(uri='dummy:directory:/bar', name='bar')
```

Parameters **`uri`** (*`string`*) – URI to browse

Return type list of *`mopidy.models.Ref`*

New in version 0.18.

`LibraryController.`**`search`**(*query=None*, *uris=None*, *exact=False*, ***kwargs*)

Search the library for tracks where `field` contains `values`.

If `uris` is given, the search is limited to results from within the URI roots. For example passing `uris=['file:']` will limit the search to the local backend.

Examples:

```
# Returns results matching 'a' in any backend
search({'any': ['a']})

# Returns results matching artist 'xyz' in any backend
search({'artist': ['xyz']})

# Returns results matching 'a' and 'b' and artist 'xyz' in any
# backend
search({'any': ['a', 'b'], 'artist': ['xyz']})

# Returns results matching 'a' if within the given URI roots
# "file:///media/music" and "spotify:"
```

(continues on next page)

(continued from previous page)

```
search({'any': ['a']}, uris=['file:///media/music', 'spotify:'])

# Returns results matching artist 'xyz' and 'abc' in any backend
search({'artist': ['xyz', 'abc']})
```

Parameters

- **query** (*dict*) – one or more queries to search for
- **uris** (list of string or `None`) – zero or more URI roots to limit the search to
- **exact** (`bool`) – if the search should use exact matching

Return type list of *mopidy.models.SearchResult*

New in version 1.0: The `exact` keyword argument, which replaces *find_exact()*.

Deprecated since version 1.0: Previously, if the query was empty, and the backend could support it, all available tracks were returned. This has not changed, but it is strongly discouraged. No new code should rely on this behavior.

Deprecated since version 1.1: Providing the search query via `kwargs` is no longer supported.

`LibraryController.`**`lookup`**(*uri=None*, *uris=None*)

Lookup the given URIs.

If the URI expands to multiple tracks, the returned list will contain them all.

Parameters

- **uri** (string or `None`) – track URI
- **uris** (list of string or `None`) – track URIs

Return type list of *mopidy.models.Track* if uri was set or {uri: list of *mopidy.models.Track*} if uris was set.

New in version 1.0: The `uris` argument.

Deprecated since version 1.0: The `uri` argument. Use `uris` instead.

`LibraryController.`**`refresh`**(*uri=None*)

Refresh library. Limit to URI and below if an URI is given.

Parameters **uri** (*string*) – directory or track URI

`LibraryController.`**`get_images`**(*uris*)

Lookup the images for the given URIs

Backends can use this to return image URIs for any URI they know about be it tracks, albums, playlists... The lookup result is a dictionary mapping the provided URIs to lists of images.

Unknown URIs or URIs the corresponding backend couldn't find anything for will simply return an empty list for that URI.

Parameters **uris** (*list of string*) – list of URIs to find images for

Return type {uri: tuple of *mopidy.models.Image*}

New in version 1.0.

`LibraryController.`**`get_distinct`**(*field*, *query=None*)

List distinct values for a given field from the library.

This has mainly been added to support the list commands the MPD protocol supports in a more sane fashion. Other frontends are not recommended to use this method.

Parameters

- **`field`** (*`string`*) – One of `track`, `artist`, `albumartist`, `album`, `composer`, `performer`, `date` or `genre`.
- **`query`** (*`dict`*) – Query to use for limiting results, see *`search()`* for details about the query format.

Return type set of values corresponding to the requested field type.

New in version 1.0.

Playlists controller

class `mopidy.core.`**`PlaylistsController`**

Fetching

`PlaylistsController.`**`as_list`**`()`

Get a list of the currently available playlists.

Returns a list of *`Ref`* objects referring to the playlists. In other words, no information about the playlists' content is given.

Return type list of *`mopidy.models.Ref`*

New in version 1.0.

`PlaylistsController.`**`get_items`**`(`*uri*`)`

Get the items in a playlist specified by `uri`.

Returns a list of *`Ref`* objects referring to the playlist's items.

If a playlist with the given `uri` doesn't exist, it returns `None`.

Return type list of *`mopidy.models.Ref`*, or `None`

New in version 1.0.

`PlaylistsController.`**`lookup`**`(`*uri*`)`

Lookup playlist with given URI in both the set of playlists and in any other playlist sources. Returns `None` if not found.

Parameters **`uri`** (*`string`*) – playlist URI

Return type *`mopidy.models.Playlist`* or `None`

`PlaylistsController.`**`refresh`**`(`*uri_scheme=None*`)`

Refresh the playlists in *`playlists`*.

If `uri_scheme` is `None`, all backends are asked to refresh. If `uri_scheme` is an URI scheme handled by a backend, only that backend is asked to refresh. If `uri_scheme` doesn't match any current backend, nothing happens.

Parameters **`uri_scheme`** (*`string`*) – limit to the backend matching the URI scheme

Manipulating

`PlaylistsController.`**`create`**`(`*name*, *uri_scheme=None*`)`

Create a new playlist.

If `uri_scheme` matches an URI scheme handled by a current backend, that backend is asked to create the playlist. If `uri_scheme` is `None` or doesn't match a current backend, the first backend is asked to create the playlist.

All new playlists must be created by calling this method, and **not** by creating new instances of `mopidy.models.Playlist`.

Parameters

- **`name`** (*string*) – name of the new playlist
- **`uri_scheme`** (*string*) – use the backend matching the URI scheme

Return type `mopidy.models.Playlist` or `None`

`PlaylistsController.`**`save`**`(`*playlist*`)`

Save the playlist.

For a playlist to be saveable, it must have the `uri` attribute set. You must not set the `uri` atribute yourself, but use playlist objects returned by `create()` or retrieved from `playlists`, which will always give you saveable playlists.

The method returns the saved playlist. The return playlist may differ from the saved playlist. E.g. if the playlist name was changed, the returned playlist may have a different URI. The caller of this method must throw away the playlist sent to this method, and use the returned playlist instead.

If the playlist's URI isn't set or doesn't match the URI scheme of a current backend, nothing is done and `None` is returned.

Parameters **`playlist`** (`mopidy.models.Playlist`) – the playlist

Return type `mopidy.models.Playlist` or `None`

`PlaylistsController.`**`delete`**`(`*uri*`)`

Delete playlist identified by the URI.

If the URI doesn't match the URI schemes handled by the current backends, nothing happens.

Parameters **`uri`** (*string*) – URI of the playlist to delete

Mixer controller

`class` `mopidy.core.`**`MixerController`**

`MixerController.`**`get_mute`**`()`

Get mute state.

`True` if muted, `False` unmuted, `None` if unknown.

`MixerController.`**`set_mute`**`(`*mute*`)`

Set mute state.

`True` to mute, `False` to unmute.

Returns `True` if call is successful, otherwise `False`.

`MixerController.`**`get_volume`**`()`

Get the volume.

Integer in range [0..100] or `None` if unknown.

The volume scale is linear.

`MixerController.`**`set_volume`**`(`*volume*`)`

Set the volume.

The volume is defined as an integer in range [0..100].

The volume scale is linear.

Returns `True` if call is successful, otherwise `False`.

History controller

`class` `mopidy.core.`**`HistoryController`**

`HistoryController.`**`get_history`**`()`

Get the track history.

The timestamps are milliseconds since epoch.

Returns the track history

Return type list of (timestamp, *`mopidy.models.Ref`*) tuples

`HistoryController.`**`get_length`**`()`

Get the number of tracks in the history.

Returns the history length

Return type int

Core events

`class` `mopidy.core.`**`CoreListener`**

Marker interface for recipients of events sent by the core actor.

Any Pykka actor that mixes in this class will receive calls to the methods defined here when the corresponding events happen in the core actor. This interface is used both for looking up what actors to notify of the events, and for providing default implementations for those listeners that are not interested in all events.

`mute_changed``(`*mute*`)`

Called whenever the mute state is changed.

MAY be implemented by actor.

Parameters **`mute`** (*`boolean`*) – the new mute state

`on_event``(`*event*, ***kwargs*`)`

Called on all events.

MAY be implemented by actor. By default, this method forwards the event to the specific event methods.

Parameters

- **`event`** (*`string`*) – the event name
- **`kwargs`** – any other arguments to the specific event handlers

`options_changed`()
Called whenever an option is changed.

MAY be implemented by actor.

`playback_state_changed`(*old_state*, *new_state*)
Called whenever playback state is changed.

MAY be implemented by actor.

Parameters

- **`old_state`** (string from *`mopidy.core.PlaybackState`* field) – the state before the change
- **`new_state`** (string from *`mopidy.core.PlaybackState`* field) – the state after the change

`playlist_changed`(*playlist*)
Called whenever a playlist is changed.

MAY be implemented by actor.

Parameters **`playlist`** (*`mopidy.models.Playlist`*) – the changed playlist

`playlist_deleted`(*uri*)
Called whenever a playlist is deleted.

MAY be implemented by actor.

Parameters **`uri`** (*`string`*) – the URI of the deleted playlist

`playlists_loaded`()
Called when playlists are loaded or refreshed.

MAY be implemented by actor.

`seeked`(*time_position*)
Called whenever the time position changes by an unexpected amount, e.g. at seek to a new time position.

MAY be implemented by actor.

Parameters **`time_position`** (*`int`*) – the position that was seeked to in milliseconds

`static send`(*event*, ***kwargs*)
Helper to allow calling of core listener events

`stream_title_changed`(*title*)
Called whenever the currently playing stream title changes.

MAY be implemented by actor.

`track_playback_ended`(*tl_track*, *time_position*)
Called whenever playback of a track ends.

MAY be implemented by actor.

Parameters

- **`tl_track`** (*`mopidy.models.TlTrack`*) – the track that was played before playback stopped
- **`time_position`** (*`int`*) – the time position in milliseconds

`track_playback_paused`(*tl_track*, *time_position*)
Called whenever track playback is paused.

MAY be implemented by actor.

Parameters

- **tl_track** (*mopidy.models.TlTrack*) – the track that was playing when playback paused
- **time_position** (*int*) – the time position in milliseconds

track_playback_resumed(*tl_track*, *time_position*)
Called whenever track playback is resumed.

MAY be implemented by actor.

Parameters

- **tl_track** (*mopidy.models.TlTrack*) – the track that was playing when playback resumed
- **time_position** (*int*) – the time position in milliseconds

track_playback_started(*tl_track*)
Called whenever a new track starts playing.

MAY be implemented by actor.

Parameters **tl_track** (*mopidy.models.TlTrack*) – the track that just started playing

tracklist_changed()
Called whenever the tracklist is changed.

MAY be implemented by actor.

volume_changed(*volume*)
Called whenever the volume is changed.

MAY be implemented by actor.

Parameters **volume** (*int*) – the new volume in the range [0..100]

Deprecated API features

Warning: Though these features still work, they are slated to go away in the next major Mopidy release.

Core

Core.**version**
Deprecated since version 1.0: Use *get_version()* instead.

Core.**uri_schemes**
Deprecated since version 1.0: Use *get_uri_schemes()* instead.

TracklistController

TracklistController.**tl_tracks**
Deprecated since version 1.0: Use *get_tl_tracks()* instead.

TracklistController.**tracks**
Deprecated since version 1.0: Use *get_tracks()* instead.

`TracklistController.`**`version`**
Deprecated since version 1.0: Use `get_version()` instead.

`TracklistController.`**`length`**
Deprecated since version 1.0: Use `get_length()` instead.

`TracklistController.`**`consume`**
Deprecated since version 1.0: Use `get_consume()` and `set_consume()` instead.

`TracklistController.`**`random`**
Deprecated since version 1.0: Use `get_random()` and `set_random()` instead.

`TracklistController.`**`repeat`**
Deprecated since version 1.0: Use `get_repeat()` and `set_repeat()` instead.

`TracklistController.`**`single`**
Deprecated since version 1.0: Use `get_single()` and `set_single()` instead.

PlaylistsController

`PlaybackController.`**`get_mute`**`()`
Deprecated since version 1.0: Use `core.mixer.get_mute()` instead.

`PlaybackController.`**`get_volume`**`()`
Deprecated since version 1.0: Use `core.mixer.get_volume()` instead.

`PlaybackController.`**`current_tl_track`**
Deprecated since version 1.0: Use `get_current_tl_track()` instead.

`PlaybackController.`**`current_track`**
Deprecated since version 1.0: Use `get_current_track()` instead.

`PlaybackController.`**`state`**
Deprecated since version 1.0: Use `get_state()` and `set_state()` instead.

`PlaybackController.`**`time_position`**
Deprecated since version 1.0: Use `get_time_position()` instead.

`PlaybackController.`**`mute`**
Deprecated since version 1.0: Use `core.mixer.get_mute()` and `core.mixer.set_mute()` instead.

`PlaybackController.`**`volume`**
Deprecated since version 1.0: Use `core.mixer.get_volume()` and `core.mixer.set_volume()` instead.

LibraryController

`LibraryController.`**`find_exact`**`(`*query=None*, *uris=None*, ***kwargs*`)`
Search the library for tracks where `field` is `values`.

Deprecated since version 1.0: Use `search()` with `exact` set.

PlaybackController

`PlaylistsController.`**`filter`**`(`*criteria=None*, ***kwargs*`)`
Filter playlists by the given criterias.

Examples:

```
# Returns track with name 'a'
filter({'name': 'a'})

# Returns track with URI 'xyz'
filter({'uri': 'xyz'})

# Returns track with name 'a' and URI 'xyz'
filter({'name': 'a', 'uri': 'xyz'})
```

Parameters **`criteria`** (*`dict`*) – one or more criteria to match by

Return type list of *`mopidy.models.Playlist`*

Deprecated since version 1.0: Use *`as_list()`* and filter yourself.

`PlaylistsController.`**`get_playlists`**(*include_tracks=True*)

Get the available playlists.

Return type list of *`mopidy.models.Playlist`*

Changed in version 1.0: If you call the method with `include_tracks=False`, the *`last_modified`* field of the returned playlists is no longer set.

Deprecated since version 1.0: Use *`as_list()`* and *`get_items()`* instead.

`PlaylistsController.`**`playlists`**

Deprecated since version 1.0: Use *`as_list()`* and *`get_items()`* instead.

32.2.2 Frontend API

The following requirements applies to any frontend implementation:

- A frontend MAY do mostly whatever it wants to, including creating threads, opening TCP ports and exposing Mopidy for a group of clients.
- A frontend MUST implement at least one Pykka actor, called the "main actor" from here on.
- The main actor MUST accept two constructor arguments:
  - `config`, which is a dict structure with the entire Mopidy configuration.
  - `core`, which will be an `ActorProxy` for the core actor. This object gives access to the full *mopidy.core — Core API*.
- It MAY use additional actors to implement whatever it does, and using actors in frontend implementations is encouraged.
- The frontend is enabled if the extension it is part of is enabled. See *Extension development* for more information.
- The main actor MUST be able to start and stop the frontend when the main actor is started and stopped.
- The frontend MAY require additional config values to be set for it to work.
- Such config values MUST be documented.
- The main actor MUST raise the `mopidy.exceptions.FrontendError` with a descriptive error message if the defined config values are not adequate for the frontend to work properly.
- Any actor which is part of the frontend MAY implement the *`mopidy.core.CoreListener`* interface to receive notification of the specified events.

Frontend implementations

See *Frontend extensions*.

32.2.3 `mopidy.backend` — Backend API

The backend API is the interface that must be implemented when you create a backend. If you are working on a frontend and need to access the backends, see the *mopidy.core — Core API* instead.

URIs and routing of requests to the backend

When Mopidy's core layer is processing a client request, it routes the request to one or more appropriate backends based on the URIs of the objects the request touches on. The objects' URIs are compared with the backends' `uri_schemes` to select the relevant backends.

An often used pattern when implementing Mopidy backends is to create your own URI scheme which you use for all tracks, playlists, etc. related to your backend. In most cases the Mopidy URI is translated to an actual URI that GStreamer knows how to play right before playback. For example:

- Spotify already has its own URI scheme (`spotify:track:...`, `spotify:playlist:...`, etc.) used throughout their applications, and thus Mopidy-Spotify simply uses the same URI scheme. Playback is handled by pushing raw audio data into a GStreamer `appsrc` element.
- Mopidy-SoundCloud created it's own URI scheme, after the model of Spotify, and use URIs of the following forms: `soundcloud:search`, `soundcloud:user-...`, `soundcloud:exp-...`, and `soundcloud:set-...`. Playback is handled by converting the custom `soundcloud:..` URIs to `http://` URIs immediately before they are passed on to GStreamer for playback.
- Mopidy differentiates between `file://...` URIs handled by *Mopidy-Stream* and `local:...` URIs handled by *Mopidy-Local*. *Mopidy-Stream* can play `file://...` URIs pointing to tracks and playlists located anywhere on your system, but it doesn't know a thing about the object before you play it. On the other hand, *Mopidy-Local* scans a predefined `local/media_dir` to build a meta data library of all known tracks. It is thus limited to playing tracks residing in the media library, but can provide additional features like directory browsing and search. In other words, we have two different ways of playing local music, handled by two different backends, and have thus created two different URI schemes to separate their handling. The `local:...` URIs are converted to `file://...` URIs immediately before they are passed on to GStreamer for playback.

If there isn't an existing URI scheme that fits for your backend's purpose, you should create your own, and name it after your extension's `ext_name`. Care should be taken not to conflict with already in use URI schemes. It is also recommended to design the format such that tracks, playlists and other entities can be distinguished easily.

Backend class

class mopidy.backend.**Backend**

Backend API

If the backend has problems during initialization it should raise `mopidy.exceptions.BackendError` with a descriptive error message. This will make Mopidy print the error message and exit so that the user can fix the issue.

Parameters

- **config** (*dict*) – the entire Mopidy configuration
- **audio** (`pykka.ActorProxy` for *mopidy.audio.Audio*) – actor proxy for the audio subsystem

audio = None
Actor proxy to an instance of *mopidy.audio.Audio*.

Should be passed to the backend constructor as the kwarg `audio`, which will then set this field.

library = None
The library provider. An instance of *LibraryProvider*, or `None` if the backend doesn't provide a library.

ping()
Called to check if the actor is still alive.

playback = None
The playback provider. An instance of *PlaybackProvider*, or `None` if the backend doesn't provide playback.

playlists = None
The playlists provider. An instance of *PlaylistsProvider*, or class:*None* if the backend doesn't provide playlists.

uri_schemes = []
List of URI schemes this backend can handle.

Playback provider

class mopidy.backend.**PlaybackProvider**(*audio*, *backend*)

Parameters

- **audio** (actor proxy to an instance of *mopidy.audio.Audio*) – the audio actor
- **backend** (*mopidy.backend.Backend*) – the backend

change_track(*track*)
Swith to provided track.

MAY be reimplemented by subclass.

It is unlikely it makes sense for any backends to override this. For most practical purposes it should be considered an internal call between backends and core that backend authors should not touch.

The default implementation will call *translate_uri()* which is what you want to implement.

Parameters track (*mopidy.models.Track*) – the track to play

Return type `True` if successful, else `False`

get_time_position()
Get the current time position in milliseconds.

MAY be reimplemented by subclass.

Return type int

pause()
Pause playback.

MAY be reimplemented by subclass.

Return type `True` if successful, else `False`

play()
Start playback.

MAY be reimplemented by subclass.

Return type `True` if successful, else `False`

`prepare_change`()
Indicate that an URI change is about to happen.

MAY be reimplemented by subclass.

It is extremely unlikely it makes sense for any backends to override this. For most practical purposes it should be considered an internal call between backends and core that backend authors should not touch.

`resume`()
Resume playback at the same time position playback was paused.

MAY be reimplemented by subclass.

Return type `True` if successful, else `False`

`seek`(*time_position*)
Seek to a given time position.

MAY be reimplemented by subclass.

Parameters **`time_position`** (*`int`*) – time position in milliseconds

Return type `True` if successful, else `False`

`stop`()
Stop playback.

MAY be reimplemented by subclass.

Should not be used for tracking if tracks have been played or when we are done playing them.

Return type `True` if successful, else `False`

`translate_uri`(*uri*)
Convert custom URI scheme to real playable URI.

MAY be reimplemented by subclass.

This is very likely the *only* thing you need to override as a backend author. Typically this is where you convert any Mopidy specific URI to a real URI and then return it. If you can't convert the URI just return `None`.

Parameters **`uri`** (*`string`*) – the URI to translate

Return type string or `None` if the URI could not be translated

Playlists provider

`class` `mopidy.backend.`**`PlaylistsProvider`**(*backend*)
A playlist provider exposes a collection of playlists, methods to create/change/delete playlists in this collection, and lookup of any playlist the backend knows about.

Parameters **`backend`** (*`mopidy.backend.Backend`* instance) – backend the controller is a part of

`as_list`()
Get a list of the currently available playlists.

Returns a list of *`Ref`* objects referring to the playlists. In other words, no information about the playlists' content is given.

Return type list of *`mopidy.models.Ref`*

New in version 1.0.

create(*name*)
Create a new empty playlist with the given name.

Returns a new playlist with the given name and an URI.

MUST be implemented by subclass.

Parameters **name** (*string*) – name of the new playlist

Return type *mopidy.models.Playlist*

delete(*uri*)
Delete playlist identified by the URI.

MUST be implemented by subclass.

Parameters **uri** (*string*) – URI of the playlist to delete

get_items(*uri*)
Get the items in a playlist specified by `uri`.

Returns a list of *Ref* objects referring to the playlist's items.

If a playlist with the given `uri` doesn't exist, it returns `None`.

Return type list of *mopidy.models.Ref*, or `None`

New in version 1.0.

lookup(*uri*)
Lookup playlist with given URI in both the set of playlists and in any other playlist source.

Returns the playlists or `None` if not found.

MUST be implemented by subclass.

Parameters **uri** (*string*) – playlist URI

Return type *mopidy.models.Playlist* or `None`

refresh()
Refresh the playlists in `playlists`.

MUST be implemented by subclass.

save(*playlist*)
Save the given playlist.

The playlist must have an `uri` attribute set. To create a new playlist with an URI, use *create()*.

Returns the saved playlist or `None` on failure.

MUST be implemented by subclass.

Parameters **playlist** (*mopidy.models.Playlist*) – the playlist to save

Return type *mopidy.models.Playlist* or `None`

Library provider

class mopidy.backend.**LibraryProvider**(*backend*)

Parameters **backend** (*mopidy.backend.Backend*) – backend the controller is a part of

browse (*uri*)
See `mopidy.core.LibraryController.browse()`.

If you implement this method, make sure to also set `root_directory`.

MAY be implemented by subclass.

get_distinct (*field*, *query=None*)
See `mopidy.core.LibraryController.get_distinct()`.

MAY be implemented by subclass.

Default implementation will simply return an empty set.

Note that backends should always return an empty set for unexpected field types.

get_images (*uris*)
See `mopidy.core.LibraryController.get_images()`.

MAY be implemented by subclass.

Default implementation will simply call lookup and try and use the album art for any tracks returned. Most extensions should replace this with something smarter or simply return an empty dictionary.

lookup (*uri*)
See `mopidy.core.LibraryController.lookup()`.

MUST be implemented by subclass.

refresh (*uri=None*)
See `mopidy.core.LibraryController.refresh()`.

MAY be implemented by subclass.

root_directory = None
`mopidy.models.Ref.directory` instance with a URI and name set representing the root of this library's browse tree. URIs must use one of the schemes supported by the backend, and name should be set to a human friendly value.

MUST be set by any class that implements `LibraryProvider.browse()`.

search (*query=None*, *uris=None*, *exact=False*)
See `mopidy.core.LibraryController.search()`.

MAY be implemented by subclass.

New in version 1.0: The `exact` param which replaces the old `find_exact`.

Backend listener

class mopidy.backend.**BackendListener**
Marker interface for recipients of events sent by the backend actors.

Any Pykka actor that mixes in this class will receive calls to the methods defined here when the corresponding events happen in a backend actor. This interface is used both for looking up what actors to notify of the events, and for providing default implementations for those listeners that are not interested in all events.

Normally, only the Core actor should mix in this class.

playlists_loaded ()
Called when playlists are loaded or refreshed.

MAY be implemented by actor.

static send(*event*, ***kwargs*)
Helper to allow calling of backend listener events

Backend implementations

See *Backend extensions*.

32.2.4 `mopidy.ext` – Extension API

If you want to learn how to make Mopidy extensions, read *Extension development*.

class `mopidy.ext.`**Extension**
Base class for Mopidy extensions

dist_name = None
The extension's distribution name, as registered on PyPI

Example: `Mopidy-Soundspot`

ext_name = None
The extension's short name, as used in setup.py and as config section name

Example: `soundspot`

classmethod get_cache_dir(*config*)
Get or create cache directory for the extension.

Use this directory to cache data that can safely be thrown away.

Parameters **config** – the Mopidy config object

Returns string

get_command()
Command to expose to command line users running `mopidy`.

Returns Instance of a *Command* class.

classmethod get_config_dir(*config*)
Get or create configuration directory for the extension.

Parameters **config** – the Mopidy config object

Returns string

get_config_schema()
The extension's config validation schema

Returns *ConfigSchema*

classmethod get_data_dir(*config*)
Get or create data directory for the extension.

Use this directory to store data that should be persistent.

Parameters **config** – the Mopidy config object

Returns string

get_default_config()
The extension's default config as a bytestring

Returns bytes or unicode

setup (*registry*)

Register the extension's components in the extension `Registry`.

For example, to register a backend:

```
def setup(self, registry):
    from .backend import SoundspotBackend
    registry.add('backend', SoundspotBackend)
```

See `Registry` for a list of registry keys with a special meaning. Mopidy will instantiate and start any classes registered under the `frontend` and `backend` registry keys.

This method can also be used for other setup tasks not involving the extension registry.

Parameters **registry** (`Registry`) – the extension registry

validate_environment ()

Checks if the extension can run in the current environment.

Dependencies described by `setup.py` are checked by Mopidy, so you should not check their presence here.

If a problem is found, raise `ExtensionError` with a message explaining the issue.

Raises `ExtensionError`

Returns `None`

version = None

The extension's version

Should match the `__version__` attribute on the extension's main Python module and the version registered on PyPI.

class mopidy.ext.**ExtensionData** (*extension*, *entry_point*, *config_schema*, *config_defaults*, *command*)

command

Alias for field number 4

config_defaults

Alias for field number 3

config_schema

Alias for field number 2

entry_point

Alias for field number 1

extension

Alias for field number 0

class mopidy.ext.**Registry**

Registry of components provided by Mopidy extensions.

Passed to the `setup()` method of all extensions. The registry can be used like a dict of string keys and lists.

Some keys have a special meaning, including, but not limited to:

- `backend` is used for Mopidy backend classes.
- `frontend` is used for Mopidy frontend classes.
- `local:library` is used for Mopidy-Local libraries.

Extensions can use the registry for allow other to extend the extension itself. For example the `Mopidy-Local` use the `local:library` key to allow other extensions to register library providers for `Mopidy-Local` to use. Extensions should namespace custom keys with the extension's *`ext_name`*, e.g. `local:foo` or `http:bar`.

`add`(*name*, *cls*)
Add a component to the registry.

Multiple classes can be registered to the same name.

`mopidy.ext.`**`load_extensions`**()
Find all installed extensions.

Returns list of installed extensions

`mopidy.ext.`**`validate_extension_data`**(*data*)
Verify extension's dependencies and environment.

Parameters **`extensions`** – an extension to check

Returns if extension should be run

32.3 Web/JavaScript

32.3.1 HTTP server side API

The *Mopidy-HTTP* extension comes with an HTTP server to host Mopidy's *HTTP JSON-RPC API*. This web server can also be used by other extensions that need to expose something over HTTP.

The HTTP server side API can be used to:

- host static files for e.g. a Mopidy client written in pure JavaScript,
- host a Tornado application, or
- host a WSGI application, including e.g. Flask applications.

To host static files using the web server, an extension needs to register a name and a file path in the extension registry under the `http:static` key.

To extend the web server with a web application, an extension must register a name and a factory function in the extension registry under the `http:app` key.

For details on how to make a Mopidy extension, see the *Extension development* guide.

Static web client example

To serve static files, you just need to register an `http:static` dictionary in the extension registry. The dictionary must have two keys: `name` and `path`. The `name` is used to build the URL the static files will be served on. By convention, it should be identical with the extension's *`ext_name`*, like in the following example. The `path` tells Mopidy where on the disk the static files are located.

Assuming that the code below is located in the file `mywebclient/__init__.py`, the files in the directory `mywebclient/static/` will be made available at `/mywebclient/` on Mopidy's web server. For example, `mywebclient/static/foo.html` will be available at http://localhost:6680/mywebclient/foo.html.

```
from __future__ import absolute_import, unicode_literals

import os

from mopidy import ext


class MyWebClientExtension(ext.Extension):
    ext_name = 'mywebclient'

    def setup(self, registry):
        registry.add('http:static', {
            'name': self.ext_name,
            'path': os.path.join(os.path.dirname(__file__), 'static'),
        })

    # See the Extension API for the full details on this class
```

Tornado application example

The *Mopidy-HTTP* extension's web server is based on the Tornado web framework. Thus, it has first class support for Tornado request handlers.

In the following example, we create a `tornado.web.RequestHandler` called `MyRequestHandler` that responds to HTTP GET requests with the string `Hello, world! This is Mopidy $version`, where it gets the Mopidy version from Mopidy's core API.

To hook the request handler into Mopidy's web server, we must register a dictionary under the `http:app` key in the extension registry. The dictionary must have two keys: `name` and `factory`.

The `name` is used to build the URL the app will be served on. By convention, it should be identical with the extension's *`ext_name`*, like in the following example.

The `factory` must be a function that accepts two arguments, `config` and `core`, respectively a dict structure of Mopidy's config and a `pykka.ActorProxy` to the full Mopidy core API. The `factory` function must return a list of Tornado request handlers. The URL patterns of the request handlers should not include the `name`, as that will be prepended to the URL patterns by the web server.

When the extension is installed, Mopidy will respond to requests to http://localhost:6680/mywebclient/ with the string `Hello, world! This is Mopidy $version`.

```
from __future__ import absolute_import, unicode_literals

import os

import tornado.web

from mopidy import ext


class MyRequestHandler(tornado.web.RequestHandler):
    def initialize(self, core):
        self.core = core

    def get(self):
        self.write(
            'Hello, world! This is Mopidy %s' %
```

(continues on next page)

(continued from previous page)

```
            self.core.get_version().get())


def my_app_factory(config, core):
    return [
        ('/', MyRequestHandler, {'core': core})
    ]


class MyWebClientExtension(ext.Extension):
    ext_name = 'mywebclient'

    def setup(self, registry):
        registry.add('http:app', {
            'name': self.ext_name,
            'factory': my_app_factory,
        })

    # See the Extension API for the full details on this class
```

WSGI application example

WSGI applications are second-class citizens on Mopidy's HTTP server. The WSGI applications are run inside Tornado, which is based on non-blocking I/O and a single event loop. In other words, your WSGI applications will only have a single thread to run on, and if your application is doing blocking I/O, it will block all other requests from being handled by the web server as well.

The example below shows how a WSGI application that returns the string `Hello, world! This is Mopidy $version` on all requests. The WSGI application is wrapped as a Tornado application and mounted at http://localhost:6680/mywebclient/.

```
from __future__ import absolute_import, unicode_literals

import os

import tornado.web
import tornado.wsgi

from mopidy import ext


def my_app_factory(config, core):

    def wsgi_app(environ, start_response):
        status = '200 OK'
        response_headers = [('Content-type', 'text/plain')]
        start_response(status, response_headers)
        return [
            'Hello, world! This is Mopidy %s\n' %
            self.core.get_version().get()
        ]

    return [
        ('(.*)', tornado.web.FallbackHandler, {
            'fallback': tornado.wsgi.WSGIContainer(wsgi_app),
```

(continues on next page)

(continued from previous page)

```
        }),
    ]


class MyWebClientExtension(ext.Extension):
    ext_name = 'mywebclient'

    def setup(self, registry):
        registry.add('http:app', {
            'name': self.ext_name,
            'factory': my_app_factory,
        })

    # See the Extension API for the full details on this class
```

API implementors

See *Web extensions*.

32.3.2 HTTP JSON-RPC API

The *Mopidy-HTTP* extension makes Mopidy's *mopidy.core — Core API* available using JSON-RPC over HTTP using HTTP POST and WebSockets. We also provide a JavaScript wrapper, called *Mopidy.js*, around the JSON-RPC over WebSocket API for use both from browsers and Node.js. The *Mopidy-API-Explorer* extension, can also be used to get you familiarized with HTTP based APIs.

HTTP POST API

The Mopidy web server accepts HTTP requests with the POST method to http://localhost:6680/mopidy/rpc, where the `localhost:6680` part will vary with your local setup. The HTTP POST endpoint gives you access to Mopidy's full core API, but does not give you notification on events. If you need to listen to events, you should probably use the WebSocket API instead.

Example usage from the command line:

```
$ curl -d '{"jsonrpc": "2.0", "id": 1, "method": "core.playback.get_state"}' http://
↪localhost:6680/mopidy/rpc
{"jsonrpc": "2.0", "id": 1, "result": "stopped"}
```

For details on the request and response format, see *JSON-RPC 2.0 messages*.

WebSocket API

The Mopidy web server exposes a WebSocket at http://localhost:6680/mopidy/ws, where the `localhost:6680` part will vary with your local setup. The WebSocket gives you access to Mopidy's full API and enables Mopidy to instantly push events to the client, as they happen.

On the WebSocket we send two different kind of messages: The client can send *JSON-RPC 2.0 requests*, and the server will respond with JSON-RPC 2.0 responses. In addition, the server will send *event messages* when something happens on the server. Both message types are encoded as JSON objects.

If you're using the API from JavaScript, either in the browser or in Node.js, you should use *Mopidy.js JavaScript library* which wraps the WebSocket API in a nice JavaScript API.

JSON-RPC 2.0 messages

JSON-RPC 2.0 messages can be recognized by checking for the key named `jsonrpc` with the string value `2.0`. For details on the messaging format, please refer to the JSON-RPC 2.0 spec.

All methods in the *mopidy.core — Core API* is made available through JSON-RPC calls over the WebSocket. For example, `mopidy.core.PlaybackController.play()` is available as the JSON-RPC method `core.playback.play`.

Example JSON-RPC request:

```
{"jsonrpc": "2.0", "id": 1, "method": "core.playback.get_current_track"}
```

Example JSON-RPC response:

```
{"jsonrpc": "2.0", "id": 1, "result": {"__model__": "Track", "...": "..."}}
```

The JSON-RPC method `core.describe` returns a data structure describing all available methods. If you're unsure how the core API maps to JSON-RPC, having a look at the `core.describe` response can be helpful.

Event messages

Event objects will always have a key named `event` whose value is the event type. Depending on the event type, the event may include additional fields for related data. The events maps directly to the `mopidy.core.CoreListener` API. Refer to the `CoreListener` method names is the available event types. The `CoreListener` method's keyword arguments are all included as extra fields on the event objects. Example event message:

```
{"event": "track_playback_started", "track": {...}}
```

32.3.3 Mopidy.js JavaScript library

We've made a JavaScript library, Mopidy.js, which wraps the *WebSocket API* and gets you quickly started with working on your client instead of figuring out how to communicate with Mopidy.

Getting the library for browser use

Regular and minified versions of Mopidy.js, ready for use, is installed together with Mopidy. When the HTTP extension is enabled, the files are available at:

- http://localhost:6680/mopidy/mopidy.js
- http://localhost:6680/mopidy/mopidy.min.js

You may need to adjust hostname and port for your local setup.

Thus, if you use Mopidy to host your web client, like described in *Static web client example*, you can load the latest version of Mopidy.js by adding the following script tag to your HTML file:

```
<script type="text/javascript" src="/mopidy/mopidy.min.js"></script>
```

If you don't use Mopidy to host your web client, you can find the JS files in the Git repo at:

- `mopidy/http/data/mopidy.js`
- `mopidy/http/data/mopidy.min.js`

Getting the library for Node.js or Browserify use

If you want to use Mopidy.js from Node.js or on the web through Browserify, you can install Mopidy.js using npm:

```
npm install mopidy
```

After npm completes, you can import Mopidy.js using `require()`:

```
var Mopidy = require("mopidy");
```

Getting the library for development on the library

If you want to work on the Mopidy.js library itself, you'll find the source code and a complete development setup in the Mopidy.js Git repo. The instructions in `README.md` will guide you on your way.

Creating an instance

Once you have Mopidy.js loaded, you need to create an instance of the wrapper:

```
var mopidy = new Mopidy();
```

When you instantiate `Mopidy()` without arguments, it will connect to the WebSocket at `/mopidy/ws/` on the current host. Thus, if you don't host your web client using Mopidy's web server, or if you use Mopidy.js from a Node.js environment, you'll need to pass the URL to the WebSocket end point:

```
var mopidy = new Mopidy({
    webSocketUrl: "ws://localhost:6680/mopidy/ws/"
});
```

It is also possible to create an instance first and connect to the WebSocket later:

```
var mopidy = new Mopidy({autoConnect: false});
// ... do other stuff, like hooking up events ...
mopidy.connect();
```

When creating an instance, you can specify the following settings:

`autoConnect` Whether or not to connect to the WebSocket on instance creation. Defaults to true.

`backoffDelayMin` The minimum number of milliseconds to wait after a connection error before we try to reconnect. For every failed attempt, the backoff delay is doubled until it reaches `backoffDelayMax`. Defaults to 1000.

`backoffDelayMax` The maximum number of milliseconds to wait after a connection error before we try to reconnect. Defaults to 64000.

`callingConvention` Which calling convention to use when calling methods.

If set to "by-position-only", methods expect to be called with positional arguments, like `mopidy.foo.bar(null, true, 2)`.

If set to "by-position-or-by-name", methods expect to be called either with an array of position arguments, like `mopidy.foo.bar([null, true, 2])`, or with an object of named arguments, like `mopidy.foo.bar({id: 2})`. The advantage of the "by-position-or-by-name" calling convention is that arguments with default values can be left out of the named argument object. Using named arguments also makes the code more readable, and more resistent to future API changes.

Note: For backwards compatibility, the default is "by-position-only". In the future, the default will change to "by-position-or-by-name". You should explicitly set this setting to your choice, so you won't be affected when the default changes.

New in version 0.19: (Mopidy.js 0.4)

`console` If set, this object will be used to log errors from Mopidy.js. This is mostly useful for testing Mopidy.js.

`webSocket` An existing WebSocket object to be used instead of creating a new WebSocket. Defaults to undefined.

`webSocketUrl` URL used when creating new WebSocket objects. Defaults to `ws://<document.location.host>/mopidy/ws`, or `ws://localhost/mopidy/ws` if `document.location.host` isn't available, like it is in the browser environment.

Hooking up to events

Once you have a Mopidy.js object, you can hook up to the events it emits. To explore your possibilities, it can be useful to subscribe to all events and log them:

```
mopidy.on(console.log.bind(console));
```

Several types of events are emitted:

- You can get notified about when the Mopidy.js object is connected to the server and ready for method calls, when it's offline, and when it's trying to reconnect to the server by looking at the events `state:online`, `state:offline`, `reconnectionPending`, and `reconnecting`.
- You can get events sent from the Mopidy server by looking at the events with the name prefix `event:`, like `event:trackPlaybackStarted`.
- You can introspect what happens internally on the WebSocket by looking at the events emitted with the name prefix `websocket:`.

Mopidy.js uses the event emitter library BANE, so you should refer to BANE's short API documentation to see how you can hook up your listeners to the different events.

Calling core API methods

Once your Mopidy.js object has connected to the Mopidy server and emits the `state:online` event, it is ready to accept core API method calls:

```
mopidy.on("state:online", function () {
    mopidy.playback.next();
});
```

Any calls you make before the `state:online` event is emitted will fail. If you've hooked up an errback (more on that a bit later) to the promise returned from the call, the errback will be called with a `Mopidy.ConnectionError` instance.

All methods in Mopidy's *mopidy.core — Core API* is available via Mopidy.js. For example, the *`mopidy.core.PlaybackController.get_state()`* method is available in JSON-RPC as the method `core.playback.get_state` and in Mopidy.js as `mopidy.playback.getState()`.

Both the WebSocket API and the JavaScript API are based on introspection of the core Python API. Thus, they will always be up to date and immediately reflect any changes we do to the core API.

The best way to explore the JavaScript API, is probably by opening your browser's console, and using its tab completion to navigate the API. You'll find the Mopidy core API exposed under `mopidy.playback`, `mopidy.tracklist`, `mopidy.playlists`, and `mopidy.library`.

All methods in the JavaScript API have an associated data structure describing the Python params it expects, and most methods also have the Python API documentation available. This is available right there in the browser console, by looking at the method's `description` and `params` attributes:

```
console.log(mopidy.playback.next.params);
console.log(mopidy.playback.next.description);
```

JSON-RPC 2.0 limits method parameters to be sent *either* by-position or by-name. Combinations of both, like we're used to from Python, isn't supported by JSON-RPC 2.0.

Obviously, you'll want to get a return value from many of your method calls. Since everything is happening across the WebSocket and maybe even across the network, you'll get the results asynchronously. Instead of having to pass callbacks and errbacks to every method you call, the methods return "promise" objects, which you can use to pipe the future result as input to another method, or to hook up callback and errback functions.

```
var track = mopidy.playback.getCurrentTrack();
// => ``track`` isn't a track, but a "promise" object
```

Instead, typical usage will look like this:

```
var printCurrentTrack = function (track) {
    if (track) {
        console.log("Currently playing:", track.name, "by",
            track.artists[0].name, "from", track.album.name);
    } else {
        console.log("No current track");
    }
};

mopidy.playback.getCurrentTrack()
    .done(printCurrentTrack);
```

The function passed to `done()`, `printCurrentTrack`, is the callback that will be called if the method call succeeds. If anything goes wrong, `done()` will throw an exception.

If you want to explicitly handle any errors and avoid an exception being thrown, you can register an error handler function anywhere in a promise chain. The function will be called with the error object as the only argument:

```
mopidy.playback.getCurrentTrack()
    .catch(console.error.bind(console))
    .done(printCurrentTrack);
```

You can also register the error handler at the end of the promise chain by passing it as the second argument to `done()`:

```
mopidy.playback.getCurrentTrack()
    .done(printCurrentTrack, console.error.bind(console));
```

If you don't hook up an error handler function and never call `done()` on the promise object, warnings will be logged to the console complaining that you have unhandled errors. In general, unhandled errors will not go silently missing.

The promise objects returned by Mopidy.js adheres to the CommonJS Promises/A standard. We use the implementation known as when.js, and reexport it as `Mopidy.when` so you don't have to duplicate the dependency. Please refer to when.js' documentation or the standard for further details on how to work with promise objects.

Cleaning up

If you for some reason want to clean up after Mopidy.js before the web page is closed or navigated away from, you can close the WebSocket, unregister all event listeners, and delete the object like this:

```
// Close the WebSocket without reconnecting. Letting the object be garbage
// collected will have the same effect, so this isn't strictly necessary.
mopidy.close();

// Unregister all event listeners. If you don't do this, you may have
// lingering references to the object causing the garbage collector to not
// clean up after it.
mopidy.off();

// Delete your reference to the object, so it can be garbage collected.
mopidy = null;
```

Example to get started with

1. Make sure that you've installed all dependencies required by *Mopidy-HTTP*.
2. Create an empty directory for your web client.
3. Change the `http/static_dir` config value to point to your new directory.
4. Start/restart Mopidy.
5. Create a file in the directory named `index.html` containing e.g. "Hello, world!".
6. Visit http://localhost:6680/ to confirm that you can view your new HTML file there.
7. Include Mopidy.js in your web page:

   ```
   <script type="text/javascript" src="/mopidy/mopidy.min.js"></script>
   ```

8. Add one of the following Mopidy.js examples of how to queue and start playback of your first playlist either to your web page or a JavaScript file that you include in your web page.

   "Imperative" style:

   ```
   var trackDesc = function (track) {
       return track.name + " by " + track.artists[0].name +
           " from " + track.album.name;
   };

   var queueAndPlay = function (playlistNum, trackNum) {
       playlistNum = playlistNum || 0;
       trackNum = trackNum || 0;
       mopidy.playlists.getPlaylists().then(function (playlists) {
           var playlist = playlists[playlistNum];
           console.log("Loading playlist:", playlist.name);
           return mopidy.tracklist.add(playlist.tracks).then(function (tlTracks) {
               return mopidy.playback.play(tlTracks[trackNum]).then(function () {
                   return mopidy.playback.getCurrentTrack().then(function (track) {
                       console.log("Now playing:", trackDesc(track));
                   });
               });
           });
       })
   ```

(continues on next page)

(continued from previous page)

```
    .catch(console.error.bind(console)) // Handle errors here
    .done();                            // ...or they'll be thrown here
};

var mopidy = new Mopidy();             // Connect to server
mopidy.on(console.log.bind(console));  // Log all events
mopidy.on("state:online", queueAndPlay);
```

Approximately the same behavior in a more functional style, using chaining of promises.

```
var get = function (key, object) {
    return object[key];
};

var printTypeAndName = function (model) {
    console.log(model.__model__ + ": " + model.name);
    // By returning the playlist, this function can be inserted
    // anywhere a model with a name is piped in the chain.
    return model;
};

var trackDesc = function (track) {
    return track.name + " by " + track.artists[0].name +
        " from " + track.album.name;
};

var printNowPlaying = function () {
    // By returning any arguments we get, the function can be inserted
    // anywhere in the chain.
    var args = arguments;
    return mopidy.playback.getCurrentTrack()
        .then(function (track) {
            console.log("Now playing:", trackDesc(track));
            return args;
        });
};

var queueAndPlay = function (playlistNum, trackNum) {
    playlistNum = playlistNum || 0;
    trackNum = trackNum || 0;
    mopidy.playlists.getPlaylists()
        // => list of Playlists
        .fold(get, playlistNum)
        // => Playlist
        .then(printTypeAndName)
        // => Playlist
        .fold(get, 'tracks')
        // => list of Tracks
        .then(mopidy.tracklist.add)
        // => list of TlTracks
        .fold(get, trackNum)
        // => TlTrack
        .then(mopidy.playback.play)
        // => null
        .then(printNowPlaying)
        // => null
        .catch(console.error.bind(console))  // Handle errors here
```

(continues on next page)

(continued from previous page)

```
        // => null
        .done();                        // ...or they'll be thrown here
};

var mopidy = new Mopidy();              // Connect to server
mopidy.on(console.log.bind(console));   // Log all events
mopidy.on("state:online", queueAndPlay);
```

9. The web page should now queue and play your first playlist every time you load it. See the browser's console for output from the function, any errors, and all events that are emitted.

32.4 Audio

32.4.1 `mopidy.audio` — Audio API

The audio API is the interface we have built around GStreamer to support our specific use cases. Most backends should be able to get by with simply setting the URI of the resource they want to play, for these cases the default playback provider should be used.

For more advanced cases such as when the raw audio data is delivered outside of GStreamer or the backend needs to add metadata to the currently playing resource, developers should sub-class the base playback provider and implement the extra behaviour that is needed through the following API:

class `mopidy.audio.`**`Audio`**(*config*, *mixer*)
Audio output through GStreamer.

`emit_data`(*buffer_*)
Call this to deliver raw audio data to be played.

If the buffer is `None`, the end-of-stream token is put on the playbin. We will get a GStreamer message when the stream playback reaches the token, and can then do any end-of-stream related tasks.

Note that the URI must be set to `appsrc://` for this to work.

Returns `True` if data was delivered.

Parameters **buffer** (`gst.Buffer` or `None`) – buffer to pass to appsrc

Return type boolean

`emit_end_of_stream`()
Put an end-of-stream token on the playbin. This is typically used in combination with `emit_data()`.

We will get a GStreamer message when the stream playback reaches the token, and can then do any end-of-stream related tasks.

Deprecated since version 1.0: Use `emit_data()` with a `None` buffer instead.

`enable_sync_handler`()
Enable manual processing of messages from bus.

Should only be used by tests.

`get_current_tags`()
Get the currently playing media's tags.

If no tags have been found, or nothing is playing this returns an empty dictionary. For each set of tags we collect a tags_changed event is emitted with the keys of the changes tags. After such calls users may call this function to get the updated values.

Return type {key: [values]} dict for the current media.

get_position()
Get position in milliseconds.

Return type int

mixer = None
The software mixing interface `mopidy.audio.actor.SoftwareMixer`

pause_playback()
Notify GStreamer that it should pause playback.

Return type `True` if successfull, else `False`

prepare_change()
Notify GStreamer that we are about to change state of playback.

This function *MUST* be called before changing URIs or doing changes like updating data that is being pushed. The reason for this is that GStreamer will reset all its state when it changes to `gst.STATE_READY`.

set_about_to_finish_callback(*callback*)
Configure audio to use an about-to-finish callback.

This should be used to achieve gapless playback. For this to work the callback *MUST* call `set_uri()` with the new URI to play and block until this call has been made. `prepare_change()` is not needed before `set_uri()` in this one special case.

Parameters **callback** (*callable*) – Callback to run when we need the next URI.

set_appsrc(*caps*, *need_data=None*, *enough_data=None*, *seek_data=None*)
Switch to using appsrc for getting audio to be played.

You *MUST* call `prepare_change()` before calling this method.

Parameters

- **caps** (*string*) – GStreamer caps string describing the audio format to expect
- **need_data** (*callable which takes data length hint in ms*) – callback for when appsrc needs data
- **enough_data** (*callable*) – callback for when appsrc has enough data
- **seek_data** (*callable which takes time position in ms*) – callback for when data from a new position is needed to continue playback

set_metadata(*track*)
Set track metadata for currently playing song.

Only needs to be called by sources such as *appsrc* which do not already inject tags in playbin, e.g. when using `emit_data()` to deliver raw audio data to GStreamer.

Parameters **track** (*mopidy.models.Track*) – the current track

set_position(*position*)
Set position in milliseconds.

Parameters **position** (*int*) – the position in milliseconds

Return type `True` if successful, else `False`

set_uri(*uri*)
Set URI of audio to be played.

You *MUST* call `prepare_change()` before calling this method.

Parameters **uri** (*string*) – the URI to play

start_playback()
Notify GStreamer that it should start playback.

Return type `True` if successfull, else `False`

state = u'stopped'
The GStreamer state mapped to `mopidy.audio.PlaybackState`

stop_playback()
Notify GStreamer that is should stop playback.

Return type `True` if successfull, else `False`

wait_for_state_change()
Block until any pending state changes are complete.

Should only be used by tests.

Audio listener

class mopidy.audio.**AudioListener**
Marker interface for recipients of events sent by the audio actor.

Any Pykka actor that mixes in this class will receive calls to the methods defined here when the corresponding events happen in the core actor. This interface is used both for looking up what actors to notify of the events, and for providing default implementations for those listeners that are not interested in all events.

position_changed(*position*)
Called whenever the position of the stream changes.

MAY be implemented by actor.

Parameters **position** (*int*) – Position in milliseconds.

reached_end_of_stream()
Called whenever the end of the audio stream is reached.

MAY be implemented by actor.

static send(*event*, ***kwargs*)
Helper to allow calling of audio listener events

state_changed(*old_state*, *new_state*, *target_state*)
Called after the playback state have changed.

Will be called for both immediate and async state changes in GStreamer.

Target state is used to when we should be in the target state, but temporarily need to switch to an other state. A typical example of this is buffering. When this happens an event with *old=PLAYING, new=PAUSED, target=PLAYING* will be emitted. Once we have caught up a *old=PAUSED, new=PLAYING, target=None* event will be be generated.

Regular state changes will not have target state set as they are final states which should be stable.

MAY be implemented by actor.

Parameters

- **old_state** (string from `mopidy.core.PlaybackState` field) – the state before the change

- **new_state** (string from `mopidy.core.PlaybackState` field) – the state after the change
- **target_state** (string from `mopidy.core.PlaybackState` field or `None` if this is a final state.) – the intended state

stream_changed(*uri*)
Called whenever the audio stream changes.

MAY be implemented by actor.

Parameters **uri** (*string*) – URI the stream has started playing.

tags_changed(*tags*)
Called whenever the current audio stream's tags change.

This event signals that some track metadata has been updated. This can be metadata such as artists, titles, organization, or details about the actual audio such as bit-rates, numbers of channels etc.

For the available tag keys please refer to GStreamer documentation for tags.

MAY be implemented by actor.

Parameters **tags** (`set` of strings) – The tags that have just been updated.

Audio scanner

class mopidy.audio.scan.**Scanner**(*timeout=1000*, *proxy_config=None*)
Helper to get tags and other relevant info from URIs.

Parameters

- **timeout** – timeout for scanning a URI in ms
- **proxy_config** – dictionary containing proxy config strings.

scan(*uri*, *timeout=None*)
Scan the given uri collecting relevant metadata.

Parameters

- **uri** (*string*) – URI of the resource to scan.
- **timeout** (*int*) – timeout for scanning a URI in ms. Defaults to the `timeout` value used when creating the scanner.

Returns A named tuple containing `(uri, tags, duration, seekable, mime)`. `tags` is a dictionary of lists for all the tags we found. `duration` is the length of the URI in milliseconds, or `None` if the URI has no duration. `seekable` is boolean. indicating if a seek would succeed.

Audio utils

mopidy.audio.utils.**calculate_duration**(*num_samples*, *sample_rate*)
Determine duration of samples using GStreamer helper for precise math.

mopidy.audio.utils.**clocktime_to_millisecond**(*value*)
Convert an internal GStreamer time to millisecond time.

mopidy.audio.utils.**convert_taglist**(*taglist*)
Convert a `gst.Taglist` to plain Python types.

Knows how to convert:

- Dates
- Buffers
- Numbers
- Strings
- Booleans

Unknown types will be ignored and debug logged. Tag keys are all strings defined as part GStreamer under GstTagList.

Parameters **`taglist`** (`gst.Taglist`) – A GStreamer taglist to be converted.

Return type dictionary of tag keys with a list of values.

`mopidy.audio.utils.`**`convert_tags_to_track`**(*tags*)
Convert our normalized tags to a track.

Parameters **`tags`** (`dict`) – dictionary of tag keys with a list of values

Return type *`mopidy.models.Track`*

`mopidy.audio.utils.`**`create_buffer`**(*data*, *capabilites=None*, *timestamp=None*, *duration=None*)
Create a new GStreamer buffer based on provided data.

Mainly intended to keep gst imports out of non-audio modules.

`mopidy.audio.utils.`**`millisecond_to_clocktime`**(*value*)
Convert a millisecond time to internal GStreamer time.

`mopidy.audio.utils.`**`setup_proxy`**(*element*, *config*)
Configure a GStreamer element with proxy settings.

Parameters

- **`element`** (`gst.GstElement`) – element to setup proxy in.
- **`config`** (`dict`) – proxy settings to use.

`mopidy.audio.utils.`**`supported_uri_schemes`**(*uri_schemes*)
Determine which URIs we can actually support from provided whitelist.

Parameters **`uri_schemes`** (*`list or set or URI schemes as strings.`*) – list/set of URIs to check support for.

Return type set of URI schemes we can support via this GStreamer install.

32.4.2 `mopidy.mixer` — Audio mixer API

`class` `mopidy.mixer.`**`Mixer`**(*config*)
Audio mixer API

If the mixer has problems during initialization it should raise `mopidy.exceptions.MixerError` with a descriptive error message. This will make Mopidy print the error message and exit so that the user can fix the issue.

Parameters **`config`** (*`dict`*) – the entire Mopidy configuration

`get_mute`()
Get mute state of the mixer.

MAY be implemented by subclass.

Return type `True` if muted, `False` if unmuted, `None` if unknown.

get_volume()

Get volume level of the mixer on a linear scale from 0 to 100.

Example values:

0: Minimum volume, usually silent.

100: Maximum volume.

None: Volume is unknown.

MAY be implemented by subclass.

Return type int in range [0..100] or `None`

name = None

Name of the mixer.

Used when configuring what mixer to use. Should match the `ext_name` of the extension providing the mixer.

ping()

Called to check if the actor is still alive.

set_mute(*mute*)

Mute or unmute the mixer.

MAY be implemented by subclass.

Parameters **mute** (`bool`) – `True` to mute, `False` to unmute

Return type `True` if success, `False` if failure

set_volume(*volume*)

Set volume level of the mixer.

MAY be implemented by subclass.

Parameters **volume** (`int`) – Volume in the range [0..100]

Return type `True` if success, `False` if failure

trigger_mute_changed(*mute*)

Send `mute_changed` event to all mixer listeners.

This method should be called by subclasses when the mute state is changed, either because of a call to `set_mute()` or because of any external entity changing the mute state.

trigger_volume_changed(*volume*)

Send `volume_changed` event to all mixer listeners.

This method should be called by subclasses when the volume is changed, either because of a call to `set_volume()` or because of any external entity changing the volume.

class mopidy.mixer.**MixerListener**

Marker interface for recipients of events sent by the mixer actor.

Any Pykka actor that mixes in this class will receive calls to the methods defined here when the corresponding events happen in the mixer actor. This interface is used both for looking up what actors to notify of the events, and for providing default implementations for those listeners that are not interested in all events.

mute_changed(*mute*)

Called after the mute state has changed.

MAY be implemented by actor.

Parameters **mute** (`bool`) – `True` if muted, `False` if not muted

static send(*event*, ***kwargs*)
Helper to allow calling of mixer listener events

volume_changed(*volume*)
Called after the volume has changed.

MAY be implemented by actor.

Parameters **volume** (`int in range [0..100]`) – the new volume

Mixer implementations

See *Mixer extensions*.

32.5 Utilities

32.5.1 `mopidy.commands` — Commands API

class mopidy.commands.**Command**
Command parser and runner for building trees of commands.

This class provides a wraper around `argparse.ArgumentParser` for handling this type of command line application in a better way than argprases own sub-parser handling.

add_argument(**args*, ***kwargs*)
Add an argument to the parser.

This method takes all the same arguments as the `argparse.ArgumentParser` version of this method.

add_child(*name*, *command*)
Add a child parser to consider using.

Parameters **name** (`string`) – name to use for the sub-command that is being added.

exit(*status_code=0*, *message=None*, *usage=None*)
Optionally print a message and exit.

format_help(*prog=None*)
Format help for current parser and children.

format_usage(*prog=None*)
Format usage for current parser.

parse(*args*, *prog=None*)
Parse command line arguments.

Will recursively parse commands until a final parser is found or an error occurs. In the case of errors we will print a message and exit. Otherwise, any overrides are applied and the current parser stored in the command attribute of the return value.

Parameters

- **args** (`list of strings`) – list of arguments to parse
- **prog** (`string`) – name to use for program

Return type `argparse.Namespace`

run(**args*, ***kwargs*)
Run the command.

Must be implemented by sub-classes that are not simply an intermediate in the command namespace.

set(***kwargs*)
Override a value in the finaly result of parsing.

32.5.2 `mopidy.config` — Config API

class `mopidy.config.`**Proxy**(*data*)

`mopidy.config.`**read**(*config_file*)
Helper to load config defaults in same way across core and extensions

Config section schemas

class `mopidy.config.schemas.`**ConfigSchema**(*name*)
Logical group of config values that correspond to a config section.

Schemas are set up by assigning config keys with config values to instances. Once setup `deserialize()` can be called with a dict of values to process. For convienience we also support `format()` method that can used for converting the values to a dict that can be printed and `serialize()` for converting the values to a form suitable for persistence.

deserialize(*values*)
Validates the given `values` using the config schema.

Returns a tuple with cleaned values and errors.

serialize(*values*, *display=False*)
Converts the given `values` to a format suitable for persistence.

If `display` is `True` secret config values, like passwords, will be masked out.

Returns a dict of config keys and values.

class `mopidy.config.schemas.`**MapConfigSchema**(*name*, *value_type*)
Schema for handling multiple unknown keys with the same type.

Does not sub-class `ConfigSchema`, but implements the same serialize/deserialize interface.

Config value types

class `mopidy.config.types.`**Boolean**(*optional=False*)
Boolean value.

Accepts `1`, `yes`, `true`, and `on` with any casing as `True`.

Accepts `0`, `no`, `false`, and `off` with any casing as `False`.

class `mopidy.config.types.`**ConfigValue**
Represents a config key's value and how to handle it.

Normally you will only be interacting with sub-classes for config values that encode either deserialization behavior and/or validation.

Each config value should be used for the following actions:

1. Deserializing from a raw string and validating, raising ValueError on failure.

2. Serializing a value back to a string that can be stored in a config.
3. Formatting a value to a printable form (useful for masking secrets).

`None` values should not be deserialized, serialized or formatted, the code interacting with the config should simply skip None config values.

`deserialize`(*value*)
Cast raw string to appropriate type.

`serialize`(*value*, *display=False*)
Convert value back to string for saving.

class `mopidy.config.types.`**`Deprecated`**
Deprecated value

Used for ignoring old config values that are no longer in use, but should not cause the config parser to crash.

class `mopidy.config.types.`**`ExpandedPath`**(*original*, *expanded*)

class `mopidy.config.types.`**`Hostname`**(*optional=False*)
Network hostname value.

class `mopidy.config.types.`**`Integer`**(*minimum=None*, *maximum=None*, *choices=None*, *optional=False*)
Integer value.

class `mopidy.config.types.`**`List`**(*optional=False*)
List value.

Supports elements split by commas or newlines. Newlines take presedence and empty list items will be filtered out.

class `mopidy.config.types.`**`LogLevel`**
Log level value.

Expects one of `critical`, `error`, `warning`, `info`, `debug`, or `all`, with any casing.

class `mopidy.config.types.`**`Path`**(*optional=False*)
File system path

The following expansions of the path will be done:

- ~ to the current user's home directory
- `$XDG_CACHE_DIR` according to the XDG spec
- `$XDG_CONFIG_DIR` according to the XDG spec
- `$XDG_DATA_DIR` according to the XDG spec
- `$XDG_MUSIC_DIR` according to the XDG spec

class `mopidy.config.types.`**`Port`**(*choices=None*, *optional=False*)
Network port value.

Expects integer in the range 0-65535, zero tells the kernel to simply allocate a port for us.

class `mopidy.config.types.`**`Secret`**(*optional=False*, *choices=None*)
Secret string value.

Is decoded as utf-8 and n t escapes should work and be preserved.

Should be used for passwords, auth tokens etc. Will mask value when being displayed.

class `mopidy.config.types.`**`String`**(*optional=False*, *choices=None*)

String value.

Is decoded as utf-8 and n t escapes should work and be preserved.

Config value validators

`mopidy.config.validators.`**`validate_choice`**(*value*, *choices*)

Validate that `value` is one of the `choices`

Normally called in `deserialize()`.

`mopidy.config.validators.`**`validate_maximum`**(*value*, *maximum*)

Validate that `value` is at most `maximum`

Normally called in `deserialize()`.

`mopidy.config.validators.`**`validate_minimum`**(*value*, *minimum*)

Validate that `value` is at least `minimum`

Normally called in `deserialize()`.

`mopidy.config.validators.`**`validate_required`**(*value*, *required*)

Validate that `value` is set if `required`

Normally called in `deserialize()` on the raw string, _not_ the converted value.

32.5.3 `mopidy.httpclient` — HTTP Client helpers

`mopidy.httpclient.`**`format_proxy`**(*proxy_config*, *auth=True*)

Convert a Mopidy proxy config to the commonly used proxy string format.

Outputs `scheme://host:port`, `scheme://user:pass@host:port` or `None` depending on the proxy config provided.

You can also opt out of getting the basic auth by setting `auth` to `False`.

New in version 1.1.

`mopidy.httpclient.`**`format_user_agent`**(*name=None*)

Construct a User-Agent suitable for use in client code.

This will identify use by the provided `name` (which should be on the format `dist_name/version`), Mopidy version and Python version.

New in version 1.1.

32.5.4 `mopidy.zeroconf` — Zeroconf API

class `mopidy.zeroconf.`**`Zeroconf`**(*name*, *port*, *stype=None*, *domain=None*, *text=None*)

Publish a network service with Zeroconf.

Currently, this only works on Linux using Avahi via D-Bus.

Parameters

- **name** (`str`) – human readable name of the service, e.g. 'MPD on neptune'
- **port** (`int`) – TCP port of the service, e.g. 6600
- **stype** (`str`) – service type, e.g. '_mpd._tcp'

- **domain** (*str*) – local network domain name, defaults to ‘’
- **host** (*str*) – interface to advertise the service on, defaults to all interfaces
- **text** (*list of str*) – extra information depending on `stype`, defaults to empty list

publish()
: Publish the service.

  Call when your service starts.

unpublish()
: Unpublish the service.

  Call when your service shuts down.

CHAPTER 33

Module reference

33.1 `mopidy.local` — Local backend

For details on how to use Mopidy's local backend, see *Mopidy-Local*.

33.1.1 Local library API

class `mopidy.local.`**`Library`**(*config*)

Local library interface.

Extensions that wish to provide an alternate local library storage backend need to sub-class this class and install and configure it with an extension. Both scanning and library calls will use the active local library.

Parameters **`config`** – Config dictionary

`ROOT_DIRECTORY_URI = u'local:directory'`

URI of the local backend's root directory.

This constant should be used by libraries implementing the `Library.browse()` method.

`add`(*track*, *tags=None*, *duration=None*)

Add the given track to library. Optional args will only be added if `add_supports_tags_and_duration` has been set.

Parameters

- **`track`** (`Track`) – Track to add to the library
- **`tags`** (`dictionary of tag keys with a list of values.`) – All the tags the scanner found for the media. See `mopidy.audio.utils` for details about the tags.
- **`duration`** (`int` or `None`) – Duration of media in milliseconds or `None` if unknown

`add_supports_tags_and_duration = False`

Feature marker to indicate that you want `add()` calls to be called with optional arguments tags and duration.

begin()

Prepare library for accepting updates. Exactly what this means is highly implementation depended. This must however return an iterator that generates all tracks in the library for efficient scanning.

Return type *Track* iterator

browse(*uri*)

Browse directories and tracks at the given URI.

The URI for the root directory is a constant available at *Library.ROOT_DIRECTORY_URI*.

Parameters **path** (*string*) – URI to browse.

Return type List of *Ref* tracks and directories.

clear()

Clear out whatever data storage is used by this backend.

Return type Boolean indicating if state was cleared.

close()

Close any resources used for updating, commit outstanding work etc.

flush()

Called for every n-th track indicating that work should be committed. Sub-classes are free to ignore these hints.

Return type Boolean indicating if state was flushed.

get_distinct(*field*, *query=None*)

List distinct values for a given field from the library.

Parameters

- **field** (*string*) – One of `artist`, `albumartist`, `album`, `composer`, `performer`, ``date``or ``genre``.
- **query** (*dict*) – Query to use for limiting results, see *search()* for details about the query format.

Return type set of values corresponding to the requested field type.

get_images(*uris*)

Lookup the images for the given URIs.

The default implementation will simply call *lookup()* and try and use the album art for any tracks returned. Most local libraries should replace this with something smarter or simply return an empty dictionary.

Parameters **uris** (*list*) – list of URIs to find images for

Return type {uri: tuple of *mopidy.models.Image*}

load()

(Re)load any tracks stored in memory, if any, otherwise just return number of available tracks currently available. Will be called at startup for both library and update use cases, so if you plan to store tracks in memory this is when the should be (re)loaded.

Return type `int` representing number of tracks in library.

lookup(*uri*)

Lookup the given URI.

Parameters **uri** (*string*) – track URI

Return type list of *Track* (or single *Track* for backward compatibility)

name = None
Name of the local library implementation, must be overriden.

remove(*uri*)
Remove the given track from the library.

Parameters **uri** (*str*) – URI to remove from the library/

search(*query=None*, *limit=100*, *offset=0*, *exact=False*, *uris=None*)
Search the library for tracks where `field` contains `values`.

Parameters

- **query** (*dict*) – one or more queries to search for
- **limit** (*int*) – maximum number of results to return
- **offset** (*int*) – offset into result set to use.
- **exact** (*bool*) – whether to look for exact matches
- **uris** (list of strings or `None`) – zero or more URI roots to limit the search to

Return type *SearchResult*

33.1.2 Translation utils

`mopidy.local.translator.`**local_uri_to_file_uri**(*uri*, *media_dir*)
Convert local track or directory URI to file URI.

`mopidy.local.translator.`**local_uri_to_path**(*uri*, *media_dir*)
Convert local track or directory URI to absolute path.

`mopidy.local.translator.`**path_to_file_uri**(*abspath*)
Convert absolute path to file URI.

`mopidy.local.translator.`**path_to_local_directory_uri**(*relpath*)
Convert path relative to *local/media_dir* directory URI.

`mopidy.local.translator.`**path_to_local_track_uri**(*relpath*)
Convert path relative to *local/media_dir* to local track URI.

33.2 `mopidy.mpd` — MPD server

For details on how to use Mopidy's MPD server, see *Mopidy-MPD*.

33.2.1 MPD tokenizer

`mopidy.mpd.tokenize.`**split**(*line*)
Splits a line into tokens using same rules as MPD.

- Lines may not start with whitespace
- Tokens are split by arbitrary amount of spaces or tabs
- First token must match *[a-z][a-z0-9_]**
- Remaining tokens can be unquoted or quoted tokens.
- Unquoted tokens consist of all printable characters except double quotes, single quotes, spaces and tabs.

- Quoted tokens are surrounded by a matching pair of double quotes.
- The closing quote must be followed by space, tab or end of line.
- Any value is allowed inside a quoted token. Including double quotes, assuming it is correctly escaped.
- Backslash inside a quoted token is used to escape the following character.

For examples see the tests for this function.

33.2.2 MPD dispatcher

class `mopidy.mpd.dispatcher.`**`MpdContext`**(*dispatcher*, *session=None*, *config=None*, *core=None*, *uri_map=None*)

This object is passed as the first argument to all MPD command handlers to give the command handlers access to important parts of Mopidy.

`browse`(*path*, *recursive=True*, *lookup=True*)

Browse the contents of a given directory path.

Returns a sequence of two-tuples `(path, data)`.

If `recursive` is true, it returns results for all entries in the given path.

If `lookup` is true and the `path` is to a track, the returned `data` is a future which will contain the results from looking up the URI with *`mopidy.core.LibraryController.lookup()`*. If `lookup` is false and the `path` is to a track, the returned `data` will be a *`mopidy.models.Ref`* for the track.

For all entries that are not tracks, the returned `data` will be `None`.

`core = None`

The Mopidy core API. An instance of *`mopidy.core.Core`*.

`dispatcher = None`

The current *`MpdDispatcher`*.

`events = None`

The active subsystems that have pending events.

`lookup_playlist_name_from_uri`(*uri*)

Helper function to retrieve the unique MPD playlist name from its uri.

`lookup_playlist_uri_from_name`(*name*)

Helper function to retrieve a playlist from its unique MPD name.

`password = None`

The MPD password

`session = None`

The current `mopidy.mpd.MpdSession`.

`subscriptions = None`

The subsytems that we want to be notified about in idle mode.

class `mopidy.mpd.dispatcher.`**`MpdDispatcher`**(*session=None*, *config=None*, *core=None*, *uri_map=None*)

The MPD session feeds the MPD dispatcher with requests. The dispatcher finds the correct handler, processes the request and sends the response back to the MPD session.

`handle_request`(*request*, *current_command_list_index=None*)

Dispatch incoming requests to the correct handler.

33.2.3 MPD protocol

rather incomplete with regards to data formats, both for requests and responses. Thus, we have had to talk a great deal with the the original MPD server using telnet to get the details we need to implement our own MPD server which is compatible with the numerous existing MPD clients.

`mopidy.mpd.protocol.`**`BOOL`**`(`*value*`)`
Convert the values 0 and 1 into booleans.

`class` `mopidy.mpd.protocol.`**`Commands`**
Collection of MPD commands to expose to users.

Normally used through the global instance which command handlers have been installed into.

`add``(`*name*, *auth_required=True*, *list_command=True*, ***validators*`)`
Create a decorator that registers a handler and validation rules.

Additional keyword arguments are treated as converters/validators to apply to tokens converting them to proper Python types.

Requirements for valid handlers:

- must accept a context argument as the first arg.
- may not use variable keyword arguments, `**kwargs`.
- may use variable arguments `*args` *or* a mix of required and optional arguments.

Decorator returns the unwrapped function so that tests etc can use the functions with values with correct python types instead of strings.

Parameters

- **`name`** (*`string`*) – Name of the command being registered.
- **`auth_required`** (*`bool`*) – If authorization is required.
- **`list_command`** (*`bool`*) – If command should be listed in reflection.

`call``(`*tokens*, *context=None*`)`
Find and run the handler registered for the given command.

If the handler was registered with any converters/validators they will be run before calling the real handler.

Parameters

- **`tokens`** (*`list`*) – List of tokens to process
- **`context`** (*`MpdContext`*) – MPD context.

`mopidy.mpd.protocol.`**`ENCODING = u'UTF-8'`**
The MPD protocol uses UTF-8 for encoding all data.

`mopidy.mpd.protocol.`**`INT`**`(`*value*`)`
Converts a value that matches [+-]?d+ into and integer.

`mopidy.mpd.protocol.`**`LINE_TERMINATOR = u'\n'`**
The MPD protocol uses `\n` as line terminator.

`mopidy.mpd.protocol.`**`RANGE`**`(`*value*`)`
Convert a single integer or range spec into a slice

`n` should become `slice(n, n+1)` `n:` should become `slice(n, None)` `n:m` should become `slice(n, m)` and `m > n` must hold

`mopidy.mpd.protocol.`**`UINT`**`(`*value*`)`
Converts a value that matches d+ into an integer.

`mopidy.mpd.protocol.`**`VERSION = u'0.19.0'`**
The MPD protocol version is 0.19.0.

`mopidy.mpd.protocol.`**`commands = <mopidy.mpd.protocol.Commands object>`**
Global instance to install commands into

`mopidy.mpd.protocol.`**`load_protocol_modules`**`()`
The protocol modules must be imported to get them registered in `commands`.

Audio output

`mopidy.mpd.protocol.audio_output.`**`disableoutput`**`(`*context*, *outputid*`)`
musicpd.org, audio output section:

```
disableoutput {ID}
```

Turns an output off.

`mopidy.mpd.protocol.audio_output.`**`enableoutput`**`(`*context*, *outputid*`)`
musicpd.org, audio output section:

```
enableoutput {ID}
```

Turns an output on.

`mopidy.mpd.protocol.audio_output.`**`outputs`**`(`*context*`)`
musicpd.org, audio output section:

```
outputs
```

Shows information about all outputs.

`mopidy.mpd.protocol.audio_output.`**`toggleoutput`**`(`*context*, *outputid*`)`
musicpd.org, audio output section:

```
toggleoutput {ID}
```

Turns an output on or off, depending on the current state.

Channels

`mopidy.mpd.protocol.channels.`**`channels`**`(`*context*`)`
musicpd.org, client to client section:

```
channels
```

Obtain a list of all channels. The response is a list of “channel:” lines.

`mopidy.mpd.protocol.channels.`**`readmessages`**`(`*context*`)`
musicpd.org, client to client section:

```
readmessages
```

Reads messages for this client. The response is a list of “channel:” and “message:” lines.

`mopidy.mpd.protocol.channels.`**`sendmessage`**`(`*context*, *channel*, *text*`)`
musicpd.org, client to client section:

```
sendmessage {CHANNEL} {TEXT}
```

Send a message to the specified channel.

`mopidy.mpd.protocol.channels.`**`subscribe`**`(`*context*, *channel*`)`

musicpd.org, client to client section:

```
subscribe {NAME}
```

Subscribe to a channel. The channel is created if it does not exist already. The name may consist of alphanumeric ASCII characters plus underscore, dash, dot and colon.

`mopidy.mpd.protocol.channels.`**`unsubscribe`**`(`*context*, *channel*`)`

musicpd.org, client to client section:

```
unsubscribe {NAME}
```

Unsubscribe from a channel.

Command list

`mopidy.mpd.protocol.command_list.`**`command_list_begin`**`(`*context*`)`

musicpd.org, command list section:

To facilitate faster adding of files etc. you can pass a list of commands all at once using a command list. The command list begins with `command_list_begin` or `command_list_ok_begin` and ends with `command_list_end`.

It does not execute any commands until the list has ended. The return value is whatever the return for a list of commands is. On success for all commands, `OK` is returned. If a command fails, no more commands are executed and the appropriate `ACK` error is returned. If `command_list_ok_begin` is used, `list_OK` is returned for each successful command executed in the command list.

`mopidy.mpd.protocol.command_list.`**`command_list_end`**`(`*context*`)`

See *`command_list_begin()`*.

`mopidy.mpd.protocol.command_list.`**`command_list_ok_begin`**`(`*context*`)`

See *`command_list_begin()`*.

Connection

`mopidy.mpd.protocol.connection.`**`close`**`(`*context*`)`

musicpd.org, connection section:

```
close
```

Closes the connection to MPD.

`mopidy.mpd.protocol.connection.`**`kill`**`(`*context*`)`

musicpd.org, connection section:

```
kill
```

Kills MPD.

`mopidy.mpd.protocol.connection.`**`password`**`(`*context*, *password*`)`

musicpd.org, connection section:

```
password {PASSWORD}
```

This is used for authentication with the server. `PASSWORD` is simply the plaintext password.

`mopidy.mpd.protocol.connection.`**`ping`**`(`*context*`)`

musicpd.org, connection section:

`ping`

Does nothing but return `OK`.

Current playlist

`mopidy.mpd.protocol.current_playlist.`**`add`**`(`*context*, *uri*`)`

musicpd.org, current playlist section:

`add {URI}`

Adds the file `URI` to the playlist (directories add recursively). `URI` can also be a single file.

Clarifications:

- `add ""` should add all tracks in the library to the current playlist.

`mopidy.mpd.protocol.current_playlist.`**`addid`**`(`*context*, *uri*, *songpos=None*`)`

musicpd.org, current playlist section:

`addid {URI} [POSITION]`

Adds a song to the playlist (non-recursive) and returns the song id.

`URI` is always a single file or URL. For example:

```
addid "foo.mp3"
Id: 999
OK
```

Clarifications:

- `addid ""` should return an error.

`mopidy.mpd.protocol.current_playlist.`**`addtagid`**`(`*context*, *tlid*, *tag*, *value*`)`

musicpd.org, current playlist section:

`addtagid {SONGID} {TAG} {VALUE}`

Adds a tag to the specified song. Editing song tags is only possible for remote songs. This change is volatile: it may be overwritten by tags received from the server, and the data is gone when the song gets removed from the queue.

New in version 0.19: New in MPD protocol version 0.19

`mopidy.mpd.protocol.current_playlist.`**`clear`**`(`*context*`)`

musicpd.org, current playlist section:

`clear`

Clears the current playlist.

`mopidy.mpd.protocol.current_playlist.`**`cleartagid`**`(`*context*, *tlid*, *tag*`)`

musicpd.org, current playlist section:

`cleartagid {SONGID} [TAG]`

Removes tags from the specified song. If TAG is not specified, then all tag values will be removed. Editing song tags is only possible for remote songs.

New in version 0.19: New in MPD protocol version 0.19

`mopidy.mpd.protocol.current_playlist.`**`delete`**`(`*context*, *songrange*`)`

musicpd.org, current playlist section:

```
delete [{POS} | {START:END}]
```

Deletes a song from the playlist.

`mopidy.mpd.protocol.current_playlist.`**`deleteid`**(*context*, *tlid*)

musicpd.org, current playlist section:

```
deleteid {SONGID}
```

Deletes the song `SONGID` from the playlist

`mopidy.mpd.protocol.current_playlist.`**`move_range`**(*context*, *songrange*, *to*)

musicpd.org, current playlist section:

```
move [{FROM} | {START:END}] {TO}
```

Moves the song at `FROM` or range of songs at `START:END` to `TO` in the playlist.

`mopidy.mpd.protocol.current_playlist.`**`moveid`**(*context*, *tlid*, *to*)

musicpd.org, current playlist section:

```
moveid {FROM} {TO}
```

Moves the song with `FROM` (songid) to `TO` (playlist index) in the playlist. If `TO` is negative, it is relative to the current song in the playlist (if there is one).

`mopidy.mpd.protocol.current_playlist.`**`playlist`**(*context*)

musicpd.org, current playlist section:

```
playlist
```

Displays the current playlist.

Note: Do not use this, instead use `playlistinfo`.

`mopidy.mpd.protocol.current_playlist.`**`playlistfind`**(*context*, *tag*, *needle*)

musicpd.org, current playlist section:

```
playlistfind {TAG} {NEEDLE}
```

Finds songs in the current playlist with strict matching.

`mopidy.mpd.protocol.current_playlist.`**`playlistid`**(*context*, *tlid=None*)

musicpd.org, current playlist section:

```
playlistid {SONGID}
```

Displays a list of songs in the playlist. `SONGID` is optional and specifies a single song to display info for.

`mopidy.mpd.protocol.current_playlist.`**`playlistinfo`**(*context*, *parameter=None*)

musicpd.org, current playlist section:

```
playlistinfo [[SONGPOS] | [START:END]]
```

Displays a list of all songs in the playlist, or if the optional argument is given, displays information only for the song `SONGPOS` or the range of songs `START:END`.

ncmpc and mpc:

- uses negative indexes, like `playlistinfo "-1"`, to request the entire playlist

`mopidy.mpd.protocol.current_playlist.`**`playlistsearch`**(*context*, *tag*, *needle*)

musicpd.org, current playlist section:

```
playlistsearch {TAG} {NEEDLE}
```

Searches case-sensitively for partial matches in the current playlist.

GMPC:

- uses `filename` and `any` as tags

`mopidy.mpd.protocol.current_playlist.`**`plchanges`**(*context*, *version*)
musicpd.org, current playlist section:

```
plchanges {VERSION}
```

Displays changed songs currently in the playlist since `VERSION`.

To detect songs that were deleted at the end of the playlist, use `playlistlength` returned by status command.

MPDroid:

- Calls `plchanges "-1"` two times per second to get the entire playlist.

`mopidy.mpd.protocol.current_playlist.`**`plchangesposid`**(*context*, *version*)
musicpd.org, current playlist section:

```
plchangesposid {VERSION}
```

Displays changed songs currently in the playlist since `VERSION`. This function only returns the position and the id of the changed song, not the complete metadata. This is more bandwidth efficient.

To detect songs that were deleted at the end of the playlist, use `playlistlength` returned by status command.

`mopidy.mpd.protocol.current_playlist.`**`prio`**(*context*, *priority*, *position*)
musicpd.org, current playlist section:

```
prio {PRIORITY} {START:END...}
```

Set the priority of the specified songs. A higher priority means that it will be played first when "random" mode is enabled.

A priority is an integer between 0 and 255. The default priority of new songs is 0.

`mopidy.mpd.protocol.current_playlist.`**`prioid`**(*context*, **args*)
musicpd.org, current playlist section:

```
prioid {PRIORITY} {ID...}
```

Same as prio, but address the songs with their id.

`mopidy.mpd.protocol.current_playlist.`**`rangeid`**(*context*, *tlid*, *songrange*)
musicpd.org, current playlist section:

```
rangeid {ID} {START:END}
```

Specifies the portion of the song that shall be played. START and END are offsets in seconds (fractional seconds allowed); both are optional. Omitting both (i.e. sending just ":") means "remove the range, play everything". A song that is currently playing cannot be manipulated this way.

New in version 0.19: New in MPD protocol version 0.19

`mopidy.mpd.protocol.current_playlist.`**`shuffle`**(*context*, *songrange=None*)
musicpd.org, current playlist section:

```
shuffle [START:END]
```

Shuffles the current playlist. `START:END` is optional and specifies a range of songs.

`mopidy.mpd.protocol.current_playlist.`**`swap`**`(`*context*, *songpos1*, *songpos2*`)`

musicpd.org, current playlist section:

`swap {SONG1} {SONG2}`

Swaps the positions of `SONG1` and `SONG2`.

`mopidy.mpd.protocol.current_playlist.`**`swapid`**`(`*context*, *tlid1*, *tlid2*`)`

musicpd.org, current playlist section:

`swapid {SONG1} {SONG2}`

Swaps the positions of `SONG1` and `SONG2` (both song ids).

Mounts and neighbors

`mopidy.mpd.protocol.mount.`**`listmounts`**`(`*context*`)`

musicpd.org, mounts and neighbors section:

`listmounts`

Queries a list of all mounts. By default, this contains just the configured music_directory. Example:

```
listmounts
mount:
storage: /home/foo/music
mount: foo
storage: nfs://192.168.1.4/export/mp3
OK
```

New in version 0.19: New in MPD protocol version 0.19

`mopidy.mpd.protocol.mount.`**`listneighbors`**`(`*context*`)`

musicpd.org, mounts and neighbors section:

`listneighbors`

Queries a list of "neighbors" (e.g. accessible file servers on the local net). Items on that list may be used with the mount command. Example:

```
listneighbors
neighbor: smb://FOO
name: FOO (Samba 4.1.11-Debian)
OK
```

New in version 0.19: New in MPD protocol version 0.19

`mopidy.mpd.protocol.mount.`**`mount`**`(`*context*, *path*, *uri*`)`

musicpd.org, mounts and neighbors section:

`mount {PATH} {URI}`

Mount the specified remote storage URI at the given path. Example:

```
mount foo nfs://192.168.1.4/export/mp3
```

New in version 0.19: New in MPD protocol version 0.19

`mopidy.mpd.protocol.mount.`**`unmount`**`(`*context*, *path*`)`

musicpd.org, mounts and neighbors section:

`unmount {PATH}`

Unmounts the specified path. Example:

```
unmount foo
```

New in version 0.19: New in MPD protocol version 0.19

Music database

`mopidy.mpd.protocol.music_db.`**`count`**`(`*context*, **args*`)`

musicpd.org, music database section:

`count {TAG} {NEEDLE}`

Counts the number of songs and their total playtime in the db matching `TAG` exactly.

GMPC:

- use multiple tag-needle pairs to make more specific searches.

`mopidy.mpd.protocol.music_db.`**`find`**`(`*context*, **args*`)`

musicpd.org, music database section:

`find {TYPE} {WHAT}`

Finds songs in the db that are exactly `WHAT`. `TYPE` can be any tag supported by MPD, or one of the two special parameters - `file` to search by full path (relative to database root), and `any` to match against all available tags. `WHAT` is what to find.

GMPC:

- also uses `find album "[ALBUM]" artist "[ARTIST]"` to list album tracks.

ncmpc:

- capitalizes the type argument.

ncmpcpp:

- also uses the search type “date”.
- uses “file” instead of “filename”.

`mopidy.mpd.protocol.music_db.`**`findadd`**`(`*context*, **args*`)`

musicpd.org, music database section:

`findadd {TYPE} {WHAT}`

Finds songs in the db that are exactly `WHAT` and adds them to current playlist. Parameters have the same meaning as for `find`.

`mopidy.mpd.protocol.music_db.`**`list_`**`(`*context*, **args*`)`

musicpd.org, music database section:

`list {TYPE} [ARTIST]`

Lists all tags of the specified type. `TYPE` should be `album`, `artist`, `albumartist`, `date`, or `genre`.

`ARTIST` is an optional parameter when type is `album`, `date`, or `genre`. This filters the result list by an artist.

Clarifications:

The musicpd.org documentation for `list` is far from complete. The command also supports the following variant:

`list {TYPE} {QUERY}`

Where `QUERY` applies to all `TYPE`. `QUERY` is one or more pairs of a field name and a value. If the `QUERY` consists of more than one pair, the pairs are AND-ed together to find the result. Examples of valid queries and what they should return:

`list "artist" "artist" "ABBA"` List artists where the artist name is “ABBA”. Response:

```
Artist: ABBA
OK
```

`list "album" "artist" "ABBA"` Lists albums where the artist name is “ABBA”. Response:

```
Album: More ABBA Gold: More ABBA Hits
Album: Absolute More Christmas
Album: Gold: Greatest Hits
OK
```

`list "artist" "album" "Gold: Greatest Hits"` Lists artists where the album name is “Gold: Greatest Hits”. Response:

```
Artist: ABBA
OK
```

`list "artist" "artist" "ABBA" "artist" "TLC"` Lists artists where the artist name is “ABBA” *and* “TLC”. Should never match anything. Response:

```
OK
```

`list "date" "artist" "ABBA"` Lists dates where artist name is “ABBA”. Response:

```
Date:
Date: 1992
Date: 1993
OK
```

`list "date" "artist" "ABBA" "album" "Gold: Greatest Hits"` Lists dates where artist name is “ABBA” and album name is “Gold: Greatest Hits”. Response:

```
Date: 1992
OK
```

`list "genre" "artist" "The Rolling Stones"` Lists genres where artist name is “The Rolling Stones”. Response:

```
Genre:
Genre: Rock
OK
```

ncmpc:

- capitalizes the field argument.

`mopidy.mpd.protocol.music_db.`**`listall`**(*context*, *uri=None*)

musicpd.org, music database section:

```
listall [URI]
```

Lists all songs and directories in `URI`.

Do not use this command. Do not manage a client-side copy of MPD's database. That is fragile and adds huge overhead. It will break with large databases. Instead, query MPD whenever you need something.

Warning: This command is disabled by default in Mopidy installs.

`mopidy.mpd.protocol.music_db.`**`listallinfo`**(*context*, *uri=None*)

musicpd.org, music database section:

```
listallinfo [URI]
```

Same as `listall`, except it also returns metadata info in the same format as `lsinfo`.

Do not use this command. Do not manage a client-side copy of MPD's database. That is fragile and adds huge overhead. It will break with large databases. Instead, query MPD whenever you need something.

Warning: This command is disabled by default in Mopidy installs.

`mopidy.mpd.protocol.music_db.`**`listfiles`**(*context*, *uri=None*)

musicpd.org, music database section:

```
listfiles [URI]
```

Lists the contents of the directory URI, including files are not recognized by MPD. URI can be a path relative to the music directory or an URI understood by one of the storage plugins. The response contains at least one line for each directory entry with the prefix "file: " or "directory: ", and may be followed by file attributes such as "Last-Modified" and "size".

For example, "smb://SERVER" returns a list of all shares on the given SMB/CIFS server; "nfs://servername/path" obtains a directory listing from the NFS server.

New in version 0.19: New in MPD protocol version 0.19

`mopidy.mpd.protocol.music_db.`**`lsinfo`**(*context*, *uri=None*)

musicpd.org, music database section:

```
lsinfo [URI]
```

Lists the contents of the directory `URI`.

When listing the root directory, this currently returns the list of stored playlists. This behavior is deprecated; use `listplaylists` instead.

MPD returns the same result, including both playlists and the files and directories located at the root level, for both `lsinfo`, `lsinfo ""`, and `lsinfo "/"`.

`mopidy.mpd.protocol.music_db.`**`readcomments`**(*context*, *uri*)

musicpd.org, music database section:

```
readcomments [URI]
```

Read "comments" (i.e. key-value pairs) from the file specified by "URI". This "URI" can be a path relative to the music directory or a URL in the form "file:///foo/bar.ogg".

This command may be used to list metadata of remote files (e.g. URI beginning with "http://" or "smb://").

The response consists of lines in the form "KEY: VALUE". Comments with suspicious characters (e.g. newlines) are ignored silently.

The meaning of these depends on the codec, and not all decoder plugins support it. For example, on Ogg files, this lists the Vorbis comments.

`mopidy.mpd.protocol.music_db.`**`rescan`**(*context*, *uri=None*)

musicpd.org, music database section:

```
rescan [URI]
```

Same as `update`, but also rescans unmodified files.

`mopidy.mpd.protocol.music_db.`**`search`**(*context*, **args*)

musicpd.org, music database section:

```
search {TYPE} {WHAT} [...]
```

Searches for any song that contains `WHAT`. Parameters have the same meaning as for `find`, except that search is not case sensitive.

GMPC:

- uses the undocumented field `any`.
- searches for multiple words like this:

```
search any "foo" any "bar" any "baz"
```

ncmpc:

- capitalizes the field argument.

ncmpcpp:

- also uses the search type "date".
- uses "file" instead of "filename".

`mopidy.mpd.protocol.music_db.`**`searchadd`**(*context*, **args*)

musicpd.org, music database section:

```
searchadd {TYPE} {WHAT} [...]
```

Searches for any song that contains `WHAT` in tag `TYPE` and adds them to current playlist.

Parameters have the same meaning as for `find`, except that search is not case sensitive.

`mopidy.mpd.protocol.music_db.`**`searchaddpl`**(*context*, **args*)

musicpd.org, music database section:

```
searchaddpl {NAME} {TYPE} {WHAT} [...]
```

Searches for any song that contains `WHAT` in tag `TYPE` and adds them to the playlist named `NAME`.

If a playlist by that name doesn't exist it is created.

Parameters have the same meaning as for `find`, except that search is not case sensitive.

`mopidy.mpd.protocol.music_db.`**`update`**(*context*, *uri=None*)

musicpd.org, music database section:

`update [URI]`

Updates the music database: find new files, remove deleted files, update modified files.

`URI` is a particular directory or song/file to update. If you do not specify it, everything is updated.

Prints `updating_db: JOBID` where `JOBID` is a positive number identifying the update job. You can read the current job id in the `status` response.

Playback

`mopidy.mpd.protocol.playback.`**`consume`**`(`*context*, *state*`)`

musicpd.org, playback section:

`consume {STATE}`

Sets consume state to `STATE`, `STATE` should be 0 or 1. When consume is activated, each song played is removed from playlist.

`mopidy.mpd.protocol.playback.`**`crossfade`**`(`*context*, *seconds*`)`

musicpd.org, playback section:

`crossfade {SECONDS}`

Sets crossfading between songs.

`mopidy.mpd.protocol.playback.`**`mixrampdb`**`(`*context*, *decibels*`)`

musicpd.org, playback section:

`mixrampdb {deciBels}`

Sets the threshold at which songs will be overlapped. Like crossfading but doesn't fade the track volume, just overlaps. The songs need to have MixRamp tags added by an external tool. 0dB is the normalized maximum volume so use negative values, I prefer -17dB. In the absence of mixramp tags crossfading will be used. See http://sourceforge.net/projects/mixramp

`mopidy.mpd.protocol.playback.`**`mixrampdelay`**`(`*context*, *seconds*`)`

musicpd.org, playback section:

`mixrampdelay {SECONDS}`

Additional time subtracted from the overlap calculated by mixrampdb. A value of "nan" disables MixRamp overlapping and falls back to crossfading.

`mopidy.mpd.protocol.playback.`**`next_`**`(`*context*`)`

musicpd.org, playback section:

`next`

Plays next song in the playlist.

MPD's behaviour when affected by repeat/random/single/consume:

Given a playlist of three tracks numbered 1, 2, 3, and a currently playing track `c`. `next_track` is defined at the track that will be played upon calls to `next`.

Tests performed on MPD 0.15.4-1ubuntu3.

repeat	random	single	consume	c = 1	c = 2	c = 3	Notes
T	T	T	T	2	3	EOPL	
T	T	T	.	Rand	Rand	Rand	[1]
T	T	.	T	Rand	Rand	Rand	[4]
T	T	.	.	Rand	Rand	Rand	[4]
T	.	T	T	2	3	EOPL	
T	.	T	.	2	3	1	
T	.	.	T	3	3	EOPL	
T	.	.	.	2	3	1	
.	T	T	T	Rand	Rand	Rand	[3]
.	T	T	.	Rand	Rand	Rand	[3]
.	T	.	T	Rand	Rand	Rand	[2]
.	T	.	.	Rand	Rand	Rand	[2]
.	.	T	T	2	3	EOPL	
.	.	T	.	2	3	EOPL	
.	.	.	T	2	3	EOPL	
.	.	.	.	2	3	EOPL	

- When end of playlist (EOPL) is reached, the current track is unset.
- [1] When *random* and *single* is combined, `next` selects a track randomly at each invocation, and not just the next track in an internal prerandomized playlist.
- [2] When *random* is active, `next` will skip through all tracks in the playlist in random order, and finally EOPL is reached.
- [3] *single* has no effect in combination with *random* alone, or *random* and *consume*.
- [4] When *random* and *repeat* is active, EOPL is never reached, but the playlist is played again, in the same random order as the first time.

`mopidy.mpd.protocol.playback.`**`pause`**`(`*context*, *state=None*`)`

musicpd.org, playback section:

```
pause {PAUSE}
```

Toggles pause/resumes playing, `PAUSE` is 0 or 1.

MPDroid:

- Calls `pause` without any arguments to toogle pause.

`mopidy.mpd.protocol.playback.`**`play`**`(`*context*, *songpos=None*`)`

musicpd.org, playback section:

```
play [SONGPOS]
```

Begins playing the playlist at song number `SONGPOS`.

The original MPD server resumes from the paused state on `play` without arguments.

Clarifications:

- `play "-1"` when playing is ignored.
- `play "-1"` when paused resumes playback.
- `play "-1"` when stopped with a current track starts playback at the current track.

- `play "-1"` when stopped without a current track, e.g. after playlist replacement, starts playback at the first track.

BitMPC:

- issues `play 6` without quotes around the argument.

`mopidy.mpd.protocol.playback.`**`playid`**`(`*context*, *tlid*`)`

musicpd.org, playback section:

```
playid [SONGID]
```

Begins playing the playlist at song `SONGID`.

Clarifications:

- `playid "-1"` when playing is ignored.
- `playid "-1"` when paused resumes playback.
- `playid "-1"` when stopped with a current track starts playback at the current track.
- `playid "-1"` when stopped without a current track, e.g. after playlist replacement, starts playback at the first track.

`mopidy.mpd.protocol.playback.`**`previous`**`(`*context*`)`

musicpd.org, playback section:

```
previous
```

Plays previous song in the playlist.

MPD's behaviour when affected by repeat/random/single/consume:

Given a playlist of three tracks numbered 1, 2, 3, and a currently playing track `c`. `previous_track` is defined at the track that will be played upon `previous` calls.

Tests performed on MPD 0.15.4-1ubuntu3.

repeat	random	single	consume	c = 1	c = 2	c = 3
T	T	T	T	Rand?	Rand?	Rand?
T	T	T	.	3	1	2
T	T	.	T	Rand?	Rand?	Rand?
T	T	.	.	3	1	2
T	.	T	T	3	1	2
T	.	T	.	3	1	2
T	.	.	T	3	1	2
T	.	.	.	3	1	2
.	T	T	T	c	c	c
.	T	T	.	c	c	c
.	T	.	T	c	c	c
.	T	.	.	c	c	c
.	.	T	T	1	1	2
.	.	T	.	1	1	2
.	.	.	T	1	1	2
.	.	.	.	1	1	2

- If `time_position` of the current track is 15s or more, `previous` should do a seek to time position 0.

`mopidy.mpd.protocol.playback.`**`random`**`(`*context*, *state*`)`
musicpd.org, playback section:

```
random {STATE}
```

Sets random state to `STATE`, `STATE` should be 0 or 1.

`mopidy.mpd.protocol.playback.`**`repeat`**`(`*context*, *state*`)`
musicpd.org, playback section:

```
repeat {STATE}
```

Sets repeat state to `STATE`, `STATE` should be 0 or 1.

`mopidy.mpd.protocol.playback.`**`replay_gain_mode`**`(`*context*, *mode*`)`
musicpd.org, playback section:

```
replay_gain_mode {MODE}
```

Sets the replay gain mode. One of `off`, `track`, `album`.

Changing the mode during playback may take several seconds, because the new settings does not affect the buffered data.

This command triggers the options idle event.

`mopidy.mpd.protocol.playback.`**`replay_gain_status`**`(`*context*`)`
musicpd.org, playback section:

```
replay_gain_status
```

Prints replay gain options. Currently, only the variable `replay_gain_mode` is returned.

`mopidy.mpd.protocol.playback.`**`seek`**`(`*context*, *songpos*, *seconds*`)`
musicpd.org, playback section:

```
seek {SONGPOS} {TIME}
```

Seeks to the position `TIME` (in seconds) of entry `SONGPOS` in the playlist.

Droid MPD:

- issues `seek 1 120` without quotes around the arguments.

`mopidy.mpd.protocol.playback.`**`seekcur`**`(`*context*, *time*`)`
musicpd.org, playback section:

```
seekcur {TIME}
```

Seeks to the position `TIME` within the current song. If prefixed by '+' or '-', then the time is relative to the current playing position.

`mopidy.mpd.protocol.playback.`**`seekid`**`(`*context*, *tlid*, *seconds*`)`
musicpd.org, playback section:

```
seekid {SONGID} {TIME}
```

Seeks to the position `TIME` (in seconds) of song `SONGID`.

`mopidy.mpd.protocol.playback.`**`setvol`**`(`*context*, *volume*`)`
musicpd.org, playback section:

```
setvol {VOL}
```

Sets volume to `VOL`, the range of volume is 0-100.

Droid MPD:

- issues `setvol 50` without quotes around the argument.

`mopidy.mpd.protocol.playback.`**`single`**(*context*, *state*)

musicpd.org, playback section:

`single {STATE}`

Sets single state to `STATE`, `STATE` should be 0 or 1. When single is activated, playback is stopped after current song, or song is repeated if the `repeat` mode is enabled.

`mopidy.mpd.protocol.playback.`**`stop`**(*context*)

musicpd.org, playback section:

`stop`

Stops playing.

`mopidy.mpd.protocol.playback.`**`volume`**(*context*, *change*)

musicpd.org, playback section:

`volume {CHANGE}`

Changes volume by amount `CHANGE`.

Note: `volume` is deprecated, use `setvol` instead.

Reflection

`mopidy.mpd.protocol.reflection.`**`commands`**(*context*)

musicpd.org, reflection section:

`commands`

Shows which commands the current user has access to.

`mopidy.mpd.protocol.reflection.`**`config`**(*context*)

musicpd.org, reflection section:

`config`

Dumps configuration values that may be interesting for the client. This command is only permitted to "local" clients (connected via UNIX domain socket).

`mopidy.mpd.protocol.reflection.`**`decoders`**(*context*)

musicpd.org, reflection section:

`decoders`

Print a list of decoder plugins, followed by their supported suffixes and MIME types. Example response:

```
plugin: mad
suffix: mp3
suffix: mp2
mime_type: audio/mpeg
plugin: mpcdec
suffix: mpc
```

Clarifications:

- ncmpcpp asks for decoders the first time you open the browse view. By returning nothing and OK instead of an not implemented error, we avoid "Not implemented" showing up in the ncmpcpp interface, and we get the list of playlists without having to enter the browse interface twice.

`mopidy.mpd.protocol.reflection.`**`notcommands`**`(`*context*`)`
: *musicpd.org, reflection section:*

  `notcommands`

  Shows which commands the current user does not have access to.

`mopidy.mpd.protocol.reflection.`**`tagtypes`**`(`*context*`)`
: *musicpd.org, reflection section:*

  `tagtypes`

  Shows a list of available song metadata.

`mopidy.mpd.protocol.reflection.`**`urlhandlers`**`(`*context*`)`
: *musicpd.org, reflection section:*

  `urlhandlers`

  Gets a list of available URL handlers.

Status

`mopidy.mpd.protocol.status.`**`SUBSYSTEMS = [u'database', u'mixer', u'options', u'output', u'p`**
: Subsystems that can be registered with idle command.

`mopidy.mpd.protocol.status.`**`clearerror`**`(`*context*`)`
: *musicpd.org, status section:*

  `clearerror`

  Clears the current error message in status (this is also accomplished by any command that starts playback).

`mopidy.mpd.protocol.status.`**`currentsong`**`(`*context*`)`
: *musicpd.org, status section:*

  `currentsong`

  Displays the song info of the current song (same song that is identified in status).

`mopidy.mpd.protocol.status.`**`idle`**`(`*context*, **subsystems*`)`
: *musicpd.org, status section:*

  `idle [SUBSYSTEMS...]`

  Waits until there is a noteworthy change in one or more of MPD's subsystems. As soon as there is one, it lists all changed systems in a line in the format `changed: SUBSYSTEM`, where `SUBSYSTEM` is one of the following:

  - `database`: the song database has been modified after update.
  - `update`: a database update has started or finished. If the database was modified during the update, the database event is also emitted.
  - `stored_playlist`: a stored playlist has been modified, renamed, created or deleted
  - `playlist`: the current playlist has been modified
  - `player`: the player has been started, stopped or seeked
  - `mixer`: the volume has been changed
  - `output`: an audio output has been enabled or disabled
  - `options`: options like repeat, random, crossfade, replay gain

While a client is waiting for idle results, the server disables timeouts, allowing a client to wait for events as long as MPD runs. The idle command can be canceled by sending the command `noidle` (no other commands are allowed). MPD will then leave idle mode and print results immediately; might be empty at this time.

If the optional `SUBSYSTEMS` argument is used, MPD will only send notifications when something changed in one of the specified subsystems.

`mopidy.mpd.protocol.status.`**`noidle`**`(`*context*`)`

See `_status_idle()`.

`mopidy.mpd.protocol.status.`**`stats`**`(`*context*`)`

musicpd.org, status section:

```
stats
```

Displays statistics.

- `artists`: number of artists
- `songs`: number of albums
- `uptime`: daemon uptime in seconds
- `db_playtime`: sum of all song times in the db
- `db_update`: last db update in UNIX time
- `playtime`: time length of music played

`mopidy.mpd.protocol.status.`**`status`**`(`*context*`)`

musicpd.org, status section:

```
status
```

Reports the current status of the player and the volume level.

- `volume`: 0-100 or -1
- `repeat`: 0 or 1
- `single`: 0 or 1
- `consume`: 0 or 1
- `playlist`: 31-bit unsigned integer, the playlist version number
- `playlistlength`: integer, the length of the playlist
- `state`: play, stop, or pause
- `song`: playlist song number of the current song stopped on or playing
- `songid`: playlist songid of the current song stopped on or playing
- `nextsong`: playlist song number of the next song to be played
- `nextsongid`: playlist songid of the next song to be played
- `time`: total time elapsed (of current playing/paused song)
- `elapsed`: Total time elapsed within the current song, but with higher resolution.
- `bitrate`: instantaneous bitrate in kbps
- `xfade`: crossfade in seconds
- `audio`: sampleRate``:bits``:channels

- `updatings_db`: job id
- `error`: if there is an error, returns message here

Clarifications based on experience implementing

- `volume`: can also be -1 if no output is set.
- `elapsed`: Higher resolution means time in seconds with three decimal places for millisecond precision.

Stickers

`mopidy.mpd.protocol.stickers.`**`sticker`**(*context*, *action*, *field*, *uri*, *name=None*, *value=None*)

musicpd.org, sticker section:

`sticker list {TYPE} {URI}`

Lists the stickers for the specified object.

`sticker find {TYPE} {URI} {NAME}`

Searches the sticker database for stickers with the specified name, below the specified directory (`URI`). For each matching song, it prints the `URI` and that one sticker's value.

`sticker get {TYPE} {URI} {NAME}`

Reads a sticker value for the specified object.

`sticker set {TYPE} {URI} {NAME} {VALUE}`

Adds a sticker value to the specified object. If a sticker item with that name already exists, it is replaced.

`sticker delete {TYPE} {URI} [NAME]`

Deletes a sticker value from the specified object. If you do not specify a sticker name, all sticker values are deleted.

Stored playlists

`mopidy.mpd.protocol.stored_playlists.`**`listplaylist`**(*context*, *name*)

musicpd.org, stored playlists section:

`listplaylist {NAME}`

Lists the files in the playlist `NAME.m3u`.

Output format:

```
file: relative/path/to/file1.flac
file: relative/path/to/file2.ogg
file: relative/path/to/file3.mp3
```

`mopidy.mpd.protocol.stored_playlists.`**`listplaylistinfo`**(*context*, *name*)

musicpd.org, stored playlists section:

`listplaylistinfo {NAME}`

Lists songs in the playlist `NAME.m3u`.

Output format:

Standard track listing, with fields: file, Time, Title, Date, Album, Artist, Track

`mopidy.mpd.protocol.stored_playlists.`**`listplaylists`**(*context*)

musicpd.org, stored playlists section:

`listplaylists`

Prints a list of the playlist directory.

After each playlist name the server sends its last modification time as attribute `Last-Modified` in ISO 8601 format. To avoid problems due to clock differences between clients and the server, clients should not compare this value with their local clock.

Output format:

```
playlist: a
Last-Modified: 2010-02-06T02:10:25Z
playlist: b
Last-Modified: 2010-02-06T02:11:08Z
```

Clarifications:

- ncmpcpp 0.5.10 segfaults if we return 'playlist: ' on a line, so we must ignore playlists without names, which isn't very useful anyway.

`mopidy.mpd.protocol.stored_playlists.`**`load`**(*context*, *name*, *playlist_slice=slice(0*, *None*, *None)*)

musicpd.org, stored playlists section:

`load {NAME} [START:END]`

Loads the playlist into the current queue. Playlist plugins are supported. A range may be specified to load only a part of the playlist.

Clarifications:

- `load` appends the given playlist to the current playlist.
- MPD 0.17.1 does not support open-ended ranges, i.e. without end specified, for the `load` command, even though MPD's general range docs allows open-ended ranges.
- MPD 0.17.1 does not fail if the specified range is outside the playlist, in either or both ends.

`mopidy.mpd.protocol.stored_playlists.`**`playlistadd`**(*context*, *name*, *uri*)

musicpd.org, stored playlists section:

`playlistadd {NAME} {URI}`

Adds `URI` to the playlist `NAME.m3u`.

`NAME.m3u` will be created if it does not exist.

`mopidy.mpd.protocol.stored_playlists.`**`playlistclear`**(*context*, *name*)

musicpd.org, stored playlists section:

`playlistclear {NAME}`

Clears the playlist `NAME.m3u`.

`mopidy.mpd.protocol.stored_playlists.`**`playlistdelete`**(*context*, *name*, *songpos*)

musicpd.org, stored playlists section:

`playlistdelete {NAME} {SONGPOS}`

Deletes `SONGPOS` from the playlist `NAME.m3u`.

`mopidy.mpd.protocol.stored_playlists.`**`playlistmove`**(*context*, *name*, *from_pos*, *to_pos*)

musicpd.org, stored playlists section:

```
playlistmove {NAME} {SONGID} {SONGPOS}
```

Moves `SONGID` in the playlist `NAME.m3u` to the position `SONGPOS`.

Clarifications:

- The second argument is not a `SONGID` as used elsewhere in the protocol documentation, but just the `SONGPOS` to move *from*, i.e. `playlistmove {NAME} {FROM_SONGPOS} {TO_SONGPOS}`.

`mopidy.mpd.protocol.stored_playlists.`**`rename`**(*context*, *old_name*, *new_name*)

musicpd.org, stored playlists section:

```
rename {NAME} {NEW_NAME}
```

Renames the playlist `NAME.m3u` to `NEW_NAME.m3u`.

`mopidy.mpd.protocol.stored_playlists.`**`rm`**(*context*, *name*)

musicpd.org, stored playlists section:

```
rm {NAME}
```

Removes the playlist `NAME.m3u` from the playlist directory.

`mopidy.mpd.protocol.stored_playlists.`**`save`**(*context*, *name*)

musicpd.org, stored playlists section:

```
save {NAME}
```

Saves the current playlist to `NAME.m3u` in the playlist directory.

CHAPTER 34

Indices and tables

- genindex
- modindex

Python Module Index

Index