Tribler Documentation

Release 6.6

Tribler devs

November 15, 2016

Contents

1 Tribler **3**
1.1 Obtaining the latest release . . . 3
1.2 Obtaining support . . . 3
1.3 Contributing . . . 3
1.4 Setting up your development environment . . . 4
1.5 Running Tribler from the repository . . . 4
1.6 Packaging Tribler . . . 4
1.7 Submodule notes . . . 4

2 How to contribute to the Tribler project? **5**
2.1 Checking out the Stabilization Branch . . . 5
2.2 Reporting bugs . . . 5
2.3 Pull requests . . . 6

3 Branching model and development methodology **7**
3.1 Branching model . . . 7
3.2 Tags . . . 7
3.3 Setting up the local repo . . . 7
3.4 Working on new features or fixes . . . 8
3.5 Getting your changes merged upstream . . . 8
3.6 Misc guidelines . . . 9

4 Setting up your development environment **11**
4.1 Windows . . . 11
4.2 OS X . . . 14
4.3 Linux . . . 17

5 Building Tribler **19**
5.1 Windows . . . 19
5.2 OS X . . . 20
5.3 Debian and derivatives . . . 21
5.4 Other Unixes . . . 21

6 Architectural Structure **23**
6.1 Modules . . . 24
6.2 Maintenance . . . 25

7 Tribler REST API **27**
7.1 Overview . . . 27

7.2 Making requests 27
7.3 Error handling 27
7.4 Download states 28
7.5 Endpoints 28

8 Indices and tables **47**

HTTP Routing Table **49**

Python Module Index **51**

Contents:

CHAPTER 1

Tribler

Towards making Bittorrent anonymous and impossible to shut down.

Developers usually hang out in the official IRC channel #tribler @ FreeNode (click here for direct a webchat window)

We use our own dedicated Tor-like network for anonymous torrent downloading. We implemented and enhanced the *Tor protocol specifications* plus merged them with Bittorrent streaming. More info: https://github.com/Tribler/tribler/wiki Tribler includes our own Tor-like onion routing network with hidden services based seeding and end-to-end encryption, detailed specs: https://github.com/Tribler/tribler/wiki/Anonymous-Downloading-and-Streaming-specifications

The aim of Tribler is giving anonymous access to online (streaming) videos. We are trying to make privacy, strong cryptography and authentication the Internet norm.

Tribler currently offers a Youtube-style service. For instance, Bittorrent-compatible streaming, fast search, thumbnail previews and comments. For the past 9 years we have been building a very robust Peer-to-Peer system. Today Tribler is robust: "the only way to take Tribler down is to take The Internet down" (but a single software bug could end everything).

We make use of submodules, so remember using the –recursive argument when cloning this repo.

1.1 Obtaining the latest release

Just click here and download the latest package for your OS.

1.2 Obtaining support

If you found a bug or have a feature request, please make sure you read our contributing page and then open an issue. We will have a look at it ASAP.

1.3 Contributing

Contributions are very welcome! If you are interested in contributing code or otherwise, please have a look at our contributing page. Have a look at the issue tracker if you are looking for inspiration :).

1.4 Setting up your development environment

We support development on Linux, OS X and Windows. We have written documentation that guides you through installing the required packages when setting up a Tribler development environment. See our Linux development guide for the guide on setting up a development environment on Linux distributions. See our Windows development guide for setting everything up on Windows. See our OS X development guide for the guide to setup the development environment on OS X. For German translations, see here.

1.5 Running Tribler from the repository

First clone the repository:

```
git clone --recursive git@github.com:Tribler/tribler.git
```

or, if you haven't added your ssh key to your github account:

```
git clone --recursive https://github.com/Tribler/tribler.git
```

Second, install the dependencies.

Done! Now you can run tribler by executing the `tribler.sh` script on the root of the repository:

```
./tribler.sh
```

On Windows, you can use the following command to run Tribler:

```
python Tribler\Main\tribler.py
```

1.6 Packaging Tribler

We have written guides on how to package Tribler for distribution on various systems. Please take a look here.

1.7 Submodule notes

- As updated submodules are in detached head state, remember to check out a branch before commiting changes on them.
- If you forgot to check out a branch before doing a commit, you should get a warning telling you about it. To get the commit to a branch just check out the branch and do a git cherry-pick of the commit.
- Take care of not accidentally commiting a submodule revision change with `git commit -a`.
- Do not commit a submodule update without running all the tests first and making sure the new code is not breaking Tribler.

CHAPTER 2

How to contribute to the Tribler project?

2.1 Checking out the Stabilization Branch

The stabilization branch `next` contains the most up to date bugfixes. If your issue cannot be reproduced there, it is most likely already fixed.

To backup your Tribler installation and checkout the latest version of the stabilization branch, please perform the following steps. * Copy the `.Tribler` folder to a safe location on your system (for instance the desktop) Make sure to leave the original folder on its original location. This folder is located at `~/.Tribler/` (Linux/OS X) or `%APPDATA\.Tribler` (Windows). * Remove the `tribler` installation folder. * Go to the latest tested version of Tribler and under 'Build Artifacts', download the package appropriate to your operating system. * Install/unzip this package.

To revert back to your original version of Tribler, download the installer again and install it. Afterwards you can restore your backed up Tribler data folder.

2.2 Reporting bugs

- Make sure the issue/feature you want to report doesn't already exist.
- If you want to report more than one bug or feature, create individual issues for each of them.
- Use a clear descriptive title.
- **Provide at least the following information:**
  - The version of Tribler that you are using, if you are running from a branch, branch name and commit ID.
  - The OS and version you are running.
  - Step by step instructions to reproduce the issue in case it's a bug.
- **Attach Tribler's log file. On Windows, these are found in `%APPDATA%`. On Linux distributions, the log file is located in ~/**
  - Does it still happen if you move `%APPDATA\.Tribler` away temporarily? (Do **not** delete it!)
  - Do you have any other software installed that might interfere with Tribler?

2.3 Pull requests

When creating a new Pull request, please observe the following:

- Fixes go to `next`, features go to `devel`.
- Before starting to work on a feature or fix, check that nobody else is working on it by assigning yourself the corresponding issue. Create one if it doesn't exist. This is also useful to get feedback about if a given feature would be accepted. If you are not a member of the project, just drop a comment saying that you are working on that.
- Create one PR per feature/bugfix.
- Provide tests for any new features/fixes you implement and make sure they cover all methods and at least the important branches in the new/updated code.
- If implementing a reasonably big or experimental feature, make it toggleable if possible (For instance for a new community, new GUI stuff, etc.).
- **Keep a clean and nice git history:**
  - Rebase instead of merging back from the base branch.
  - Squash fixup commits together.
  - Have nice and descriptive commit messages.
- Do not commit extraneous/auto-generated files.
- Use Unix style newlines for any new file created.
- No print statements if it's not really justified (command line tools and such).
- Do an `autopep8` pass before submiting the pull request.
- Do a `pylint` pass with the `.pylintrc` on the root of the repository and make sure you are not raising the base branch violation count, it's bad enough as it is :).
- For more PR etiquette have a look here.

CHAPTER 3

Branching model and development methodology

In this post we'll explain the branching model and development metodology we use at TUDelft on the Tribler project.

This is mostly targeted at new students joining the team. However, it may give you some useful ideas if you are working on a similar project type.

3.1 Branching model

Tribler is developed mainly by university students (mostly MSC and PHDs) that will work on Tribler for a relatively short period of time. So pull requests usually require several review cycles and some of them take a long time to be completed and merged (development of new features are usually part of Master thesis subjects or papers and suchlike). This makes it rather hard to implement anything like traditional unsupervised continuous integration.

Our branching model is similar to the one described at length in Vincent Driessen's post with some small differences.

Our main repository contains 3 branches:

- **devel**: The main development branch; all new features and fixes for them belong here. Every time a new release cycle is started, the **next** branch gets replaced with a fork of **devel**.
- **next**: This is the stabilization branch where the **next** major release and subsequent maintenance releases will be tagged from. Only bug fixes for released code are merged here. As you may have guessed there's no bugfix releases while a feature release is stabilized (simply due to lack of manpower on the project). All the changes applied here here are regularly merged to **devel**.
- **master**: Contains the code of the latest stable release. It gets updated from **next** after every release.

3.2 Tags

Every revision that will result in a (pre)release gets tagged with a version number.

3.3 Setting up the local repo

1. Fork Tribler's upstream repository.
2. Make a local clone of it:

```
git clone -o MrStudent --recursive --recurse-submodules --single-branch \
git@github.com:MrStudent/tribler
```

3. Add the upstream remote:

```
git remote add upstream https://github.com/Tribler/tribler
```

Note that an /HTTPS/ URL is used here instead of an /SSH/ one (git@github.com/yadayada). This is done in order to prevent accidental pushes to the main repository. However, it will only work if you don't set up /HTTPS/ auth for github. Any attempt to push something there and git will ask you for credentials.

4. Profit!

3.4 Working on new features or fixes

1. Make sure there's an issue for it and get it assigned to you. If there isn't, create it yourself. Otherwise you risk your changes not getting accepted upstream or wasting time on changes that are already being worked on by other developers.
2. Create your feature or bugfix branch. New feature branches can be created like this:

```
git fetch --all && git checkout upstream/devel -b fix_2344_my_new_feature
```

For bug fixes:

```
git fetch --all && git checkout upstream/next -b fix_2344_my_new_bugfix
```

2344 would be the issue number this branch is dealing with. This makes it trivial to identify the purpose of a branch if one hasn't had been able to work on it for a while and can't remember right away.

3. Create a Pull Request.

It is usually a good idea to create a pull request for a branch even if it's a work in progress. Doing so will make our Jenkins instance run all the checks, tests and experiments every time you push a change so you can have continuous feedback on the state of your branch.

When creating a PR, always prepend the PR title with **WIP** until it's ready for the final round of reviews. More about this on the next section.

Notes:

- Always fork directly from upstream's remote branches as opposed to your own (remote or local) **devel** or **next** branches. Those are useless as they will quickly get out of date, so kill them with fire:

```
git branch -d next
git branch -d devel
```

- Once one of your branches has been merged upstream try to always delete them from your remote to avoid cluttering other people's remote listings (I've got around 15 remotes on my local Tribler repos and it can become annoying to look for a particular branch among dozens and dozens of other people's stale branches). This can be done either from github's PR web interface by clicking on the "delete branch" button after the merge has been done or with:

```
git push MrStudent :fix_2344_my_new_bugfix
```

3.5 Getting your changes merged upstream

When you think your PR is complete you need to get at least one peer to review your proposed changes as many times as necessary until it's ready. If you can't agree on something add another peer to the discussion to break the tie or talk to the lead developer.

All updates during the review/fix iteration cycles should be made with fixup commits to make it easier for the reviewer(s) to spot the new changes that need review on each iteration. (read the `--fixup` argument on the git-commit manpage if you don't know what a fixup commit is).

Once the reviewer gives the OK and the tests and checks are passing, the fixup commits can then be squashed and the **WIP** prefix can be switched to **READY**. The lead developer will then do the final review round.

As mentioned before, any requested modifications should come in the form of fixup commits to ease reviewing.

Once the final OK is given, all fixup commits should be squashed and the branch will get merged.

3.6 Misc guidelines

- **Keep an eye on the PRs you've reviewed** You will probably learn something from other reviewers and find out what you missed out during yours.
- **Don't send PR from your remote's ~devel~ branch** Use proper names for your branches. It will be more informative and they become part of the merge commit message.
- **Keep it small** The smaller the PRs are, the less review cycles will be needed and the quicker they will get merged.
- **Try to write as many tests as you can before writing any code** It will help you think about the problem you are trying to solve and it usually helps to write code that's easier to test.
- **Have the right amount of commits on your PRs** Don't have a feature implementation spread across a gazillion commits. For instance if a given feature requires some refactoring, your history could look like this:
  - "Refactor foo class to allow for bar" (At this point, the code should still work)
  - "Tests for feature $X"
  - "Implement feature $X"
- **Write clean and self contained commits** Each commit should make sense and be reviewable by itself. It doesn't make sense to break something on one commit and fix it on another later on in the same PR. It also makes reviews much harder.
- **Avoid unrelated and/or unnecessary modifications** If you are fixing a bug or implementing a feature, avoid unnecessary refactoring, white space changes, cosmetic code reordering, etc. It will introduce gratuitous merge conflicts to your and others' branches and make it harder to track changes (for instance with git blame).
- **Don't rename a file and modify it on the same commit** If you need to rename and modify a file on the same PR, do so in two commits. This way git will always know what's going on and it will be easier to track changes across file renames.
- **Don't send pull requests with merge commits on them** Always rebase or cherry pick. If a commit on **devel** introduces merge conflicts in your branch, fix your commits by rebasing not by back merging and creating a conflict resolution commit.
- **If one of your commits fixes an issue, mention it** Add a "Closes #1234" line to the comment's body section (from line 3 onwards). This way a reference to this particular commit will be created on the issue itself and once the commit hits the target branch the issue will be closed automatically. If a whole PR is needed to close a particular issue, add the "Closes" comment on the PR body.
- **Capitalize the commit's subject** We are civilized people after all :D
- **Write concise commit messages** If a particular commit deserves a longer explanation, write a short commit message, leave a blank line after it and then go all Shakespeare from the third line (message body) onwards.

- **Read this** Really, do it.

CHAPTER 4

Setting up your development environment

This page contains instructions on how to setup your development environment to run Tribler from source.

4.1 Windows

This section contains information about setting up a Tribler development environment on Windows. Unlike Linux based systems where installing third-party libraries is often a single `apt-get` command, installing and configuring the necessary libraries requires more attention on Windows. Moreover, the Windows environment has different file stuctures. For instance, where Linux is working extensively with .so (shared object) files, Windows uses DLL files.

4.1.1 Introduction

In this guide, all required dependencies of Tribler will be explained. It presents how to install these dependencies. Some dependencies have to be built from source whereas other dependencies can be installed using a .msi installer. The guide targets Windows 10, 64-bit systems, however, it is probably not very hard to install 32-bit packages.

First, Python 2.7 should be installed. If you already have a Python version installed, please check whether this version is 64 bit before proceeding.

```
python -c "import struct;print( 8 * struct.calcsize('P'))"
```

This outputs whether your current installation is 32 or 64 bit.

Python can be downloaded from the official Python website. You should download the Windows x86-64 MSI Installer which is an executable. **During the setup, remember to install pip/setuptools and to add Python to the PATH variable to access Python from the command line. The option to add Python to the PATH variable is unchecked by default!** You can verify whether Python is installed correctly by typing `python` in the command line. Also check whether pip is working by typing `pip` in the command line. If they are not working, check whether the PATH variables are correctly set.

If you did not change the default installation location, Python should be located at `C:\\Python27\\`. The third-party libraries are located in `C:\\Python27\\Lib\\site-packages`. If you forgot to add Python to your PATH during the setup, you should need to add the `C:\\Python27\\` and `C:\\Python27\\Scripts` directories to your PATH variable.

In order to compile some of the dependencies of Tribler, you will need Visual Studio 2015 which can be downloaded from here. You should select the community edition. Visual Studio ships with a command line interface that can be used for building some of the Python packages. Moreover, it provides a nice IDE which can be used to work on Python projects. After installation of Visual Studio, you should install the Visual C++ tools. This can be done from within Visual Studio by creating a new Visual C++ project. Visual Studio then gives an option to install the Visual C++ developer tools.

In case importing one of the modules fail due to a DLL error, you can inspect if there are files missing by opening it with Dependency Walker. It should show missing dependencies. In our case, we were missing `MSVCR100.DLL` which belongs to the Microsoft Visual C++ 2010 SP1 Redistributable Package (x64). This package can be downloaded from the Microsoft website. One other DLL that was missing was `MSVCR110.DLL`, which belongs to the Visual C++ Redistributable for Visual Studio 2012 Update 4. After installing these two pakets, there should be no more import errors.

4.1.2 M2Crypto

The first package to be installed is M2Crypto which can be installed using pip (the M2Crypto binary is precompiled):

```
pip install --egg M2CryptoWin64 # use M2CryptoWin32 for the 32-bit version of M2Crypto
python -c "import M2Crypto" # test whether M2Crypto can be successfully imported
```

If the second statement does not raise an error, M2Crypto is successfully installed.

4.1.3 wxPython

The graphical interface of Tribler is built using wxPython. wxPython can be installed by using the official win64 installer for Python 2.7 from Sourceforge. **At the time of writing, wx3 is not supported yet so you should install wx2.8** (make sure to install the unicode version). You can test whether wx can be successfully imported by running:

```
python -c "import wx"
```

This statement should proceed without error.

4.1.4 pyWin32 Tools

In order to access some of the Windows API functions, pywin32 should be installed. The pywin32 installer can be downloaded from Sourceforge and make sure to select the amd64 version and the version compatible with Python 2.7.

## apsw The apsw (Another Python SQLite Wrapper) installer can be downloaded from GitHub. Again, make sure to select the amd64 version that is compatible with Python 2.7. You can test whether it is installed correctly by running:

```
python -c "import apsw"
```

4.1.5 libtorrent

This package should be compiled from source. First, install Boost which can be downloaded from SourceForge. Make sure to select the latest version and choose the version is compatible with your version of Visual C++ tools (probably msvc-14).

After installation, you should set an environment variable to let libtorrent know where Boost can be found. You can do this by going to Control Panel > System > Advanced > Environment Variables (more information about setting environment variables can be found here). Now add a variable named BOOST_ROOT and with the value of your Boost location. The default installation location for the Boost libraries is `C:\\local\\boost_<BOOST VERSION>` where `<BOOST VERSION>` indicates the installed Boost version.

Next, you should build Boost.build. You can do this by opening the Visual Studio command prompt and navigating to your Boost libraries. Navigate to `tools\\build` and execute `bootstrap.bat`. This will create the `b2.exe` file. In order to invoke `b2` from anywhere in your command line, you should add the Boost directory to your user PATH environment variable. After modifying your PATH, you should reopen your command prompt.

Now, download the libtorrent source code from GitHub and extract it. It is advised to compile version 1.0.8. Note that you if you have a 32-bit system, you can download the `.msi` installer so you do not have to compile libtorrent yourself. Open the Developer Command Prompt shipped with Visual Studio (not the regular command prompt) and navigate to the location where you extracted the libtorrent source. In the directory where the libtorrent source code is located, navigate to `bindings\\python` and build libtorrent by executing the following command (this takes a while so make sure to grab a coffee while waiting):

```
b2 boost=source libtorrent-link=static address-model=64
```

This command will build a static libtorrent 64-bit debug binary. You can also build a release binary by appending `release` to the command given above. After the build has been completed, the resulting `libtorrent.pyd` can be found in `LIBTORRENT_SOURCE\\bindings\\python\\bin\\msvc-14\\debug\\address-model-64\\boost-source\\l` where `LIBTORRENT_SOURCE` indicates the directory with the libtorrent source files. Copy `libtorrent.pyd` to your site-packages location (the default location is `C:\\Python27\\Lib\\site-packages`) and test libtorrent by executing:

```
python -c "import libtorrent"
```

4.1.6 libsodium

Libsodium can be download as precompiled binary from their website. Download the latest version, built with msvc. Extract the archive to any location on your machine. Next, you should add the location of the dynamic library to your `PATH` variables (either as system variable or as user variable). These library files can be found in `LIBSODIUM_ROOT\\x64\\Release\\v140\\dynamic\\` where `LIBSODIUM_ROOT` is the location of your extracted libsodium files. After modifying your PATH, you should reopen your command prompt. You test whether Python is able to load `libsodium.dll` by executing:

```
python -c "import ctypes; ctypes.cdll.LoadLibrary('libsodium')"
```

4.1.7 LevelDB

The next dependency to be installed is levelDB. LevelDB is a fast key-value storage written by Google. LevelDB itself is written in C++ but there are several Python wrappers available. In this guide, you will compile leveldb from source. First, download the source code from GitHub (either clone the repository or download the source code as zip). The readme on this repo contains some basic instructions on how to compile leveldb.

Next, open the `levedb_ext.sln` file in Visual Studio. This guide is based on the `x64 release` configuration. If you want to build a 32-bit leveldb project, change the configuration to `win32 release`.

You should edit the file paths of the include directories and the linker directories. These can be edited by right clicking on the project and selecting `properties`. You will need to update `additional include directories` (under C/C++ -> general) to point to your Python include directory (often located in `C:\\Python27\\include`). This is needed for the compilation of the Python bindings. Also, make sure that the following `preprocessor definitions` (found under C/C++ -> preprocessor) are defined: `WIN32` and `LEVELDB_PLATFORM_WINDOWS`.

Next, `additional library directories` should be adjusted, found under Linker -> General. You should add the directory where your Python libraries are residing, often in `C:\\Python27\\libs`.

Compile by pressing the `build leveldb_ext` in the build menu. If any errors are showing up during compilation, please refer to the Visual Studio log file and check what's going wrong. Often, this should be a missing include/linker directory. If compilation is successful, a `leveldb_ext.pyd` file should have been created in the project directory.

Copy this file to your site-packages location and rename it to `leveldb.pyd` so Python is able to find it. You can test whether your binary is working by using the following command which should execute without any errors:

```
python -c "import leveldb"
```

4.1.8 VLC

To install VLC, you can download the official installer from the VideoLAN website. Make sure to install the 64-bit version of VLC.

4.1.9 Additional Packages

There are some additional packages which should be installed. They can easily be installed using pip:

```
pip install cherrypy chardet configobj cryptography decorator feedparser netifaces pillow twisted
```

4.1.10 Running Tribler

You should now be able to run Tribler from command line. Grab a copy of the Tribler source code and navigate in a command line interface to the source code directory. Start Tribler by running:

```
python Tribler\Main\tribler.py
```

You might get errors about imports in the Tribler module. To fix this, you should add the location where the Tribler directory is located to the `PYTHONPATH` user environment variables. Information about changing environment variables can be found here.

If there are any problems with the guide above, please feel free to fix any errors or create an issue so we can look into it.

4.2 OS X

This section contains information about setting up a Tribler development environment on OS X. Unlike Linux based systems where installing third-party libraries is often a single `apt-get` command, installing and configuring the necessary libraries requires more attention on OS X. This guide has been tested with OS X 10.10.5 (Yosemite) but should also work for OS X 10.11 (El Capitan).

Note that the guide below assumes that Python is installed in the default location of Python (shipped with OS X). This location is normally in `/Library/Python/2.7`. Writing to this location requires root acccess when using easy_install or pip. To avoid root commands, you can install Python in a virtualenv. More information about setting up Python in a virtualenv can be found here.

4.2.1 Introduction

Compilation of C/C++ libraries should be performed using Clang which is part of the Xcode Command Line Tools. The Python version shipped with OS X can be used and this guide has been tested using Python 2.7. The current installed version and binary of Python can be found by executing:

```
python --version # gets the python version
which python # prints the path of the Python executable
```

Note that the default location of third-party Python libraries (for example, installed with `pip`) can be found in `/Library/Python/2.7/site-packages`.

Many packages can be installed by using the popular brew and pip executables. Brew and pip can be installed by using:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
sudo easy_install pip
```

This should be done after accepting the Xcode license so open Xcode at least once before installing Brew.

4.2.2 Xcode Tools

The installation of Xcode is required in order to compile some C/C++ libraries. Xcode is an IDE developed by Apple and can be downloaded for free from the Mac App Store. After installation, the Command Line Tools should be installed by executing:

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

4.2.3 WxPython

WxPython is the Graphical User Interface manager and an installer can be downloaded from their website. Note that at this point, Wx 2.8 should still be used but support for 2.8 will be dropped soon and the Wx 2.8 library should be replaced by Wx 3.0. You probably need the Cocoa version of Wx.

Note: there is a bug on OS X 10.11 (El Capitan) where the installer gives an error that there is no software available to install. A workaround for this is to install the required files manually. This can be done by opening the `.pkg` file. First, you should run the `preflight.sh` script as root to clean up any old installation of wx. Next, unzip the `wxPython3.0-osx-cocoa-py2.7.pax.gz` file. This will create a `usr` directory which should be copied to `/usr` on the system. Note that you need root permissions to write to this directory (you can open a finder window with the needed permissions by running `sudo open /usr` in terminal). To link wx so Python can find it, you should run the `postflight.sh` as root.

4.2.4 M2Crypto

To install M2Crypto, Openssl has to be installed first. The shipped version of openssl by Apple gives errors when compiling M2Crypto so a self-compiled version should be used. Start by downloading openssl 0.98 from here, extract it and install it:

```
./config --prefix=/usr/local
make && make test
sudo make install
openssl version # this should be 0.98
```

Also Swig 3.0.4 is required for the compilation of the M2Crypto library. The easiest way to install it, it to download Swig 3.0.4 from source here and compile it using:

```
./configure
make
sudo make install
```

Note: if you get an error about a missing PCRE library, install it with brew using `brew install pcre`.

Now we can install M2Crypto. First download the source (version 0.22.3 is confirmed to work on El Capitan and Yosemite) and install it:

```
python setup.py build build_ext --openssl=/usr/local
sudo python setup.py install build_ext --openssl=/usr/local
```

Reopen your terminal window and test it out by executing:

```
python -c "import M2Crypto"
```

4.2.5 Apsw

Apsw can be installed by brew but this does not seem to work to compile the last version (the Clang compiler uses the `sqlite.h` include shipped with Xcode which is outdated). Instead, the source should be downloaded from their Github repository (make sure to download a release version) and compiled using:

```
sudo python setup.py fetch --all build --enable-all-extensions install test
python -c "import apsw" # verify whether apsw is successfully installed
```

4.2.6 Libtorrent

An essential dependency of Tribler is libtorrent. libtorrent is dependent on Boost, a set of C++ libraries. Boost can be installed with the following command:

```
brew install boost
brew install boost-python
```

Now we can install libtorrent:

```
brew install libtorrent-rasterbar --with-python
```

After the installation, we should add a pointer to the `site-packages` of Python so it can find the new libtorrent library using the following command:

```
sudo echo 'import site; site.addsitedir("/usr/local/lib/python2.7/site-packages")' >> /Library/Python
```

This command basically adds another location for the Python site-packages (the location where libtorrent-rasterbar is installed). This command should be executed since the location where brew installs the Python packages is not in sys.path. You can test whether libtorrent is correctly installed by executing:

```
python
>>> import libtorrent
```

4.2.7 Other Packages

There are a bunch of other packages that can easily be installed using pip and brew:

```
brew install homebrew/python/pillow gmp mpfr libmpc libsodium
pip install --user cherrypy cffi chardet configobj cryptography decorator feedparser gmpy2 idna level
```

If you encounter any error during the installation of Pillow, make sure that libjpeg and zlib are installed. They can be installed using:

```
brew tap homebrew/dupes
brew install libjpeg zlib
brew link --force zlib
```

Tribler should now be able to startup without warnings by executing this command in the Tribler root directory:

```
./tribler.sh
```

If there are any missing packages, they can often be installed by one pip or brew command. If there are any problems with the guide above, please feel free to fix any errors or create an issue so we can look into it.

4.2.8 System Integrity Protection on El Capitan

The new security system in place in El Capitan can prevent `libsodium.dylib` from being dynamically linked into Tribler when running Python. If this library cannot be loaded, it gives an error that libsodium could not be found. This is because the `DYLD_LIBRARY_PATH` cannot be set when Python starts. More information about this can be read here.

There are two solutions for this problem. First, `libsodium.dylib` can symlinked into the Tribler root directory. This can be done by executing the following command **in the Tribler root directory**:

```
ln -s /usr/local/lib/libsodium.dylib
```

Now the `ctypes` Python library will be able to find the `libsodium.dylib` file.

The second solution is to disable SIP. This is not recommended since it makes the system more vulnerable for attacks. Information about disabling SIP can be found here.

4.3 Linux

This section contains information about setting up a Tribler development environment on Linux systems.

4.3.1 Debian/Ubuntu/Mint

Execute the following command in your terminal:

```
sudo apt-get install libav-tools libjs-excanvas libjs-mootools libsodium13 libx11-6 python-apsw pytho
sudo pip install decorator libnacl
```

4.3.2 Experimental support for Ubuntu 16.04

Please try if the latest experimental build works for you.

```
bash
sudo apt-get install libsodium-dev python-nacl
```

Next, download the latest .deb file from here.

4.3.3 Installing libsodium13 and python-cryptography on Ubuntu 14.04

While installing libsodium13 and python-cryptography on a clean Ubuntu 14.04 install (possibly other versions as well), the situation can occur where the Ubuntu terminal throws the following error when trying to install the dependencies mentioned earlier in the README.rst:

> E: Unable to locate package libsodium13 E: Unable to locate package python-cryptography

This means that the required packages are not directly in the available package list of Ubuntu 14.04.

To install the packages, the required files have to be downloaded from their respecive websites.

For libsodium13, download `libsodium13\_1.0.1-1\_<ProcessorType\>.deb` from http://packages.ubuntu.com/vivid/libsodium13](http://packages.ubuntu.com/vivid/libsodium13

For python-cryptography, download `python-cryptography\_0.8-1ubuntu2\_<ProcessorType\>.deb` from http://packages.ubuntu.com/vivid/python-cryptography.

Installing the files Through terminal

After downloading files go to the download folder and install the files through terminal:

For amd64:

```
cd ./Downloads
dpkg -i libsodium13_1.0.1-1_amd64.deb
dpkg -i python-cryptography_0.8-1ubuntu2_amd64.deb
```

For i386:

```
cd ./Downloads
dpkg -i libsodium13_1.0.1-1_i386.deb
dpkg -i python-cryptography_0.8-1ubuntu2_i386.deb
```

Through file navigator:

Using the file navigator to go to the download folder and by clicking on the .deb files to have the software installer install the packages.

Now installing the list of dependencies should no longer throw an error.

If there are any problems with the guide above, please feel free to fix any errors or create an issue so we can look into it.

4.3.4 Arch Linux

Execute the following command in your terminal:

```
sudo pacman -S libsodium libtorrent-rasterbar python2-apsw python2-cherrypy python2-cryptography pyth
```

CHAPTER 5

Building Tribler

This page contains instructions on how to build and package Tribler.

5.1 Windows

This section contains information about building Tribler on Windows. In the end you should be left with a `.exe` file which, when opened, enables users to install Tribler on their system. This guide installs a 64-bit version of Tribler and has been tested on Windows 10 and Windows 2008 Server R2, 64-bit. It is recommended to create this builder on a system that is already able to run Tribler from a git checkout (it means that all the required packages required by Tribler are installed already). In case you want to build a 32 bit version, just install all the dependencies mentioned in 32 bit version. Information about setting up a developer environment on Windows can be found on tribler_dev_windows.

When you have installed zope, an empty `__init__.py` file must be present in the zope folder. If this file is missing, a `No module named zope` error will be thrown. Create this file in the `site-packes/zope` folder if it does not exist.

5.1.1 Required packages

To build a Tribler installer, you'll need some additional scripts and packages. The versions used as of writing this guide are mentioned next to the package or script. * The git command tools (version 2.7.0) are required to fetch the latest release information. These can be downloaded from here. * Py2Exe (0.6.9), a tool to create an executeable from python files. Grab the latest version here. * The builder needs to find all packages that are required by Tribler so make sure you can run Tribler on your machine and that there are no missing dependencies. * Nullsoft Scriptable Install System (NSIS) (version 2.5.0) is a script-driven Installer authoring tool for Microsoft Windows with minimal overhead. It can be downloaded here. We selected version 2.5 as the uninstall functions were not called properly in 3.03b. * Three plugins are required.The UAC plugin is the first. This can be downloaded from here (version 0.2.4c). How to install a plugin can be found here. * The second plugin that is needed is AccessControl plug-in (version 1.0.8.1). It can be downloaded here. * The third plugin required is NSIS Simple Firewall Plugin (version 1.2.0). You can download it here. * The fourth plugin needed is NSProcess (Version 1.6.7), which can be downloaded here. * A version of Microsoft Visual Studio should be installed (we use 2012), but make sure you do not have the build-tools only. The full (community) edition can be downloaded here.

5.1.2 Building & Packaging Tribler

Start by cloning Tribler if you haven't done already (using the `git clone --recursive` command). Next, create a `build` folder directly on your `C:\` drive. Inside the `build` folder, put the following items:

1. A static version (64 bit, git-1d8f9b7) of ffmpeg, available here. Place it in a folder called `ffmpeg` in the `build` folder.
2. A folder `certs` containing a `.pfx` key. In our case it's named `swarmplayerprivatekey.pfx`. Make sure to rename paths in `makedist_win.bat` to match your file name.
3. A `vlc` folder containing a full instalation of vlc (Version 2.2.1).
4. `vc_redist_90.exe` (Microsoft Visual C++ 2008 Redistributable Package), which is available here. In case you build 32 bit, get the x86 version here. Don't forget to rename the file.
5. `vc_redist_110.exe` (Visual C++ Redistributable for Visual Studio 2012), which is available here. In case you build 32 bit, get the x86 version. Once more, don't forget to rename the file.
6. *libsodium.dll* which can be downloaded from libsodium.org (as of writing version 1.0.8).

Then, set a `PASSWORD` environment variable with its value set to the password matching the one set in your `.pfx` file.

Finally, open a command prompt and enter the following commands (Change 11.0 depending on your version of Microsoft Visual Studio): Note that for building 32 bit you need to pass anything but 64, i.e. 32 or 86 to the `update_version_from_git.py` script.

```
setlocal enabledelayedexpansion
call "C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\vcvarsall.bat"
SET PATH=%PATH%;C:\Windows\system32;C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\bin
dir
cd tribler
python Tribler/Main/Build/update_version_from_git.py 64
xcopy C:\build\vlc vlc /E /I
win\makedist_win.bat
```

This builds an `.exe` installer which installs Tribler when ran.

5.2 OS X

This section contains information about building Tribler on OS X. The final result you should have is a `.dmg` file which, when opened, allows `Tribler.app` to be copied to the Applications directory. This guide has been tested on OS X 10.11 (El Capitan). It is recommended to run this builder on a system that is already able to run Tribler without problems (it means that all the required packages required by Tribler are installed already). Information about setting up a developer environment on OS X can be found here.

5.2.1 Required packages

To build and distribute Tribler, there are some required scripts and packages: * The git command tools are required to fetch the latest release information. They are installed when you start Xcode for the first time but you can also install it using `brew` or another package library. * Py2app. The built-in version of py2app does not function correctly when System Integrity Protection (SIP) is turned on. You can either turn SIP off (instructions on how to do this can be found here) or you can install a more recent version of py2app using PIP in your user-defined `site-packages`. Note that you should place the `site-packages` directory with py2app in it higher in the `PYTHONPATH` environment variable than the `site-packages` directory managed by the system. Otherwise, the builder will chooose the py2app package installed by the system. * The builder needs to find all packages that are required by Tribler so make sure you can run Tribler on your machine and that there are no missing dependencies. * In order to attach the EULA to the `.dmg` file, we make use of the `eulagise` script. This script is written in PERL and is based on a more fully-featured script. The script can be dowloaded from GitHub. The builder expects the script to be executable and added to the `PATH` environment variable. This can be done with the following commands:

```
cp eulagise.pl /usr/local/bin/eulagise
chmod +x /usr/local/bin/eulagise
eulagise # to test it - it should show that you should add some flags
```

5.2.2 Building Tribler on OS X

Start by checking out the directory you want to clone (using `git clone --recursive`). Open a terminal and `cd` to this new cloned directory (referenced to as `tribler_source` in this guide).

First we need to copy the ffmpeg library to `tribler_source`. You can download this file from here. Next, create a directory named `vlc` in `tribler_source` and copy the `ffmpeg` file to that directory. Make sure to name the file `ffmpeg`, otherwise the builder cannot find it.

Next, we should inject version information into the files about the latest release. This is done by the `update_version_from_git.py` script found in `Tribler/Main/Build`. Invoke it from the `tribler_source` directory by executing:

```
Tribler/Main/Build/update_version_from_git.py
```

Before we can build the `.dmg` file, some environment variables need to be set:

```
export MACOSX_DEPLOYMENT_TARGET=10.11
export CFLAGS=' -mmacosx-version-min=10.6 -O -g '
export CXXFLAGS=' -mmacosx-version-min=10.6 -O -g '
```

If you are building on another environment, you should change `MACOSX_DEPLOYMENT_TARGET` to match your version of OS X. The `-mmacosx-version-min` is required so the builder can optimize the build depending on the minimum supported version.

Now execute the builder with the following command:

```
./mac/makedistmac_64bit.sh
```

This will create the `.dmg` file in the `tribler_source/dist` directory.

5.3 Debian and derivatives

Run the following commands in your terminal:

```
sudo apt-get install devscripts python-setuptools
cd tribler
Tribler/Main/Build/update_version_from_git.py
debuild -i -us -uc -b
```

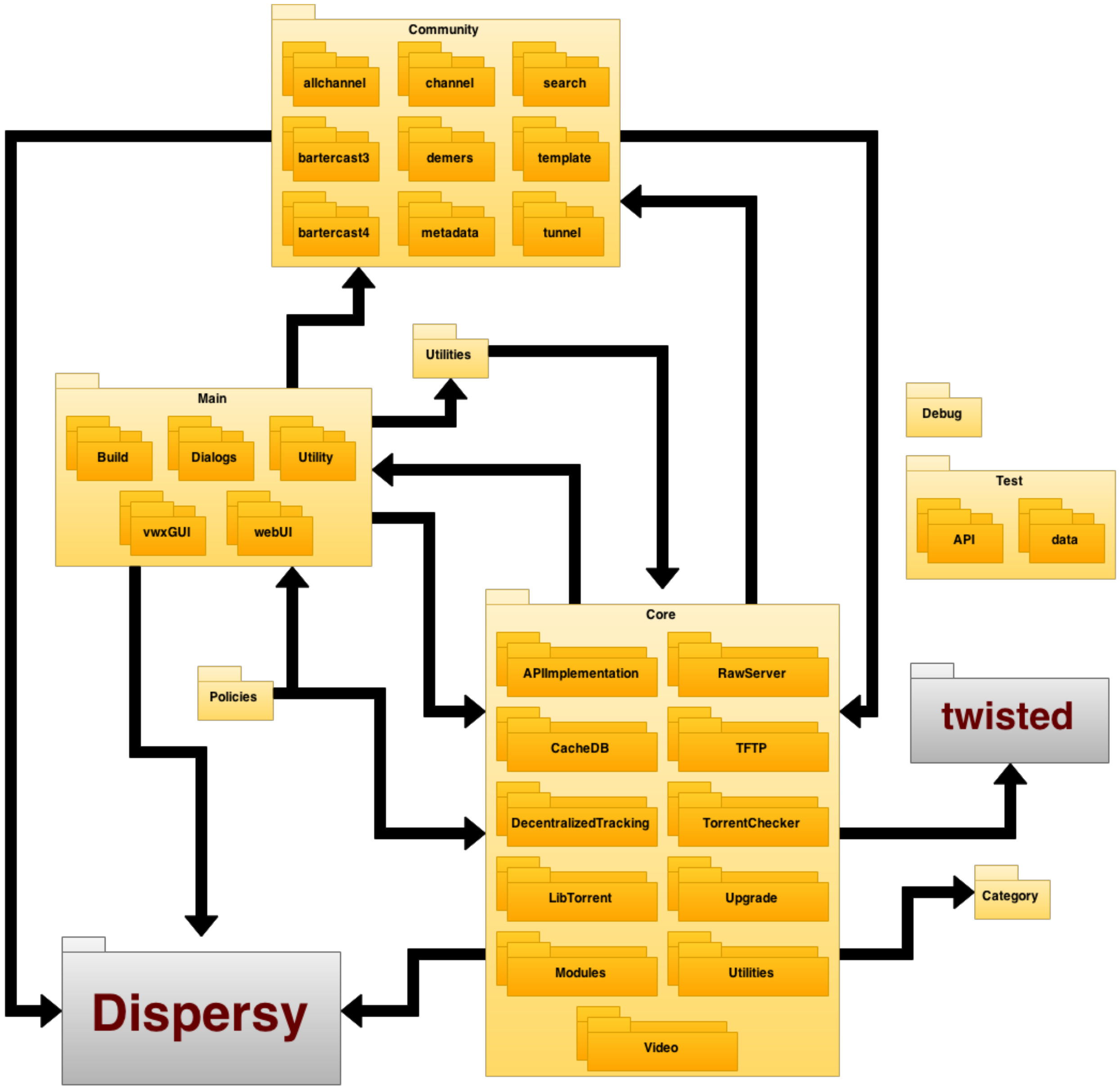
5.4 Other Unixes

We don't have a generic setup.py yet.

So for the time being, the easiest way to package Tribler is to put `Tribler/` in `/usr/share/tribler/` and `debian/bin/tribler` in `/usr/bin/`. A good reference for the dependency list is `debian/control`.

CHAPTER 6

Architectural Structure



6.1 Modules

`Tribler.Category`	
`Tribler.Core`	The Core package contains the core functionalities of the Tribler project
`Tribler.Core.APIImplementation`	The APIImplementation contains the API for torrents, creating them and mainta
`Tribler.Core.CacheDB`	The CacheDB package contains the cachedDB for Tribler including a notifier an
`Tribler.Core.Config`	This package contains code for the configuration objects.
`Tribler.Core.DecentralizedTracking`	The DecentralizedTracking package contains the pymdht library for profiling ne
`Tribler.Core.Libtorrent`	The Libtorrent package contains code to manage the torrent library.
`Tribler.Core.Modules`	The Modules package contains the tracker manager
`Tribler.Core.Modules.channel`	
`Tribler.Core.Modules.restapi`	This package contains code for the Tribler HTTP API.
`Tribler.Core.TFTP`	Contains the the TFTP handler that should be registered at the thread pool to har
`Tribler.Core.TorrentChecker`	The TorrentChecker package contains code that checks and schedules torrents.
`Tribler.Core.Upgrade`	The upgrade package contains information on how to update Tribler and its data
`Tribler.Core.Utilities`	The Utilities package contains different utility files that are used in the project
`Tribler.Core.Video`	The Video package contains the VLCWrapper and code to stream live data in Tr
`Tribler.Main`	The Main package contains code for the GUI of Tribler.
`Tribler.Main.Utility`	The Utility package contains different utility classes used in the Main package
`Tribler.Test`	The Test package contains unit tests for tribler.
`Tribler.Test.API`	The API package contains tests for the API.
`Tribler.Test.Category`	
`Tribler.Test.Core`	This package contains unit tests for the Tribler core.
`Tribler.community.allchannel`	The allchannel package contains the AllChannel community which is used to co
`Tribler.community.bartercast4`	The bartercast4 package TTTODO: what does this do?
`Tribler.community.channel`	The channel package contains a Dispersy community which is used to implemen
`Tribler.community.demers`	The demers package contains a Dispersy community which can be used to verify
`Tribler.community.multichain`	
`Tribler.community.search`	The search package contains a Dispersy community which is used to implement
`Tribler.community.template`	The template package contains Example files for communities
`Tribler.community.tunnel`	The Tunnel community package

6.2 Maintenance

Inside the Resources folder, there is a .xml file that can be opened in Draw.io, where it can be adapted/altered as needed.

Tribler REST API

7.1 Overview

The Tribler REST API allows you to create your own applications with the channels, torrents and other data that can be found in Tribler. Moreover, you can control Tribler and add data to Tribler using various endpoints. This documentation explains the format and structure of the endpoints that can be found in this API. **Note that this API is currently under development and more endpoints will be added over time**.

7.2 Making requests

The API has been built using Twisted Web. Requests go over HTTP where GET requests should be used when data is fetched from the Tribler core and POST requests should be used if data in the core is manipulated (such as adding a torrent or removing a download). Responses of the requests are in JSON format. Tribler should be running either headless or with the GUI before you can use this API.

Some requests require one or more parameters. These parameters are passed using the JSON format. An example of performing a request with parameters using the curl command line tool can be found below:

```
curl -X PUT http://localhost:8085/mychannel/rssfeeds/http%3A%2F%2Frssfeed.com%2Frss.xml
```

7.3 Error handling

If an unhandled exception occurs the response will have code HTTP 500 and look like this:

```
{
    "error": {
        "handled": False,
        "code": "SomeException",
        "message": "Human readable error message"
    }
}
```

If a valid request of a client caused a recoverable error the response will have code HTTP 500 and look like this:

```
{
    "error": {
        "handled": True,
        "code": "DuplicateChannelNameError",
        "message": "Channel name already exists: foo"
    }
}
```

7.4 Download states

There are various download states possible which are returned when fetching downloads. These states are explained in the table below.

DLSTATUS_ALLOCATING_DISKSPACE	Libtorrent is allocating disk space for the download
DLSTATUS_WAITING4HASHCHECK	The download is waiting for the hash check to be performed
DLSTATUS_HASHCHECKING	Libtorrent is checking the hashes of the download
DLSTATUS_DOWNLOADING	The torrent is being downloaded
DLSTATUS_SEEDING	The torrent has been downloaded and is now being seeded to other peers
DLSTATUS_STOPPED	The torrent has stopped downloading, either because the downloading has completed or the user has stopped the download
DLSTATUS_STOPPED_ON_ERROR	The torrent has stopped because an error occurred
DLSTATUS_METADATA	The torrent information is being fetched from the DHT
DLSTATUS_CIRCUITS	The (anonymous) download is building circuits

7.5 Endpoints

7.5.1 Discovered channels

class `Tribler.Core.Modules.restapi.channels.channels_discovered_endpoint.`**`ChannelsDiscoveredEn`**

This class is responsible for requests regarding the discovered channels.

`render_GET``(_)`

`GET /channels/discovered`

A GET request to this endpoint returns all channels discovered in Tribler.

Example request:

```
curl -X GET http://localhost:8085/channels/discovered
```

Example response:

```
{
    "channels": [{
        "id": 3,
        "dispersy_cid": "da69aaad39ccf468aba2ab9177d5f8d8160135e6",
        "name": "My fancy channel",
        "description": "A description of this fancy channel",
        "subscribed": False,
        "votes": 23,
        "torrents": 3,
        "spam": 5,
        "modified": 14598395,
        "can_edit": True
    }, ...]
}
```

render_PUT(*request*)

PUT /channels/discovered

Create your own new channel. The passed mode and descriptions are optional. Valid modes include: 'open', 'semi-open' or 'closed'. By default, the mode of the new channel is 'closed'.

Example request:

```
curl -X PUT http://localhost:8085/channels/discovered
--data "name=fancy name&description=fancy description&mode=open"
```

Example response:

```
{
    "added": 23
}
```

statuscode 500 if a channel with the specified name already exists.

class Tribler.Core.Modules.restapi.channels.channels_discovered_endpoint.**ChannelsDiscoveredSp**

This class is responsible for dispatching requests to perform operations in a specific discovered channel.

render_GET(*request*)

GET /channels/discovered/(string: channelid)

Return the name, description and identifier of a channel.

Example request:

```
curl -X GET http://localhost:8085/channels/discovered/4a9cfc7ca9d15617765f4151dd9fae94c8
```

Example response:

```
{
    "overview": {
        "name": "My Tribler channel",
        "description": "A great collection of open-source movies",
        "identifier": "4a9cfc7ca9d15617765f4151dd9fae94c8f3ba11"
    }
}
```

statuscode 404 if your channel has not been created (yet).

7.5.2 Subscribed channels

class `Tribler.Core.Modules.restapi.channels.channels_subscription_endpoint.`**`ChannelsModifySubs`**

This class is responsible for methods that modify the list of RSS feed URLs (adding/removing feeds).

`render_DELETE`(*request*)

`DELETE /channels/subscribed/(string: channelid)`

Unsubscribe from a specific channel. Returns error 404 if you are not subscribed to this channel.

Example request:

```
curl -X DELETE http://localhost:8085/channels/subscribed/da69aaad39ccf468aba2ab9177d5f8d
```

Example response:

```
{
    "unsubscribed" : True
}
```

statuscode 404 if you are not subscribed to the specified channel.

`render_PUT`(*request*)

`PUT /channels/subscribed/(string: channelid)`

Subscribe to a specific channel. Returns error 409 if you are already subscribed to this channel.

Example request:

```
curl -X PUT http://localhost:8085/channels/subscribed/da69aaad39ccf468aba2ab9177d5f8d816
```

Example response:

```
{
    "subscribed" : True
}
```

statuscode 409 (conflict) if you are already subscribed to the specified channel.

class `Tribler.Core.Modules.restapi.channels.channels_subscription_endpoint.`**`ChannelsSubscribed`**

This class is responsible for requests regarding the subscriptions to channels.

`render_GET`(_)

`GET /channels/subscribed`

Returns all the channels the user is subscribed to.

Example request:

```
curl -X GET http://localhost:8085/channels/subscribed
```

Example response:

```
{
    "subscribed": [{
        "id": 3,
        "dispersy_cid": "da69aaad39ccf468aba2ab9177d5f8d8160135e6",
        "name": "My fancy channel",
        "description": "A description of this fancy channel",
        "subscribed": True,
        "votes": 23,
        "torrents": 3,
        "spam": 5,
        "modified": 14598395,
        "can_edit": True,
    }, ...]
}
```

7.5.3 Popular channels

7.5.4 My channel

class `Tribler.Core.Modules.restapi.channels.my_channel_endpoint.`**`MyChannelEndpoint`**(*session*)

This class is responsible for managing requests regarding your channel.

`render_GET`(*request*)

`GET /mychannel`

Return the name, description and identifier of your channel. This endpoint returns a 404 HTTP response if you have not created a channel (yet).

Example request:

```
curl -X GET http://localhost:8085/mychannel
```

Example response:

```
{
    "overview": {
        "name": "My Tribler channel",
        "description": "A great collection of open-source movies",
        "identifier": "4a9cfc7ca9d15617765f4151dd9fae94c8f3ba11"
    }
}
```

statuscode 404 if your channel has not been created (yet).

`render_POST`(*request*)

`POST /mychannel`

Modify the name and/or the description of your channel. This endpoint returns a 404 HTTP response if you have not created a channel (yet).

Example request:

```
curl -X POST http://localhost:8085/mychannel
--data "name=My fancy playlist&description=This playlist contains some random movies"
```

Example response:

```
{
    "modified": True
}
```

statuscode 404 if your channel has not been created (yet).

7.5.5 RSS Feeds

class `Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.`**`ChannelModifyRssFeedEndpoir`**

This class is responsible for methods that modify the list of RSS feed URLs (adding/removing feeds).

`render_DELETE`(*request*)

`DELETE /channels/discovered/(string: channelid)/rssfeeds/http%3A%2F%2Ftest.com%2Frs`

Delete a RSS feed from your channel. Returns error 404 if the RSS feed that is being removed does not exist. Note that the rss feed url should be URL-encoded.

Example request:

```
curl -X DELETE http://localhost:8085/channels/discovered/abcd/rssfeeds/http%3A%2F%2Ftest
```

Example response:

```
{
    "removed": True
}
```

statuscode 404 if the specified RSS URL is not in your feed list.

`render_PUT`(*request*)

`PUT /channels/discovered/(string: channelid)/rssfeeds/http%3A%2F%2Ftest.com%2Frss.x`

Add a RSS feed to your channel. Returns error 409 if the supplied RSS feed already exists. Note that the rss feed url should be URL-encoded.

Example request:

```
curl -X PUT http://localhost:8085/channels/discovered/abcd/rssfeeds/http%3A%2F%2Ftest.com
```

Example response:

```
{
    "added": True
}
```

statuscode 409 (conflict) if the specified RSS URL is already present in your feeds.

class `Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.`**`ChannelsRecheckFeedsEndpoir`**

This class is responsible for handling requests regarding refreshing rss feeds in your channel.

`render_POST`(*request*)

`POST /channels/discovered/(string: channelid)/recheckfeeds`

Rechecks all rss feeds in your channel. Returns error 404 if you channel does not exist.

Example request:

```
curl -X POST http://localhost:8085/channels/discovered/recheckrssfeeds
```

Example response:

```
{
    "rechecked": True
}
```

statuscode 404 if you have not created a channel.

class `Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.`**`ChannelsRssFeedsEndpoint`**(*se*
cic

This class is responsible for handling requests regarding rss feeds in a channel.

`render_GET`(*request*)

`GET /channels/discovered/(string: channelid)/rssfeeds`

Returns the RSS feeds in your channel.

```
curl -X GET http://localhost:8085/channels/discovered/abcd/rssfeeds
```

Example response:

```
{
    "rssfeeds": [{
        "url": "http://rssprovider.com/feed.xml",
    }, ...]
}
```

7.5.6 Torrents

class `Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint.`**`ChannelModifyTorrentEr`**

This class is responsible for methods that modify the list of torrents (adding/removing torrents).

`render_DELETE`(*request*)

`DELETE /channels/discovered/(string: channelid)/torrents/(string: torrent infohash)`

Remove a torrent with a given infohash from a given channel.

Example request:

```
curl -X DELETE http://localhost:8085/channels/discovered/abcdefg/torrents/
97d2d8f5d37e56cfaeaae151d55f05b077074779
```

Example response:

```
{
    "removed": True
}
```

statuscode 404 if the channel is not found or if the torrent is not found in the specified channel

render_PUT(*request*)

PUT /channels/discovered/(string: channelid)/torrents/http%3A%2F%2Ftest.com%2Ftest.

Add a torrent by magnet or url to your channel. Returns error 500 if something is wrong with the torrent file and DuplicateTorrentFileError if already added to your channel (except with magnet links).

Example request:

```
curl -X PUT http://localhost:8085/channels/discovered/abcdefg/torrents/
http%3A%2F%2Ftest.com%2Ftest.torrent --data "description=nice video"
```

Example response:

```
{
    "added": "http://test.com/test.torrent"
}
```

statuscode 404 if your channel does not exist.

statuscode 500 if the specified torrent is already in your channel.

class Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint.**ChannelsTorrentsEndpoi**

This class is responsible for managing requests regarding torrents in a channel.

render_GET(*request*)

GET /channels/discovered/(string: channelid)/torrents

A GET request to this endpoint returns all discovered torrents in a specific channel. The size of the torrent is in number of bytes. The last_tracker_check value will be 0 if we did not check the tracker state of the torrent yet. Optionally, we can disable the family filter for this particular request by passing the following flag: - disable_filter: whether the family filter should be disabled for this request (1 = disabled)

Example request:

```
curl -X GET http://localhost:8085/channels/discovered/da69aaad39ccf468aba2ab9177d5f8d816
```

Example response:

```
        {
            "torrents": [{
                "id": 4,
                "infohash": "97d2d8f5d37e56cfaeaae151d55f05b077074779",
                "name": "Ubuntu-16.04-desktop-amd64",
                "size": 8592385,
                "category": "other",
                "num_seeders": 42,
                "num_leechers": 184,
                "last_tracker_check": 1463176959
            }, ...]
        }
```

statuscode 404 if the specified channel cannot be found.

render_PUT(*request*)

PUT /channels/discovered/(string: channelid)/torrents

Add a torrent file to your own channel. Returns error 500 if something is wrong with the torrent file and DuplicateTorrentFileError if already added to your channel. The torrent data is passed as base-64 encoded string. The description is optional.

Example request:

```
        curl -X PUT http://localhost:8085/channels/discovered/abcd/torrents
        --data "torrent=...&description=funny video"
```

Example response:

```
        {
            "added": True
        }
```

statuscode 404 if your channel does not exist.

statuscode 500 if the passed torrent data is corrupt.

7.5.7 Torrent info

class Tribler.Core.Modules.restapi.torrentinfo_endpoint.**TorrentInfoEndpoint**(*session*)

This endpoint is responsible for handing all requests regarding torrent info in Tribler.

render_GET(*request*)

GET /torrentinfo

A GET request to this endpoint will return information from a torrent found at a provided URI. This URI can either represent a file location, a magnet link or a HTTP(S) url. - torrent: the URI of the torrent file that should be downloaded. This parameter is required.

Example request:

```
            curl -X PUT http://localhost:8085/torrentinfo?torrent=file:/home/me/test.torrent
```

Example response:

```
{"metainfo": <torrent metainfo dictionary>}
```

7.5.8 Playlists

class `Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint.`**`ChannelsModifyPlaylis`**

This class is responsible for requests that are modifying a specific playlist in a channel.

`render_DELETE``(`*request*`)`

`DELETE /channels/discovered/(string: channelid)/playlists/(int: playlistid)`

Remove a playlist with a specified playlist id.

Example request:

```
curl -X DELETE http://localhost:8085/channels/discovered/abcd/playlists/3
```

Example response:

```
{
    "removed": True
}
```

statuscode 404 if the specified channel (community) or playlist does not exist

`render_POST``(`*request*`)`

`POST /channels/discovered/(string: channelid)/playlists/(int: playlistid)`

Edit a specific playlist. The new name and description should be passed as parameter.

Example request:

```
curl -X POST http://localhost:8085/channels/discovered/abcd/playlists/3
--data "name=test&description=my test description"
```

Example response:

```
{
    "modified": True
}
```

statuscode 404 if the specified channel (community) or playlist does not exist or if the

name and description parameters are missing.

class `Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint.`**`ChannelsPlaylistsEndp`**

This class is responsible for handling requests regarding playlists in a channel.

`render_GET``(`*request*`)`

GET /channels/discovered/(string: channelid)/playlists

Returns the playlists in your channel. Returns error 404 if you have not created a channel.

Example request:

```
curl -X GET http://localhost:8085/channels/discovered/abcd/playlists
```

Example response:

```
{
    "playlists": [{
        "id": 1,
        "name": "My first playlist",
        "description": "Funny movies",
        "torrents": [{
            "id": 4,
            "infohash": "97d2d8f5d37e56cfaeaae151d55f05b077074779",
            "name": "Ubuntu-16.04-desktop-amd64",
            "size": 8592385,
            "category": "other",
            "num_seeders": 42,
            "num_leechers": 184,
            "last_tracker_check": 1463176959
        }, ... ]
    }, ...]
}
```

statuscode 404 if you have not created a channel.

render_PUT(*request*)

PUT /channels/discovered/(string: channelid)/playlists

Create a new empty playlist with a given name and description. The name and description parameters are mandatory.

Example request:

```
curl -X PUT http://localhost:8085/channels/discovered/abcd/playlists
--data "name=My fancy playlist&description=This playlist contains some random movies"
```

Example response:

```
{
    "created": True
}
```

statuscode 400 if you are missing the name and/or description parameter

statuscode 404 if the specified channel does not exist

7.5.9 Downloads

class `Tribler.Core.Modules.restapi.downloads_endpoint.`**`DownloadExportTorrentEndpoint`**(*session*, *infohash*)

This class is responsible for requests that are exporting a download to a .torrent file.

`render_GET`(*request*)

`GET /download/(string: infohash)/torrent`

A GET request to this endpoint returns the .torrent file associated with the specified download.

Example request:

```
curl -X GET http://localhost:8085/downloads/4344503b7e797ebf31582327a5baae35b11bda01/
```

Example response:

The contents of the .torrent file.

class `Tribler.Core.Modules.restapi.downloads_endpoint.`**`DownloadsEndpoint`**(*session*)

This endpoint is responsible for all requests regarding downloads. Examples include getting all downloads, starting, pausing and stopping downloads.

`render_GET`(*request*)

`GET /downloads?get_peers=(boolean: get_peers)&get_pieces=(boolean: get_pieces)`

A GET request to this endpoint returns all downloads in Tribler, both active and inactive. The progress is a number ranging from 0 to 1, indicating the progress of the specific state (downloading, checking etc). The download speeds have the unit bytes/sec. The size of the torrent is given in bytes. The estimated time assumed is given in seconds. A description of the possible download statuses can be found in the REST API documentation.

Detailed information about peers and pieces is only requested when the get_peers and/or get_pieces flag is set. Note that setting this flag has a negative impact on performance and should only be used in situations where this data is required.

Example request:

```
curl -X GET http://localhost:8085/downloads?get_peers=1&get_pieces=1
```

Example response:

```
{
    "downloads": [{
        "name": "Ubuntu-16.04-desktop-amd64",
        "progress": 0.31459265,
        "infohash": "4344503b7e797ebf31582327a5baae35b11bda01",
        "speed_down": 4938.83,
        "speed_up": 321.84,
        "status": "DLSTATUS_DOWNLOADING",
        "size": 89432483,
        "eta": 38493,
        "num_peers": 53,
        "num_seeds": 93,
        "files": [{
```

```
                    "index": 0,
                    "name": "ubuntu.iso",
                    "size": 89432483,
                    "included": True
                }, ...],
                "trackers": [{
                    "url": "http://ipv6.torrent.ubuntu.com:6969/announce",
                    "status": "Working",
                    "peers": 42
                }, ...],
                "hops": 1,
                "anon_download": True,
                "safe_seeding": True,
                "max_upload_speed": 0,
                "max_download_speed": 0,
                "destination": "/home/user/file.txt",
                "availability": 1.234,
                "peers": [{
                    "ip": "123.456.789.987",
                    "dtotal": 23,
                    "downrate": 0,
                    "uinterested": False,
                    "wstate": "\",
                    "optimistic": False,
                    ...
                }, ...],
                "total_pieces": 420,
                "vod_mod": True,
                "vod_prebuffering_progress": 0.89,
                "vod_prebuffering_progress_consec": 0.86
            }
        }, ...]
```

render_PUT(*request*)

PUT /downloads

A PUT request to this endpoint will start a download from a provided URI. This URI can either represent a file location, a magnet link or a HTTP(S) url. - anon_hops: the number of hops for the anonymous download. 0 hops is equivalent to a plain download - safe_seeding: whether the seeding of the download should be anonymous or not (0 = off, 1 = on) - destination: the download destination path of the torrent - torrent: the URI of the torrent file that should be downloaded. This parameter is required.

Example request:

```
curl -X PUT http://localhost:8085/downloads
--data "anon_hops=2&safe_seeding=1&destination=/my/dest/on/disk/&uri=file:/home/me/te
```

Example response:

```
{"started": True, "infohash": "4344503b7e797ebf31582327a5baae35b11bda01"}
```

7.5.10 Search

class Tribler.Core.Modules.restapi.search_endpoint.**SearchCompletionsEndpoint**(*session*)
This class is responsible for managing requests regarding the search completions terms of a query.

render_GET(*request*)

GET /search/completions?q=(**string:** *query*)

A GET request to this endpoint will return autocompletion suggestions for the given query. For instance, when searching for "pioneer", this endpoint might return "pioneer one" if that torrent is present in the local database. This endpoint can be used to suggest terms to users while they type their search query.

Example request:

```
curl -X GET http://localhost:8085/search/completions?q=pioneer
```

Example response:

```
{
    "completions": ["pioneer one", "pioneer movie"]
}
```

class Tribler.Core.Modules.restapi.search_endpoint.**SearchEndpoint**(*session*)

This endpoint is responsible for searching in channels and torrents present in the local Tribler database. It also fires a remote search in the Dispersy communities.

render_GET(*request*)

GET /search?q=(**string:** *query*)

A GET request to this endpoint will create a search. Results are returned over the events endpoint, one by one. First, the results available in the local database will be pushed. After that, incoming Dispersy results are pushed. The query to this endpoint is passed using the url, i.e. /search?q=pioneer.

Example request:

```
curl -X GET http://localhost:8085/search?q=tribler
```

Example response:

```
{
    "type": "search_result_channel",
    "query": "test",
    "result": {
        "id": 3,
        "dispersy_cid": "da69aaad39ccf468aba2ab9177d5f8d8160135e6",
        "name": "My fancy channel",
        "description": "A description of this fancy channel",
        "subscribed": True,
        "votes": 23,
        "torrents": 3,
        "spam": 5,
        "modified": 14598395,
        "can_edit": False
    }
}
```

7.5.11 State info

class `Tribler.Core.Modules.restapi.state_endpoint.`**`StateEndpoint`**(*session*)
This endpoint is responsible for handing all requests regarding the state of Tribler.

`render_GET`(*request*)

`GET /state`

A GET request to this endpoint returns the current state of the Tribler core. There are three states: - STARTING: The core of Tribler is starting - UPGRADING: The upgrader is active - STARTED: The Tribler core has started

Example request:

```
curl -X GET http://localhost:8085/state
```

Example response:

```
{
    "state": "STARTED",
    "last_exception": None
}
```

7.5.12 Variables

class `Tribler.Core.Modules.restapi.variables_endpoint.`**`VariablesEndpoint`**(*session*)
This endpoint is responsible for handing all requests regarding runtime-defined variables in Tribler such as ports.

`render_GET`(*request*)

`GET /variables`

A GET request to this endpoint returns all the runtime-defined variables in Tribler.

Example request:

```
curl -X GET http://localhost:8085/variables
```

Example response:

```
{
    "variables": {
        "ports": {
            "video~port": 1234,
            "tunnel_community~socks5_listen_ports~1": 1235,
            ...
        },
        ...
    }
}
```

7.5.13 Settings

class `Tribler.Core.Modules.restapi.settings_endpoint.`**`SettingsEndpoint`**(*session*)

This endpoint is reponsible for handing all requests regarding settings and configuration.

`parse_setting`(*section*, *option*, *value*)

Set a specific Tribler setting. Throw a ValueError if this setting is not available.

`parse_settings_dict`(*settings_dict*, *depth=1*, *root_key=None*)

Parse the settings dictionary. Throws an error if the options dictionary seems to be invalid (i.e. there are keys not available in the configuration or the depth of the dictionary is too high.

`render_GET`(*request*)

`GET /settings`

A GET request to this endpoint returns all the session settings that can be found in Tribler. Please note that a port with a value of -1 means that the port is randomly assigned at startup.

Example request:

```
curl -X GET http://localhost:8085/settings
```

Example response:

```
{
    "settings": {
        "barter_community": {
            "enabled": false
        },
        "libtorrent": {
            "anon_listen_port": -1,
            ...
        },
        ...
    }
}
```

`render_POST`(*request*)

`POST /settings`

A POST request to this endpoint will update Tribler settings. A JSON-dictionary should be passed as body contents.

Example request:

```
curl -X POST http://localhost:8085/settings --data "{"
```

Example response:

```
{
    "modified": True
}
```

7.5.14 Events

class `Tribler.Core.Modules.restapi.events_endpoint.`**`EventsEndpoint`**(*session*)

Important events in Tribler are returned over the events endpoint. This connection is held open. Each event is pushed over this endpoint in the form of a JSON dictionary. Each JSON dictionary contains a type field that indicates the type of the event. Individual events are separated by a newline character (

).

Currently, the following events are implemented:

- events_start: An indication that the event socket is opened and that the server is ready to push events. This includes information about whether Tribler has started already or not and the version of Tribler used.
- search_result_channel: This event dictionary contains a search result with a channel that has been found.
- search_result_torrent: This event dictionary contains a search result with a torrent that has been found.
- upgrader_started: An indication that the Tribler upgrader has started.
- upgrader_finished: An indication that the Tribler upgrader has finished.
- upgrader_tick: An indication that the state of the upgrader has changed. The dictionary contains a human-readable string with the new state.
- watch_folder_corrupt_torrent: This event is emitted when a corrupt .torrent file in the watch folder is found. The dictionary contains the name of the corrupt torrent file.
- new_version_available: This event is emitted when a new version of Tribler is available.
- tribler_started: An indicator that Tribler has completed the startup procedure and is ready to use.
- channel_discovered: An indicator that Tribler has discovered a new channel. The event contains the name, description and dispersy community id of the discovered channel.
- torrent_discovered: An indicator that Tribler has discovered a new torrent. The event contains the infohash, name, list of trackers, list of files with name and size, and the dispersy community id of the discovered torrent.
- torrent_finished: A specific torrent has finished downloading. The event includes the infohash and name of the torrent that has finished downloading.
- torrent_error: An error has occurred during the download process of a specific torrent. The event includes the infohash and a readable string of the error message.
- tribler_exception: An exception has occurred in Tribler. The event includes a readable string of the error.

`render_GET`(*request*)

`GET /events`

A GET request to this endpoint will open the event connection.

Example request:

```
curl -X GET http://localhost:8085/events
```

7.5.15 Debug

7.5.16 Statistics

class `Tribler.Core.Modules.restapi.statistics_endpoint.`**`StatisticsCommunitiesEndpoint`**(*session*)
This class handles requests regarding Dispersy communities statistics.

`render_GET`(*request*)

`GET /statistics/communities`

A GET request to this endpoint returns general statistics of active Dispersy communities.

Example request:

```
curl -X GET http://localhost:8085/statistics/communities
```

Example response:

```
{
    "community_statistics": [{
        "identifier": "48d04e922dec4430daf22400c9d4cc5a3a53b27d",
        "member": "a66ebac9d88a239ef348a030d5ed3837868fc06d",
        "candidates": 43,
        "global_time": 42,
        "classification", "ChannelCommunity",
        "packets_sent": 43,
        "packets_received": 89,
        ...
    }, { ... }]
}
```

class `Tribler.Core.Modules.restapi.statistics_endpoint.`**`StatisticsDispersyEndpoint`**(*session*)
This class handles requests regarding Dispersy statistics.

`render_GET`(*request*)

`GET /statistics/dispersy`

A GET request to this endpoint returns general statistics in Dispersy. The returned runtime is the amount of seconds that Dispersy is active. The total uploaded and total downloaded statistics are in bytes.

Example request:

```
curl -X GET http://localhost:8085/statistics/dispersy
```

Example response:

```
{
    "dispersy_statistics": {
        "wan_address": "123.321.456.654:1234",
        "lan_address": "192.168.1.2:1435",
        "connection": "unknown",
        "runtime": 859.34,
        "total_downloaded": 538.53,
        "total_uploaded": 983.24,
        "packets_sent": 43,
        "packets_received": 89,
        ...
    }
}
```

class `Tribler.Core.Modules.restapi.statistics_endpoint.`**`StatisticsEndpoint`**(*session*)
This endpoint is responsible for handing requests regarding statistics in Tribler.

class `Tribler.Core.Modules.restapi.statistics_endpoint.`**`StatisticsTriblerEndpoint`**(*session*)
This class handles requests regarding Tribler statistics.

`render_GET`(*request*)

`GET /statistics/tribler`

A GET request to this endpoint returns general statistics in Tribler. The size of the Tribler database is in bytes.

Example request:

```
curl -X GET http://localhost:8085/statistics/tribler
```

Example response:

```
{
    "tribler_statistics": {
        "num_channels": 1234,
        "database_size": 384923,
        "torrent_queue_stats": [{
            "failed": 2,
            "total": 9,
            "type": "TFTP",
            "pending": 1,
            "success": 6
        }, ...]
    }
}
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

/channels

GET /channels/discovered, 28
GET /channels/discovered/(string:
channelid), 29
GET /channels/discovered/(string:
channelid)/playlists, 36
GET /channels/discovered/(string:
channelid)/rssfeeds, 33
GET /channels/discovered/(string:
channelid)/torrents, 34
GET /channels/subscribed, 30
POST /channels/discovered/(string:
channelid)/playlists/(int:
playlistid), 36
POST /channels/discovered/(string:
channelid)/recheckfeeds, 33
PUT /channels/discovered, 29
PUT /channels/discovered/(string:
channelid)/playlists, 37
PUT /channels/discovered/(string:
channelid)/rssfeeds/http%3A%2F%2Ftest.com%2Frss.xml,
32
PUT /channels/discovered/(string:
channelid)/torrents, 35
PUT /channels/discovered/(string:
channelid)/torrents/http%3A%2F%2Ftest.com%2Ftest.torrent,
34
PUT /channels/subscribed/(string:
channelid), 30
DELETE /channels/discovered/(string:
channelid)/playlists/(int:
playlistid), 36
DELETE /channels/discovered/(string:
channelid)/rssfeeds/http%3A%2F%2Ftest.com%2Frss.xml,
32
DELETE /channels/discovered/(string:
channelid)/torrents/(string:
torrent infohash), 33
DELETE /channels/subscribed/(string:
channelid), 30

/download

GET /download/(string:
infohash)/torrent, 38

/downloads

PUT /downloads, 39

/downloads?get_peers=(boolean: get_peers)&get_pieces=(boolean: get_pieces)

GET /downloads?get_peers=(boolean:
get_peers)&get_pieces=(boolean:
get_pieces), 38

/events

GET /events, 43

/mychannel

GET /mychannel, 31
POST /mychannel, 31

/search

GET /search/completions?q=(string:query),
40

/search?q=(string:query)

GET /search?q=(string:query), 40

/settings

GET /settings, 42
POST /settings, 42

/state

GET /state, 41

/statistics

GET /statistics/communities, 44
GET /statistics/dispersy, 44
GET /statistics/tribler, 45

/torrentinfo

`GET /torrentinfo`, 35

/variables

`GET /variables`, 41

t

Tribler.Core.Modules.restapi.channels.channels_discovered_endpoint, 28
Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint, 36
Tribler.Core.Modules.restapi.channels.channels_popular_endpoint, 31
Tribler.Core.Modules.restapi.channels.channels_rss_endpoint, 32
Tribler.Core.Modules.restapi.channels.channels_subscription_endpoint, 30
Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint, 33
Tribler.Core.Modules.restapi.channels.my_channel_endpoint, 31
Tribler.Core.Modules.restapi.downloads_endpoint, 38
Tribler.Core.Modules.restapi.events_endpoint, 43
Tribler.Core.Modules.restapi.search_endpoint, 39
Tribler.Core.Modules.restapi.settings_endpoint, 42
Tribler.Core.Modules.restapi.state_endpoint, 41
Tribler.Core.Modules.restapi.statistics_endpoint, 44
Tribler.Core.Modules.restapi.torrentinfo_endpoint, 35
Tribler.Core.Modules.restapi.variables_endpoint, 41

C

ChannelModifyRssFeedEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_rss_endpoint), 32

ChannelModifyTorrentEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint), 33

ChannelsDiscoveredEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_discovered_endpoint), 28

ChannelsDiscoveredSpecificEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_discovered_endpoint), 29

ChannelsModifyPlaylistsEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint), 36

ChannelsModifySubscriptionEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_subscription_endpoint), 30

ChannelsPlaylistsEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint), 36

ChannelsRecheckFeedsEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_rss_endpoint), 33

ChannelsRssFeedsEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_rss_endpoint), 33

ChannelsSubscribedEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_subscription_endpoint), 30

ChannelsTorrentsEndpoint (class in Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint), 34

D

DownloadExportTorrentEndpoint (class in Tribler.Core.Modules.restapi.downloads_endpoint), 38

DownloadsEndpoint (class in Tribler.Core.Modules.restapi.downloads_endpoint), 38

E

EventsEndpoint (class in Tribler.Core.Modules.restapi.events_endpoint), 43

M

MyChannelEndpoint (class in Tribler.Core.Modules.restapi.channels.my_channel_endpoint), 31

P

parse_setting() (Tribler.Core.Modules.restapi.settings_endpoint.SettingsEnd method), 42

parse_settings_dict() (Tribler.Core.Modules.restapi.settings_endpoint.SettingsEndpoint method), 42

R

render_DELETE() (Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint.C method), 36

render_DELETE() (Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.Chann method), 32

render_DELETE() (Tribler.Core.Modules.restapi.channels.channels_subscription_endpo method), 30

render_DELETE() (Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint.C method), 33

render_GET() (Tribler.Core.Modules.restapi.channels.channels_discovered_ method), 28

render_GET() (Tribler.Core.Modules.restapi.channels.channels_discovered_ method), 29

render_GET() (Tribler.Core.Modules.restapi.channels.channels_playlists_en method), 36

render_GET() (Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.ChannelsRssFeedsEndpoint method), 33
render_GET() (Tribler.Core.Modules.restapi.channels.channels_subscription_endpoint.ChannelsSubscribedEndpoint method), 30
render_GET() (Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint.ChannelsTorrentsEndpoint method), 34
render_GET() (Tribler.Core.Modules.restapi.channels.my_channel_endpoint.MyChannelEndpoint method), 31
render_GET() (Tribler.Core.Modules.restapi.downloads_endpoint.DownloadExportTorrentEndpoint method), 38
render_GET() (Tribler.Core.Modules.restapi.downloads_endpoint.DownloadsEndpoint method), 38
render_GET() (Tribler.Core.Modules.restapi.events_endpoint.EventsEndpoint method), 43
render_GET() (Tribler.Core.Modules.restapi.search_endpoint.SearchCompletionsEndpoint method), 39
render_GET() (Tribler.Core.Modules.restapi.search_endpoint.SearchEndpoint method), 40
render_GET() (Tribler.Core.Modules.restapi.settings_endpoint.SettingsEndpoint method), 42
render_GET() (Tribler.Core.Modules.restapi.state_endpoint.StateEndpoint method), 41
render_GET() (Tribler.Core.Modules.restapi.statistics_endpoint.StatisticsCommunitiesEndpoint method), 44
render_GET() (Tribler.Core.Modules.restapi.statistics_endpoint.StatisticsDispersyEndpoint method), 44
render_GET() (Tribler.Core.Modules.restapi.statistics_endpoint.StatisticsTriblerEndpoint method), 45
render_GET() (Tribler.Core.Modules.restapi.torrentinfo_endpoint.TorrentInfoEndpoint method), 35
render_GET() (Tribler.Core.Modules.restapi.variables_endpoint.VariablesEndpoint method), 41
render_POST() (Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint.ChannelsModifyPlaylistsEndpoint method), 36
render_POST() (Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.ChannelsRecheckFeedsEndpoint method), 33
render_POST() (Tribler.Core.Modules.restapi.channels.my_channel_endpoint.MyChannelEndpoint method), 31
render_POST() (Tribler.Core.Modules.restapi.settings_endpoint.SettingsEndpoint method), 42
render_PUT() (Tribler.Core.Modules.restapi.channels.channels_discovered_endpoint.ChannelsDiscoveredEndpoint method), 29
render_PUT() (Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint.ChannelsPlaylistsEndpoint method), 37
render_PUT() (Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.ChannelModifyRssFeedEndpoint method), 32
render_PUT() (Tribler.Core.Modules.restapi.channels.channels_subscription_endpoint.ChannelsModifySubscriptionEndpoint method), 30
render_PUT() (Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint.ChannelModifyTorrentEndpoint method), 34
render_PUT() (Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint.ChannelsTorrentsEndpoint method), 35
render_PUT() (Tribler.Core.Modules.restapi.downloads_endpoint.DownloadsEndpoint method), 39

S

SearchCompletionsEndpoint (class in Tribler.Core.Modules.restapi.search_endpoint), 39
SearchEndpoint (class in Tribler.Core.Modules.restapi.search_endpoint), 40
SettingsEndpoint (class in Tribler.Core.Modules.restapi.settings_endpoint), 42
StateEndpoint (class in Tribler.Core.Modules.restapi.state_endpoint), 41
StatisticsCommunitiesEndpoint (class in Tribler.Core.Modules.restapi.statistics_endpoint), 44
StatisticsDispersyEndpoint (class in Tribler.Core.Modules.restapi.statistics_endpoint), 44
StatisticsEndpoint (class in Tribler.Core.Modules.restapi.statistics_endpoint), 45
StatisticsTriblerEndpoint (class in Tribler.Core.Modules.restapi.statistics_endpoint), 45

T

TorrentInfoEndpoint (class in Tribler.Core.Modules.restapi.torrentinfo_endpoint), 35
Tribler.Core.Modules.restapi.channels.channels_discovered_endpoint (module), 28
Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint (module), 36
Tribler.Core.Modules.restapi.channels.channels_popular_endpoint (module), 31
Tribler.Core.Modules.restapi.channels.channels_rss_endpoint (module), 32
Tribler.Core.Modules.restapi.channels.channels_subscription_endpoint (module), 30
Tribler.Core.Modules.restapi.channels.channels_torrents_endpoint (module), 33
Tribler.Core.Modules.restapi.channels.my_channel_endpoint (module), 31
Tribler.Core.Modules.restapi.downloads_endpoint (module), 38
Tribler.Core.Modules.restapi.events_endpoint (module), 43
Tribler.Core.Modules.restapi.search_endpoint (module), 39
Tribler.Core.Modules.restapi.settings_endpoint (module), 42
Tribler.Core.Modules.restapi.state_endpoint (module), 41

Tribler.Core.Modules.restapi.statistics_endpoint (module), 44
Tribler.Core.Modules.restapi.torrentinfo_endpoint (module), 35
Tribler.Core.Modules.restapi.variables_endpoint (module), 41

V

VariablesEndpoint (class in Tribler.Core.Modules.restapi.variables_endpoint), 41