# Tribler Documentation

***Release 7.0.2***

**Tribler devs**

**Dec 11, 2018**

# Contents

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 11 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, macOS 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 macOS.

## 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 run_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 committing 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 committing 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 `release-X.Y.Z` 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:**

- New features always go to `devel`.
- If there is an unreleased `release-X.Y.Z` branch, fixes go there.
- Otherwise, fixes 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 submitting 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 methodology 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 follows the GitFlow model described at length in Vincent Driessen's post.

Our main repository contains 2 permanent branches:

- **devel**: The main development branch; all new features and fixes for them belong here. Every time a new release cycle is started, a new **release-X.Y.Z(-abc)** branch is forked from it.
- **master**: Contains the code of the latest stable release. It gets updated from **release-** after every release.

## 3.2 Release lifecycle

A release is started by forking from the top of **devel**. The first commit added to the release branch bumps the release version. From that moment, all the bugfixes relevant to the current release must be merged only into the corresponding release branch. No new features could be added to it.

When the release is ready, it is merged into **master** with the suitable release tag. Next it is merged into the current **devel** branch to integrate the bugfixes. If a bugfix from the current release branch is really, really necessary for the current **devel** branch, it could be cherry-picked onto **devel**. But be aware that each of these cherry-pickings eventually results in a merge conflict that should be resolved manually.

The release branch lives in the Tribler repository for as long as we support the corresponding Tribler version. Eventually, the branch is removed.

## 3.3 Tags

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

## 3.4 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!

## 3.5 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/release-X.Y.Z -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 **release-** branches. Those are useless as they will quickly get out of date, so kill them with fire:

```
git branch -d release-X.Y.Z
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
```

## 3.6 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.

## 3.7 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 structures. 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 or .exe installer. The guide targets Windows 7 or higher, 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 need to add the `C:\\Python27\\` and `C:\\Python27\\Scripts` directories to your PATH variable. Information about how to set path variable can be found here.

In order to compile some of the dependencies of Tribler, you will need Visual Studio 2015 which can be downloaded from here or 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 packages, there should be no more import errors. It may be required to enable Visual C++ Toolset on the Command Line if Native Command Line tool is not available. You can do that by following article here.

### 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 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 PyQt5

If you wish to run the Tribler Graphical User Interface, PyQt5 should be available on the system. While PyQt5 is available in the pip repository, this is only compatible with Python 3. There is an unofficial distribution available for Python 2.7 here https://github.com/pyqt/python-qt5. You can simply install PyQt5 from this repository.

```
pip install git+git://github.com/pyqt/python-qt5.git
```

After installation, check it was correctly installed

```
python -c "import PyQt5" # this should work without any error
```

**Alternatively,** if above steps do not work, follow the instructions below.

Start by downloading the Qt library from here. You can either compile it from source or use a Qt installer which automatically installs the pre-compiled libraries. Make sure to choose the correct distribution based on your platform(32/64 bit).

After the Qt installation is completed, PyQt5 should be compiled. This library depends on SIP, another library to automatically generate Python bindings from C++ code. Download the latest SIP version here, extract it, navigate to the directory where it has been extracted and compile/install it (don't forget to execute these commands in the Visual Studio command line):

```
python configure.py
nmake
nmake install
```

Next, download PyQt5 from here and make sure that you download the version that matches with the version of Qt you installed in the previous steps. Extract the binary and compile it:

```
python configure.py --qmake=<qmake_path> --disable=QtNfc --disable=QtBluetooth
nmake
nmake install
python -c "import PyQt5" # this should work without any error
```

Note that <qmake_path> is the path to the qmake.exe file path. For eg. qmake could be here C:\Qt\Qt5.6.2\5.6\msvc2015_64\bin\qmake.exe but depends on your installation. Here, we are disabling QtNfc and QtBluetooth modules which contains classes that provide connectivity between NFC & Bluetooth enabled devices respectively which we do not require in Tribler. Moreover, not disabling these modules may lead to missing DLL files causing installation to fail. So, we can safely disable them. The installation can take a while. After it has finished, the PyQt5 library is installed correctly.

### 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.

### 4.1.5 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"
```

### 4.1.6 libtorrent

To install libtorrent, you can simply copy the libtorrent.pyd file from the Github repository here and place it inside your python site-packages directory.

**Alternatively,** if above does not work then you can try to compile 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\\` 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`)

After successfully copying the `libtorrent.pyd` file either compiled or from the repository, you can check if the installation was successful:

```
python -c "import libtorrent" # this should work without any error
```

### 4.1.7 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')"
```

### 4.1.8 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.

To install LevelDB, you can simply copy the `leveldb.pyd` file from the Github repository here and place it inside your python site-packages directory. Then, check check if installation was successful:

```
python -c "import leveldb" # this should work without any error
```

**Alternatively,** 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"
```

### 4.1.9 VLC

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

### 4.1.10 NumPy & SciPy

To install NumPy & SciPy, download the respective .whl files here and install using with pip as below. Make sure to download files with cp27 in names as they are for python 2.7

```
pip install scipy0.19.1cp27cp27mwin_amd64.whl
pip install numpy1.13.1+mklcp27cp27mwin_amd64.whl
```

### 4.1.11 Additional Packages

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

```
pip install cython  # Needs to be installed first for meliae
pip install bitcoinlib cherrypy chardet configobj cryptography decorator feedparser␣
↪meliae netifaces networkx pillow psutil twisted libnacl
```

### 4.1.12 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 run_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 MacOS

Tribler development environment setup on MacOS (10.10 to 10.13).

1. MacPorts
2. HomeBrew
3. Tribler
4. Notes

### 4.2.1 MacPorts

MacPorts Install instructions at macports.org. To install the Tribler dependencies using MacPorts, please run the following command in your terminal:

```
sudo port -N install git ffmpeg qt5-qtcreator libtorrent-rasterbar gmp mpfr libmpc␣
↪libsodium py27-m2crypto py27-apsw py27-Pillow py27-twisted py27-cherrypy3 py27-cffi␣
↪py27-chardet py27-configobj py27-gmpy2 py27-pycparser py27-numpy py27-idna py27-
↪leveldb py27-cryptography py27-decorator py27-feedparser py27-netifaces py27-
↪service_identity py27-asn1-modules py27-pyinstaller py27-pyqt5 py27-sqlite py27-
↪matplotlib py27-libnacl
```

### 4.2.2 HomeBrew

#### Note

Skip to Tribler if you are using MacPorts because HomeBrew is a less complete alternative to MacPorts.

HomeBrew installation instructions can be found at brew.sh.

#### PyQt5

If you wish to run the Tribler Graphical User Interface, PyQt5 should be available on the system. While PyQt5 is available in the pip repository, this is only compatible with Python 3. To install PyQt5, we first need to install Qt5, a C++ library which can be installed with brew:

```
brew install qt5
brew cask install qt-creator # if you want the visual designer
qmake --version # test whether qt is installed correctly
```

After the installation completed, PyQt5 should be compiled. This library depends on SIP, another library to automatically generate Python bindings from C++ code. Download the latest SIP version here, extract it, navigate to the directory where it has been extracted and compile/install it:

```
python configure.py
make
sudo make install
```

Next, download PyQt5 from here and make sure that you download the version that matches with the version of Qt you installed in the previous steps. Extract the binary and compile it:

```
python configure.py
make
sudo make install
python -c "import PyQt5" # this should work without any error
```

Note that the installation can take a while. After it has finished, the PyQt5 library is installed correctly.

#### 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"
```

### 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
```

### 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/2.7/site-packages/homebrew.pth
```

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
```

#### 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
sudo easy_install pip
pip install --user cython  # Needs to be installed first for meliae
pip install --user cherrypy cffi chardet configobj cryptography decorator feedparser␣
↪gmpy2 idna leveldb meliae netifaces numpy pillow psutil pyasn1 pycparser scipy␣
↪twisted service_identity libnacl bitcoinlib
```

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
```

### 4.2.3 Tribler

```
git clone --recursive  https://github.com/Tribler/tribler.git
cd tribler
cp /usr/local/lib/libsodium.dylib ./ || cp /opt/local/lib/libsodium.dylib ./
mkdir vlc
which ffmpeg | xargs -I {} cp "{}" vlc/
```

Proceed proceed to Build instructions

### 4.2.4 Notes

#### System Integrity Protection

The security system on MacOS 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.

The best solution to this problem is to link or copy `libsodium.dylib` into the Tribler root directory.

#### Help

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 Linux

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

### 4.3.1 Debian/Ubuntu/Mint

First, install the required dependencies by executing the following command in your terminal:

```
sudo apt-get install libav-tools libsodium18 libx11-6 python-apsw python-cherrypy3␣
↪python-cryptography python-decorator python-feedparser python-leveldb python-
↪libtorrent python-matplotlib python-meliae python-m2crypto python-netifaces python-
↪pil python-psutil python-pyasn1 python-scipy python-twisted python2.7 vlc python-
↪chardet python-configobj python-pyqt5 python-pyqt5.qtsvg python-libnacl
```

Next, download the latest .deb file from here.

### 4.3.2 Installing the python-socks on Ubuntu >= 16.10

The python-pysocks package was renamed to python-socks in the Ubuntu repositories from version 16.10 (Yakkety Yak) onwards. Use the following command to resolve this dependency:

```
sudo apt-get install python-socks python-networkx python-libnacl
```

### 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 respective 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:

```
pacman -S libsodium libtorrent-rasterbar python2-pyqt5 qt5-svg phonon-qt5-vlc python2-
↪apsw python2-cherrypy python2-cryptography python2-decorator python2-feedparser␣
↪python2-chardet python2-m2crypto python2-netifaces python2-plyvel python2-twisted␣
↪python2-configobj python2-matplotlib python2-networkx python2-psutil python2-scipy␣
↪python2-libnacl
```

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-packages/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. * PyInstaller, a tool to create an executable from python files. Install the latest version from pip: `pip install pyinstaller`. * 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 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.
2. A folder `vlc` that contains `libvlc.dll`, `libvlccore.dll` and a directory `plugins` that contain the VLC plugins.
3. `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.
4. `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.
5. `libsodium.dll` which can be downloaded from libsodium.org (as of writing version 1.0.8).
6. The openssl dll files `libeay32.dll`, `libssl32.dll` and `ssleay32.dll` (place them in a directory named `openssl`).

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.

```
cd tribler
python Tribler/Main/Build/update_version_from_git.py 64
win\makedist_win.bat 64
```

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

## 5.2 MacOS

This guide explains how to build Tribler on MacOS (10.10 to 10.13). The final result is a `.dmg` file which, when opened, allows `Tribler.app` to be copied to the Applications directory and or launched. Make sure the required packages required by Tribler are installed from the Development instructions.

### 5.2.1 Required packages

- vlc: PyInstaller automatically searches for the vlc library in the system and bundles it.
- eulagise: 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 downloaded 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 macOS

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).

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
```

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
```

This will build the `.deb` file that can be used to install Tribler on other systems.

## 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

# Tribler REST API

## 6.1 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**.

## 6.2 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
```

## 6.3 Error handling

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

```
{
    "error": {
        "handled": False,
        "code": "SomeException",
```

(continues on next page)

(continued from previous page)

```
        "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"
    }
}
```

## 6.4 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 |

## 6.5 Endpoints

### 6.5.1 Discovered channels

### 6.5.2 Subscribed channels

### 6.5.3 Popular channels

**class** Tribler.Core.Modules.restapi.channels.channels_popular_endpoint.**ChannelsPopularEndpo**

**render_GET**(*request*)

**GET /channels/popular?limit=(int:max nr of channels)**

A GET request to this endpoint will return the most popular discovered channels in Tribler. You can optionally pass a limit parameter to limit the number of results.

**Example request**:

```
curl -X GET http://localhost:8085/channels/popular?limit=1
```

**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,
    }]
}
```

### 6.5.4 My channel

### 6.5.5 RSS Feeds

**class** Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.**BaseChannelsRssFeedsEndpo**

**get_my_channel_obj_or_error**(*request*)
Returns a tuple of (channel_obj, error). Callers of this method should check whether the channel_obj is None and if so, return the error.

**class** Tribler.Core.Modules.restapi.channels.channels_rss_endpoint.**ChannelModifyRssFeedEndpo**

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.com%2Frss.xml
```

**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%2Frss.xml
```

**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.**ChannelsRecheckFeedsEndpo**

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**

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",
    }, ...]
}
```

### 6.5.6 Torrents

### 6.5.7 Torrent info

### 6.5.8 Playlists

**class** Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint.**ChannelsModifyPlayl**

**render_DELETE**(*request*)

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

Remove a torrent with a specified infohash from a specified playlist.

**Example request**:

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

**Example response**:

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

**statuscode 404** if the specified channel/playlist/torrent does not exist.

**render_PUT**(*request*)

**PUT /channels/discovered/(string: channelid)/playlists/(int: playlistid)/(string: i**

Add a torrent with a specified infohash to a specified playlist. The torrent that is added to the playlist, should be present in the channel.

**Example request**:

```
curl -X PUT http://localhost:8085/channels/discovered/abcd/playlists/3/
↪abcdef
```

**Example response**:

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

**statuscode 404** if the specified channel/playlist/torrent does not exist.

**statuscode 409** if the specified torrent is already in the specified playlist.

**class** Tribler.Core.Modules.restapi.channels.channels_playlists_endpoint.**ChannelsModifyPlayl**

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.**ChannelsPlaylistsEn**

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. - disable_filter: whether the family filter should be disabled for this request (1 = disabled)

**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

### 6.5.9 Downloads

### 6.5.10 Search

### 6.5.11 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 - EXCEPTION: An exception has occurred in the core

**Example request**:

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

**Example response**:

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

### 6.5.12 Settings

### 6.5.13 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_removed_from_channel: An indicator that a torrent has been removed from a channel. The event contains the infohash and the dispersy id of the channel which contained the removed 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.
- market_ask: Tribler learned about a new ask in the market. The event includes information about the ask.
- market_bid: Tribler learned about a new bid in the market. The event includes information about the bid.
- market_ask_timeout: An ask has expired. The event includes information about the ask.
- market_bid_timeout: An bid has expired. The event includes information about the bid.
- market_transaction_complete: A transaction has been completed in the market. The event contains the transaction that was completed.
- market_payment_received: We received a payment in the market. The events contains the payment information.
- market_payment_sent: We sent a payment in the market. The events contains the payment information.
- market_iom_input_required: The Internet-of-Money modules requires user input (like a password or challenge response).

**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
```

### 6.5.14 Wallets

**class** Tribler.Core.Modules.restapi.wallets_endpoint.**WalletBalanceEndpoint**(*session*, *identifier*)

This class handles requests regarding the balance in a wallet.

**render_GET**(*request*)

**GET /wallets/(string:wallet identifier)/balance**

A GET request to this endpoint will return balance information of a specific wallet.

**Example request**:

```
curl -X GET http://localhost:8085/wallets/BTC/balance
```

**Example response**:

```
{
    "balance": {
        "available": 0.000126,
        "pending": 0.0,
        "currency": "BTC"
    }
}
```

**class** Tribler.Core.Modules.restapi.wallets_endpoint.**WalletEndpoint**(*session*, *identifier*)

This class represents the endpoint for a single wallet.

**render_PUT**(*request*)

**PUT /wallets/(string:wallet identifier)**

A request to this endpoint will create a new wallet.

**Example request**:

```
curl -X PUT http://localhost:8085/wallets/BTC
```

**Example response**:

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

**class** Tribler.Core.Modules.restapi.wallets_endpoint.**WalletTransactionsEndpoint**(*session*, *identifier*)

This class handles requests regarding the transactions of a wallet.

**render_GET**(*request*)

**GET /wallets/(string:wallet identifier)/transactions**

A GET request to this endpoint will return past transactions of a specific wallet.

**Example request**:

```
curl -X GET http://localhost:8085/wallets/BTC/transactions
```

**Example response**:

```
{
    "transactions": [{
        "currency": "BTC",
        "to": "17AVS7n3zgBjPq1JT4uVmEXdcX3vgB2wAh",
        "outgoing": false,
        "from": "",
        "description": "",
        "timestamp": "1489673696",
        "fee_amount": 0.0,
        "amount": 0.00395598,
        "id":
↪"6f6c40d034d69c5113ad8cb3710c172955f84787b9313ede1c39cac85eeaaffe"
    }, ...]
}
```

**class** Tribler.Core.Modules.restapi.wallets_endpoint.**WalletTransferEndpoint**(*session*, *identifier*)

This class handles requests regarding transferring money by a wallet.

**render_POST**(*request*)

**POST /wallets/(string:wallet identifier)/transfer**

A POST request to this endpoint will transfer some units from a wallet to another address.

**Example request**:

```
curl -X POST http://localhost:8085/wallets/BTC/transfer
--data "amount=0.3&destination=mpC1DDgSP4PKc5HxJzQ5w9q6CGLBEQuLsN"
```

**Example response**:

```
{
    "txid": "abcd"
}
```

**class** Tribler.Core.Modules.restapi.wallets_endpoint.**WalletsEndpoint**(*session*)

This class represents the root endpoint of the wallets resource.

**render_GET**(*request*)

**GET /wallets**

A GET request to this endpoint will return information about all available wallets in Tribler. This includes information about the address, a human-readable wallet name and the balance.

**Example request**:

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

**Example response**:

```
{
    "wallets": [{
        "created": True,
        "name": "Bitcoin",
        "unlocked": True,
        "precision": 8,
        "min_unit": 100000,
        "address": "17AVS7n3zgBjPq1JT4uVmEXdcX3vgB2wAh",
        "balance": {
            "available": 0.000126,
            "pending": 0.0,
            "currency": "BTC"
        }
    }, ...]
}
```


### 6.5.15 Market

**class** Tribler.Core.Modules.restapi.market.asks_bids_endpoint.**AsksEndpoint**(*session*)

This class handles requests regarding asks in the market community.

**render_GET**(*request*)

#### GET /market/asks

A GET request to this endpoint will return all ask ticks in the order book of the market community.

**Example request**:

```
curl -X GET http://localhost:8085/market/asks
```

**Example response**:


```
{
    "asks": [{
        "asset1": "BTC",
        "asset2": "MB",
        "ticks": [{
            "trader_id": "12c406358ba05e5883a75da3f009477e4ca699a9",
            "timeout": 3600,
            "assets": {
                "first": {
                    "amount": 10,
                    "type": "BTC"
                },
                "second": {
                    "amount": 10,
                    "type": "MB"
                }
            },
            "traded": 5,
            "timestamp": 1493905920.68573,
            "order_number": 1}, ...]
    }, ...]
}
```

**render_PUT**(*request*)

#### PUT /market/asks

A request to this endpoint will create a new ask order.

**Example request**:

```
curl -X PUT http://localhost:8085/market/asks --data
"first_asset_amount=10&second_asset_amount=10&first_asset_type=BTC&
↪second_asset_type=MB"
```

**Example response**:

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

**class** Tribler.Core.Modules.restapi.market.asks_bids_endpoint.**BaseAsksBidsEndpoint**(*session*)

This class acts as the base class for the asks/bids endpoint.

**static create_ask_bid_from_params**(*parameters*)

Create an ask/bid from the provided parameters in a request. This method returns a tuple with the price, quantity and timeout of the ask/bid.

**class** Tribler.Core.Modules.restapi.market.asks_bids_endpoint.**BidsEndpoint**(*session*)

This class handles requests regarding bids in the market community.

**render_GET**(*request*)

#### GET /market/bids

A GET request to this endpoint will return all bid ticks in the order book of the market community.

**Example request**:

```
curl -X GET http://localhost:8085/market/bids
```

**Example response**:

```
{
    "bids": [{
        "asset1": "BTC",
        "asset2": "MB",
        "ticks": [{
            "trader_id": "12c406358ba05e5883a75da3f009477e4ca699a9",
            "timeout": 3600,
            "assets": {
                "first": {
                    "amount": 10,
                    "type": "BTC"
                },
                "second": {
                    "amount": 10,
                    "type": "MB"
                }
            },
            "traded": 5,
            "timestamp": 1493905920.68573,
```

(continues on next page)

(continued from previous page)

```
            "order_number": 1}, ...]
    }, ...]
}
```

**render_PUT**(*request*)

**PUT /market/bids**

A request to this endpoint will create a new bid order.

**Example request**:

```
curl -X PUT http://localhost:8085/market/bids --data
"first_asset_amount=10&second_asset_amount=10&first_asset_type=BTC&
↪second_asset_type=MB"
```

**Example response**:

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

**class** Tribler.Core.Modules.restapi.market.orders_endpoint.**OrderCancelEndpoint**(*session*, *order_number*)

This class handles requests for cancelling a specific order.

**render_POST**(*request*)

**GET /market/orders/**(**string:** *order_number*)**/cancel**

A POST request to this endpoint will cancel a specific order.

**Example request**:

```
curl -X GET http://localhost:8085/market/orders/3/cancel
```

**Example response**:

```
{
    "cancelled": True
}
```

**class** Tribler.Core.Modules.restapi.market.orders_endpoint.**OrderSpecificEndpoint**(*session*, *order_number*)

**class** Tribler.Core.Modules.restapi.market.orders_endpoint.**OrdersEndpoint**(*session*)

This class handles requests regarding your orders in the market community.

**render_GET**(*request*)

**GET /market/orders**

A GET request to this endpoint will return all your orders in the market community.

**Example request**:

```
curl -X GET http://localhost:8085/market/orders
```

**Example response**:


```
{
    "orders": [{
        "trader_id": "12c406358ba05e5883a75da3f009477e4ca699a9",
        "timestamp": 1493906434.627721,
        "assets" {
            "first": {
                "amount": 3,
                "type": "BTC",
            },
            "second": {
                "amount": 3,
                "type": "MB",
            }
        }
        "reserved_quantity": 0,
        "is_ask": False,
        "timeout": 3600,
        "traded": 0,
        "order_number": 1,
        "completed_timestamp": null,
        "cancelled": False,
        "status": "open"
    }]
}
```


**class** Tribler.Core.Modules.restapi.market.transactions_endpoint.**TransactionPaymentsEndpoint**

This class handles requests for the payments of a specific transaction.

**render_GET**(*request*)

**GET /market/transactions/**(**string:** *trader_id*)**/**
**string:** *transaction_number***/payments**

A GET request to this endpoint will return all payments tied to a specific transaction.

**Example request**:

```
curl -X GET http://localhost:8085/market/transactions/
12c406358ba05e5883a75da3f009477e4ca699a9/3/payments
```

**Example response**:


```
{
    "payments": [{
        "trader_id": "12c406358ba05e5883a75da3f009477e4ca699a9",
        "transaction_number": 3,
        "price": 10,
```


(continues on next page)

(continued from previous page)

```
        "price_type": "MC",
        "quantity": 10,
        "quantity_type": "BTC",
        "transferred_quantity": 4,
        "payment_id": "abcd",
        "address_from": "my_mc_address",
        "address_to": "my_btc_address",
        "timestamp": 1493906434.627721,
    ]
}
```

**class** Tribler.Core.Modules.restapi.market.transactions_endpoint.**TransactionSpecificNumberEn**

This class handles requests for a transaction with a specific number.

**class** Tribler.Core.Modules.restapi.market.transactions_endpoint.**TransactionSpecificTraderEn**

This class handles requests for a specific transaction.

**class** Tribler.Core.Modules.restapi.market.transactions_endpoint.**TransactionsEndpoint**(*session*)

This class handles requests regarding (past) transactions in the market community.

**render_GET**(*request*)

**GET /market/transactions**

A GET request to this endpoint will return all performed transactions in the market community.

**Example request**:

```
curl -X GET http://localhost:8085/market/transactions
```

**Example response**:

```
{
    "transactions": [{
        "trader_id": "12c406358ba05e5883a75da3f009477e4ca699a9",
        "order_number": 4,
        "partner_trader_id": "34c406358ba05e5883a75da3f009477e4ca699a9
↪",
        "partner_order_number": 1,
        "transaction_number": 3,
        "assets" {
            "first": {
                "amount": 3,
                "type": "BTC",
            },
            "second": {
                "amount": 3,
                "type": "MB",
            }
        },
        "transferred" {
            "first": {
```

(continues on next page)

(continued from previous page)


```
                "amount": 3,
                "type": "BTC",
            },
            "second": {
                "amount": 3,
                "type": "MB",
            }
        }
        "timestamp": 1493906434.627721,
        "payment_complete": False
    ]
}
```


### 6.5.16 Debug

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugCPUEndpoint**(*session*)
This class handles request for information about CPU.

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugCPUHistoryEndpoint**(*session*)
This class handles request for information about CPU usage history.

**render_GET**(*request*)

**GET /debug/cpu/history**

A GET request to this endpoint returns information about CPU usage history in the form of a list.

**Example request**:

```
curl -X GET http://localhost:8085/debug/cpu/history
```

**Example response**:


```
{
    "cpu_history": [{
        "time": 1504015291214,
        "cpu": 3.4,
    }, ...]
}
```


**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugCircuitSlotsEndpoint**(*session*)
This class handles requests for information about slots in the tunnel overlay.

**render_GET**(*request*)

**GET /debug/circuits/slots**

A GET request to this endpoint returns information about the slots in the tunnel overlay.

**Example request**:

```
curl -X GET http://localhost:8085/debug/circuits/slots
```

**Example response**:

```
{
    "open_files": [{
        "path": "path/to/open/file.txt",
        "fd": 33,
    }, ...]
}
```

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugCircuitsEndpoint**(*session*)
This class handles requests regarding the tunnel community debug information.

**render_GET**(*request*)

#### GET /debug/circuits

A GET request to this endpoint returns information about the built circuits in the tunnel community.

**Example request**:

```
curl -X GET http://localhost:8085/debug/circuits
```

**Example response**:

```
{
    "circuits": [{
        "id": 1234,
        "state": "EXTENDING",
        "goal_hops": 4,
        "bytes_up": 45,
        "bytes_down": 49,
        "created": 1468176257,
        "hops": [{
            "host": "unknown"
        }, {
            "host": "39.95.147.20:8965"
        }],
        ...
    }, ...]
}
```

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugEndpoint**(*session*)
This endpoint is responsible for handing requests regarding debug information in Tribler.

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugLogEndpoint**(*session*)
This class handles the request for displaying the logs.

**render_GET**(*request*)

#### GET /debug/log?process=<core|gui>&max_lines=<max_lines>

A GET request to this endpoint returns a json with content of core or gui log file & max_lines requested

**Example request**:

```
curl -X GET http://localhost:8085/debug/log?process=core&max_lines=5
```

**Example response**:

A JSON with content of the log file & max_lines requested, for eg. {

"max_lines" : 5, "content" :"INFO 1506675301.76 sqlitecachedb:181 Reading database version...

INFO 1506675301.76 sqlitecachedb:185 Current database version is 29 INFO 1506675301.76 sqlitecachedb:203 Beginning the first transaction... INFO 1506675301.76 upgrade:93 tribler is in the latest version,... INFO 1506675302.08 LaunchManyCore:254 lmc: Starting Dispersy..."

}

**tail**(*file_handler*, *lines=1*)
Tail a file and get X lines from the end

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugMemoryDumpEndpoint**(*session*)
This class handles request for dumping memory contents.

**render_GET**(*request*)

**GET /debug/memory/dump**

A GET request to this endpoint returns a Meliae-compatible dump of the memory contents.

**Example request**:

```
curl -X GET http://localhost:8085/debug/memory/dump
```

**Example response**:

The content of the memory dump file.

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugMemoryEndpoint**(*session*)
This class handles request for information about memory.

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugMemoryHistoryEndpoint**(*session*)
This class handles request for information about memory usage history.

**render_GET**(*request*)

**GET /debug/memory/history**

A GET request to this endpoint returns information about memory usage history in the form of a list.

**Example request**:

```
curl -X GET http://localhost:8085/debug/memory/history
```

**Example response**:

```
{
    "memory_history": [{
        "time": 1504015291214,
        "mem": 324324,
    }, ...]
}
```

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugOpenFilesEndpoint**(*session*)
This class handles request for information about open files.

**render_GET**(*request*)

**GET /debug/open_files**

A GET request to this endpoint returns information about files opened by Tribler.

**Example request**:

```
curl -X GET http://localhost:8085/debug/open_files
```

**Example response**:

```
{
    "open_files": [{
        "path": "path/to/open/file.txt",
        "fd": 33,
    }, ...]
}
```

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugOpenSocketsEndpoint**(*session*)
This class handles request for information about open sockets.

**render_GET**(*request*)

#### GET /debug/open_sockets

A GET request to this endpoint returns information about open sockets.

**Example request**:

```
curl -X GET http://localhost:8085/debug/openfiles
```

**Example response**:

```
{
    "open_sockets": [{
        "family": 2,
        "status": "ESTABLISHED",
        "laddr": "0.0.0.0:0",
        "raddr": "0.0.0.0:0",
        "type": 30
    }, ...]
}
```

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugProfilerEndpoint**(*session*)
This class handles requests for the profiler.

**render_DELETE**(*request*)

#### DELETE /debug/profiler

A PUT request to this endpoint stops the profiler.

**Example request**:

```
curl -X DELETE http://localhost:8085/debug/profiler
```

**Example response**:

```
{
    "success": "true"
}
```

**render_GET**(*request*)

#### GET /debug/profiler

A GET request to this endpoint returns information about the state of the profiler. This state is either STARTED or STOPPED.

**Example request**:

```
curl -X GET http://localhost:8085/debug/profiler
```

**Example response**:

```
{
    "state": "STARTED"
}
```

**render_PUT**(*request*)

#### PUT /debug/profiler

A PUT request to this endpoint starts the profiler.

**Example request**:

```
curl -X PUT http://localhost:8085/debug/profiler
```

**Example response**:

```
{
    "success": "true"
}
```

**class** Tribler.Core.Modules.restapi.debug_endpoint.**DebugThreadsEndpoint**(*session*)

This class handles request for information about threads.

**render_GET**(*request*)

#### GET /debug/threads

A GET request to this endpoint returns information about running threads.

**Example request**:

```
curl -X GET http://localhost:8085/debug/threads
```

**Example response**:

```
{
    "threads": [{
        "thread_id": 123456,
        "thread_name": "my_thread",
        "frames": ["my_frame", ...]
    }, ...]
}
```

**class** Tribler.Core.Modules.restapi.debug_endpoint.**MemoryDumpBuffer**(*buf=''*)

Meliae expects its file handle to support write(), flush() and __call__(). The StringIO class does not support __call__(), therefore we provide this subclass.

### 6.5.17 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**:

```
{
    "dispersy_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.**StatisticsIPv8Endpoint**(*session*)
This class handles requests regarding IPv8 statistics.

**render_GET**(*request*)

**GET /statistics/ipv8**

A GET request to this endpoint returns general statistics of IPv8.

**Example request**:

```
curl -X GET http://localhost:8085/statistics/ipv8
```

**Example response**:

```
{
    "ipv8_statistics": {
        "total_up": 3424324,
        "total_down": 859484
    }
}
```

**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 7

# TrustChain

TrustChain is a tamper-resistant data structure that is used in Tribler to record community contributions. This blockchain-based distributed ledger can then be used to build a reputation mechanism that is able to identify free-riders. A basic implementation of TrustChain is available in our code base and available for other developers.

TrustChain is specifically designed to be transaction-agnostic which means that any transaction can be stored in TrustChain. In Tribler, this consists of the amount of uploaded and downloaded data.

## 7.1 Using TrustChain

Using TrustChain to store transaction is straightforward. Creating a new block is done by invoking the `sign_block` method of the community. The required arguments are the destination candidate (the counterparty of the transaction), your public key and the transaction itself, in the Python dictionary format. Note that this dictionary can only contain Python primitive types and no custom objects due to the serialization of the transaction when sending it to the other party.

Assuming that the transaction counterparty is online and the block is valid, the counterparty signs the block and sends it back where the `received_half_block` method is invoked, processing the received block.

## 7.2 Using TrustChain in your project

To use TrustChain in your own projects, one can create a subclass of `TrustChainCommunity` or use the `TrustChainCommunity` directly. This should be enough for basic usage. For more information about communities, we reference the reader to a Dispersy tutorial.

In order to implement custom transaction validation rules, a subclass of `TrustChainBlock` should be made and the `BLOCK_CLASS` variable in the `TrustChainCommunity` should be updated accordingly. By overriding the `validate_transaction` method, you can add your own custom validation rules.

# CHAPTER 8

# Twisted in Tribler

Once you have a basic understanding of Python, you will need to understand two more advanced concepts before you can start working on Tribler: generator functions and scheduling in Twisted. This document will teach you about both in the context of Tribler.

## 8.1 Generator functions

When browsing the Tribler source code you will find a lot of `yield` statements. This makes this function a *generator* and as a result you will not find any `return` statements in this function (which would be syntactically invalid). The special thing about these *generators* is that they can return intermittent values, without releasing their local context. This is an advantage when the caller of this *generator* does not necessarily need all of the outputs the *generator* could produce. Instead, the caller can decide to stop iterating over the outputs of the *generator* at any given time.

Take for example an identifier generator:

```
def get_id():
    i = 0
    while True:
        yield i
        i = i + 1
```

One could then call this `get_id()` function as follows:

```
print get_id().next()
```

## 8.2 Yielding Deferreds

Now that you know about generators we can discuss Twisted's `Deferred` objects. Essentially, the only thing a `Deferred` does, is call a specified callback function if `Deferred.callback()` has been called. For more information you can reference the official documentation at https://twistedmatrix.com/documents/16.5.0/core/howto/defer.html.

To show how these Deferreds can be used in conjunction with generators, we will give an example. In Tribler you will find a lot of the following types of code:

```
@inlineCallbacks
def some_function():
    # Wait for some_deferred_object to be called
    yield some_deferred_object
    # The some_deferred_object event has happened now
```

This block of code does two things.

1. [`yield some_deferred_object`] Yield a `Deferred` object
2. [`@inlineCallbacks`] Call all values of the generator, call the `next()` value of this generator every time the previously yielded Deferred has been fired.

Practically speaking, this pauses the control flow through this function until the `some_deferred_object` has been called. This is useful when dealing with asynchronous events, which otherwise might not be guaranteed to have happened at a certain point in time. What makes the `Deferred` structure even more useful, is that it will return with a value once it has been called. This means that we can use the return value of the yield within our generator function. In our example:

```
@inlineCallbacks
def some_function():
    # Wait for some_deferred_object to produce a value
    value = yield some_deferred_object
    # Now we can continue our control flow
    print value
```

## 8.3 Caveats

**Return values -** As previously mentioned, Python generators do not allow return values. To do this in a Deferred generator, one can use the `returnValue()` function. In our example:

```
@inlineCallbacks
def some_function():
    value = yield some_deferred_object
    returnValue(value)
```

**The main thread -** Python has some issues when it comes to function reentrancy. To this end you might have to tell the Twisted framework that it cannot schedule other functions in between yields of your generator. Like so:

```
@blocking_call_on_reactor_thread
@inlineCallbacks
def some_function():
    yield some_deferred_object
    # No other function can be called on this thread while the yield is waiting
```

Do note that your `some_deferred_object` cannot be called from the main thread now! Some other thread will have to wake the Deferred for the function to continue execution.

## 8.4 Further reading

Functional programming in Python: https://docs.python.org/2.7/howto/functional.html

Deferred reference: https://twistedmatrix.com/documents/16.5.0/core/howto/defer.html
Threading in Twisted: http://twistedmatrix.com/documents/current/core/howto/threading.html

CHAPTER 9

# Indices and tables

- genindex
- modindex
- search

# HTTP Routing Table

## /channels



## /debug



## /events



## /market



## /state



## /statistics



## /wallets

# Python Module Index

## t

# Index