pyrocore Documentation

Release 0.6.1

pyroscope

Jun 12, 2017

Contents

pyrocore is a collection of tools for the BitTorrent protocol and especially the rTorrent client. They enable you to filter rTorrent's item list for displaying or changing selected items, also creating, inspecting and changing `.torrent` files, and much more.

An optional daemon process named `pyrotorque` can add flexible queue management for rTorrent, starting items added in bulk slowly over time according to customizable rules.

It can also watch a directory tree recursively for new metafiles using *inotify*. That means `.torrent` files you drop anywhere into that watched tree are loaded instantaneously, without any polling and no extra configuration for nested directories.

Note: The *PyroScope* command line utilities (i.e. *pyrocore*) are *not* the same as rTorrent-PS, and they work perfectly fine without it; the same is true the other way 'round. It's just that both projects unsurprisingly have synergies if used together, and some features *do* only work when both are present.

You absolutely **must** read the first three chapters *Overview*, *Installation Guide*, and *Configuration Guide* — *pyrocore* utilities won't work at all or not properly if you do not provide an adequate configuration, and also modify the *rTorrent* one to provide some essential data and commands. Once you got everything basically working, *User's Manual* will show you all the common commands and use-cases. Further chapters then explain more complex use-cases and features that might not appeal or apply to you.

To get in contact and share your experiences with other users of PyroScope, join the pyroscope-users mailing list or the inofficial `##rtorrent` channel on `irc.freenode.net`.

This is also the way to resolve any problems with or questions about your configuration and software installation. *Always* look into the *Trouble-Shooting Guide* as a first measure, which is often the fastest way to get back to a working system. That guide also explains how to efficiently report your problem when you cannot fix it yourself.

CHAPTER 1

Contents of This Manual

Overview

Introduction

pyrocore is part of the PyroScope family of projects, and offers a collection of tools for the *BitTorrent Protocol* and especially the *rTorrent* client. This includes:

- *Command Line Tools* for automation of common tasks, like *metafile creation*, and *filtering and mass-changing your loaded torrents*.
- rTorrent extensions like a *rTorrent Queue Manager* and statistics (*work in progress*).
- All this is based on the `pyrocore` Python package, that you can use to *Writing Your Own Scripts* for any special needs that aren't covered by the standard tools.

See the ScreenShotGallery if you want to get a first impression without installing the software.

To get in contact and share your experiences with other users of PyroScope, join the pyroscope-users mailing list or the inofficial `##rtorrent` channel on `irc.freenode.net`.

This is also the way to resolve any problems with or questions about your configuration and software installation. *Always* look into the *Trouble-Shooting Guide* as a first measure, which is often the fastest way to get back to a working system. That guide also explains how to efficiently report your problem when you cannot fix it yourself.

Glossary

To help you better understand this manual, here are the definitions of some key concepts used in it.

(download) item An item loaded into rTorrent.

field An attribute of a download item, e.g. `name`, `completed`, and `directory`. Most of these you know from *rTorrent* or *ruTorrent*, but *PyroScope* adds some of its own. They are used in conditions to filter items using the `rtcontrol` tool, and also name the things you want to print to the console when listing items. To get a full list, use the `rtcontrol --help-fields` command.

metafile The term *metafile* means the `.torrent` file – using 'torrent' is avoided intentionally, because it's often used ambiguously to mean *either* the metafile or the *data* of a download item.

XMLRPC The protocol used to remotely control a running rTorrent process. Note that support for XMLRPC is an option that must be activated when compiling the rTorrent binary, so make sure it's active in your installation when 'nothing works' for you. A quick way to check is calling the following command:

```
$ ldd $(command which rtorrent) | grep libxmlrpc.so
        libxmlrpc.so.3 => /home/pyroscope/.local/rtorrent/0.9.6-PS-1.0/lib/
↪libxmlrpc.so.3 ...
```

Quick Start Guide

Work through these chapters in order to get the software up and running, and to learn basic concepts of using the command line tools.

- *Installation Guide*
- *Configuration Guide*
- *User's Manual*

Consult the *Trouble-Shooting Guide* if anything goes wrong. *Reporting an Issue, or Requesting a Feature* explains how to provide feedback in case you encounter a serious problem, or are missing a feature.

Warning: If you do a fresh installation of *pyrocore* in addition to an existing *rTorrent* one, you will need to follow the instructions to *Adding Missing Data to Your rTorrent Session*, which fills in some data your already running rTorrent instance is missing otherwise! So do **not** skip that section.

Further Information & Customization

- *Tips & How-Tos* highlights some specific use-cases and might give you some inspiration when solving your own problems.
- Using *Advanced Features* requires some knowledge in the area Linux, Bash, and Python beyond a novice level, but they enable you to customize your setup even further and handle very specific use-cases.
- *Custom Python Code* tells you about *Writing Your Own Scripts* as an easy way to automate anything that the standard commands can't do. There are more ways for adding your own custom logic, amongst them *Defining Custom Fields* for adding user-defined fields, available in `rtcontrol` just like built-in ones.
- *Software Updates* explains how to get newer versions of this software after the initial installation.
- *References* provides details on technical background topics like XMLRPC, and links into the web with related information.

Installation Guide

This chapter presents you with different installation options. If you start with an unconfigured host, consider using the automated setup provided by the pimp-my-box project, which will install all you need for a fully working torrenting setup including a default configuration.

These are the steps for a manual installation:

As you can see, installing the software package itself can be done in two ways, choose one of them. Afterwards, the freshly installed software *must* be provided with a configuration, as described in the *Configuration Guide*.

Note: Unless otherwise indicated by using `sudo` or mentioning it in the text, installation commands should **not** be run as `root`, but in your normal user account, or else one you specifically created for installing *rTorrent* and `pyrocore`.

When commands *and* their output are both contained in a code box, `$` represents the command prompt of your shell, followed by the command you are supposed to enter. Do **not** enter the leading `$`!

Warning: The syntax of XMLRPC commands changed with rTorrent version 0.8.9, and continues to change. Make sure that the versions of rTorrent and PyroScope you plan to install or update to are actually compatible. There are compensation mechanisms in both projects, but there are limits to those — scan the respective changelogs for breaking changes.

pyrocore 0.5+ will no longer support the old syntax, and thus not work with *rTorrent* 0.8.x versions. *rTorrent* 0.9.6 has the old commands disabled by default, and only a special command line switch will enable them again, *for now*. Also, this documentation uses the new syntax (mostly).

Preparing Your Host

Installing Dependency Packages

Before installing *pyrocore*, some software packages need to be available on your machine, Python 2 among them.

On Debian-type systems (Debian, Ubuntu, Raspbian, ...), the following ensures you have everything you need, including packages necessary for installing from source:

```
sudo apt-get install python python-dev python-virtualenv python-pip \
    python-setuptools python-pkg-resources git build-essential
```

On other Linux distributions, see the following section for further hints.

If you want to install everything in a dedicated user account, e.g. for security reasons, this will create a `rtorrent` user when entered into a `root` shell:

```
groupadd rtorrent
useradd -g rtorrent -G rtorrent,users -c "Torrent User" -s /bin/bash --create-home␣
↪rtorrent
chmod 750 ~rtorrent
su - rtorrent -c "mkdir -p ~/bin"
```

Using such a dedicated account also makes sure you don't need to have fear this software does anything malicious — if it did, it'd be contained in that account. It also makes deinstallation or start-from-zero way less of a hassle.

Installing Python2

For *Debian* and derivatives, the `apt-get` command in the previous section already took care of everything.

Other Linux distributions usually come equipped with a Python 2.7 interpreter, but on very new releases, Python 3 may be the default and Python 2.7 just an option. In case you need to install Python 2, refer to Installing Python on Linux and consider using pyenv.

The following shows how you can check what version you have as the default (the sample output is from *Ubuntu 15.04*):

```
$ /usr/bin/python --version
Python 2.7.9
```

Try calling `/usr/bin/python2` in case the above shows a `3.*` version.

Installing the *pyrocore* Package

Installing the software package itself can be done in two ways, choose one of them.

Warning: If you want to switch over from an old installation to one in `~/.local`, then *move that old directory away*, before installation! Like this:

```
( cd ~/lib && mv pyroscope pyroscope-$(date +'%Y-%m-%d').bak )
```

Your existing configuration and data is not affected by this, but make sure you read the **migration instructions** in *Software Updates*.

Option 1: Installing from GitHub

The recommended way to install this software is directly from its GitHub repository. To do that, use the following commands:

```
mkdir -p ~/bin ~/.local
git clone "https://github.com/pyroscope/pyrocore.git" ~/.local/pyroscope

# Pass "/usr/bin/python2", or whatever else fits, to the script as its
# 1st argument, if the default of "/usr/bin/python" is not a suitable
# version.
~/.local/pyroscope/update-to-head.sh

# Check success
pyroadmin --version  # call "exec $SHELL -l" if this fails, and retry
```

You can choose a different install directory, just change the paths accordingly. If then anything fails, stop changing things and stick to the trodden path.

If you previously had no `~/bin` directory, call `exec $SHELL -l` to register it in the `PATH` of your current terminal session – especially if you see an error message like `pyroadmin: command not found`.

If everything went OK, continue with the *Configuration Guide*.

Option 2: Installing from PyPI

If you chose to install a release version from the Python package repository (PyPI), the *most simple but not best way* is calling `pip install --user -U pyrocore`, and make sure `$HOME/.local/bin` is in your `$PATH`. This way is OK if you just want to use the tools for metafile handling, i.e. `mktor`, `chtor`, and `lstor`, but not the *rTorrent* tools.

The **recommended way using a dedicated virtualenv** goes like this:

```
mkdir -p ~/bin ~/.local
/usr/bin/virtualenv --no-site-packages $_/pyroscope
cd $_
ln -nfs python bin/python-pyrocore
ln -nfs $PWD/bin/python-pyrocore ~/bin
. bin/activate
xargs -n1 pip install -U <<<"pip setuptools wheel"
pip uninstall -y distribute 2>/dev/null
pip install -U "pyrocore[templating]"
ln -nfs $(egrep -l '(from.pyrocore.scripts|entry_point.*pyrocore.*console_scripts)'
↪$PWD/bin/*) ~/bin

# Check success
pyroadmin --version  # call "exec $SHELL -l" if this fails, and retry
```

If you previously had no `~/bin` directory, call `exec $SHELL -l` to register it in the `PATH` of your current terminal session – especially if you see an error message like `pyroadmin: command not found`.

If everything went OK, continue with the *Configuration Guide*.

Configuration Guide

Introduction

After you installed the software as described in the previous chapter, you need to add personal configuration that is loaded from the directory `~/.pyroscope` containing the files `config.ini` and `config.py`. A default set can be automatically created for you, see below for details.

For simple setups, you only need to edit the plain text file `config.ini`. The script `config.py` allows much more detailed control over complex setups, at the price of you knowing at least the basics of the Python programming language. See *Advanced Features* for that.

Note: For a fresh installation of this software in addition to an *existing* rTorrent one, you will also need to back-fill some data that your already running rTorrent instance is missing otherwise. If you skip this step, item filtering in `rtcontrol` and other tools will *not* work correctly for existing items. More on that below.

In summary, you'll perform these steps, explained in the sections that follow:

1. Create a directory with the default configuration.
2. Edit `~/.pyroscope/config.ini` to adapt it to your needs, e.g. add tracker aliases.
3. Modify your `~/.rtorrent.rc` to integrate necessary settings.
4. Back-fill some data into the *rTorrent* session.

To get in contact and share your experiences with other users of PyroScope, join the pyroscope-users mailing list or the inofficial `##rtorrent` channel on `irc.freenode.net`.

This is also the way to resolve any problems with or questions about your configuration and software installation. *Always* look into the *Trouble-Shooting Guide* as a first measure, which is often the fastest way to get back to a working system. That guide also explains how to efficiently report your problem when you cannot fix it yourself.

Creating a set of default configuration files

To create your own configuration, the best way is to start from the default files that are part of your PyroScope installation. To create them at the default location `~/.pyroscope`, simply call this command:

```
pyroadmin --create-config
```

Note that you can delete any defult setting from `config.ini` that you don't want changed. These defaults are *always* loaded before your own settings, from a copy the software keeps and updates.

Deleting unchanged defaults has the advantage that on software updates, you'll automatically get the newer version of settings, as soon as they're updated. The created `config.ini.default` file is just for reference, and will be overwritten on updates.

If you need several distinct configuration sets, just add the `--config-dir` option to commands like so:

```
pyroadmin --create-config --config-dir ~/rtorrent/special/.pyroscope
```

To view your loaded configuration with all the system defaults added, use this (again, the `--config-dir` option allows non-default configuration locations):

```
pyroadmin --dump-config
```

To start over with a pristine set of configuration files, and remove any stale ones, add the `--remove-all-rc-files` option:

```
pyroadmin --remove-all-rc-files --create-config
```

Be aware that this *really* removes **any** `*.rc` and `*.rc.default` file in `~/.pyroscope` and its subfolder `rtorrent.d`, before writing a new set of files.

Note: Each *PyroScope* configuration file is accompanied by a matching `*.default` file that contains the system defaults at the time you last called the `pyroadmin --create-config` command. These are over-written on repeated calls (unlike the real config files), and are for informational purposes only.

For the *rTorrent* configuration files (`rtorrent-pyro.rc[.default]` and files in `rtorrent.d`), the rules are different. These files change frequently, so the `*.default` versions are loaded usually, and you get an up-to-date version on a *rTorrent* restart.

You can ignore specific files in `rtorrent.d` if they don't fit or you want to provide your own version under *another* name. See the files themselves for instructions.

Setting values in 'config.ini'

The main configuration file consists of sections, led by a `[section]` header and followed by `name: value` entries; `name = value` is also accepted. Longer values can be broken into several lines and the continuation lines must be indented (start with a space). Note that leading whitespace is removed from values.

Lines beginning with a semicolon (`;`), a hash mark (`#`), or the letters `REM` (uppercase or lowercase) will be ignored and can be used for comments. You cannot append a comment to an option line, a comment **MUST** start at the beginning of a line!

As an example, this is a very minimal configuration file:

```
# PyroScope configuration file

[GLOBAL]
# Note that the "config_dir" value is provided by the system!
config_script = %(config_dir)s/config.py
rtorrent_rc = ~/.rtorrent.rc

[ANNOUNCE]
# Add alias names for announce URLs to this section; those aliases are used
# at many places, e.g. by the "mktor" tool

# Public trackers
PBT     = http://tracker.publicbt.com:80/announce
          udp://tracker.publicbt.com:80/announce
OBT     = http://tracker.openbittorrent.com:80/announce
          udp://tracker.openbittorrent.com:80/announce
Debian  = http://bttracker.debian.org:6969/announce
```

Note: *For advanced users:* Values can contain format strings of the form `%(name)s` which refer to other values in the same section, or values in the `[DEFAULT]` section.

Extending your '.rtorrent.rc'

You need either a `network.scgi.open_local` or `network.scgi.open_port` specification in your rTorrent configuration, else XMLRPC cannot work; `network.scgi.open_local` is preferable since more secure. Furthermore, you need to provide the path to a session directory via `session.path`. See the *rTorrent* documentation for details.

Note: Using `network.scgi.open_port` means *any* user on the machine you run *rTorrent* on can execute *arbitrary* commands with the permission of the *rTorrent* runtime user. Most people don't realize that, now you do! Also, **never** use any other address than `127.0.0.1` with it.

For the `loaded` and `completed` fields to work, as well as the `started`, `leechtime` and `seedtime` ones, you also have to add these commands (note that most settings actually reside in an included file):

```
#
# PyroScope SETTINGS
#

# Set "pyro.extended" to 1 to activate rTorrent-PS features!
# LEAVE THIS AT 0 IF YOU RUN A VANILLA rTorrent!
method.insert = pyro.extended, value|const, 0

# Set "pyro.bin_dir" to the "bin" directory where you installed the pyrocore tools!
# Make sure you end it with a "/"; if this is left empty, then the shell's path is␣
↪searched.
method.insert = pyro.bin_dir, string|const,
```

```
# Remove the ".default" if you want to change something (else your changes
# get over-written on update, when you put them into ``*.default`` files).
import = ~/.pyroscope/rtorrent-pyro.rc.default

# TORQUE: Daemon watchdog schedule
# Must be activated by touching the "~/.pyroscope/run/pyrotorque" file!
# Set the second argument to "-v" or "-q" to change log verbosity.
schedule = pyro_watchdog,30,300,"pyro.watchdog=~/.pyroscope,"
```

See this rtorrent.rc and the _rtlocal.rc file it includes for a complete example, including some view changes regarding sort order made possible by the additional custom fields.

Note: Remember to restart *rTorrent* for the new configuration to take effect. If you also installed the rtorrent-ps distribution of *rTorrent*, do not forget to activate the extended features available with it, by setting `pyro.extended` to `1` in the above configuration.

Adding Missing Data to Your rTorrent Session

Now that you have the additional configuration, *newly loaded* items will get the correct values set – but existing items are still missing them, and so those items will *not* always be filtered correctly. If you just started with a fresh install and have no items added to rTorrent yet, you can ignore this section.

To add the missing data, call these commands:

```
# Make a full, current backup of the session data
rtxmlrpc -q session.save
tar cvfz ~/session-backup-$(date +'%Y-%m-%d').tgz \
    $(echo $(rtxmlrpc session.path)/ | tr -s / /)*.torrent*

# Set missing "loaded" times to that of the .torrent file
rtcontrol '!*"*' loaded=0 -q -sname -o 'echo "$(name)s"\ntest -f "$(metafile)s" &&␣
↪rtxmlrpc -q d.custom.set $(hash)s tm_loaded \$(\
    ls -l --time-style "+%%s" "$(metafile)s" \
    | cut -f6 -d" ")\nrtxmlrpc -q d.save_full_session $(hash)s' | bash

# Set missing "completed" times to that of the data file or directory
rtcontrol '!*"*' completed=0 done=100 path=\! is_ghost=no -q -sname -o 'echo "$(name)s
↪"\ntest -e "$(realpath)s" && rtxmlrpc -q d.custom.set $(hash)s tm_completed \$(\
    ls -ld --time-style "+%%s" "$(realpath)s" \
    | cut -f6 -d" ")\nrtxmlrpc -q d.save_full_session $(hash)s' | bash
```

It's safe to call them repeatedly, since existing values are kept unchanged.

To check, use the command `rtcontrol completed=-1d -scompleted` which should now show your completed downloads of the last 24 hours, in order.

Continue with the *User's Manual* to get to know all the commands.

User's Manual

This chapter provides an overview of all the command line tools and their everyday use, focussing on `rtcontrol` as the most powerful of them. The following chapters then go into more advanced use-cases and features.

Command Line Tools

Overview of CLI Tools

`rtcontrol` is the work-horse for rTorrent automation, it takes filter conditions of the form ‹`field`›=‹`value`› and selects a set of download items according to them. That result can then be printed to the console according to a specified format, or put into any rTorrent view for further inspection. You can also take some bulk action on the selected items, e.g. starting, stopping, or deleting them.

`rtxmlrpc` sends single XMLRPC commands to rTorrent, and `rtmv` allows you to move around the data of download items in the file system, while continuing to seed that data.

The following commands help you with managing metafiles:

- `lstor` safely lists their contents in various formats.
- `mktor` creates them, with support for painless cross-seeding.
- `chtor` changes existing metafiles, e.g. to add fast-resume information.
- `hashcheck` simply checks data against a given metafile's piece hashes.

`pyrotorque` is a companion daemon process to rTorrent that handles automation tasks like queue management, instant metafile loading from a directory tree via file system notifications, and other background tasks.

`pyroadmin` is a helper for administrative tasks (mostly configuration handling). and `rtevent` is experimental and incomplete.

Bash Completion

If you don't know what bash completion is, or want to handle this later, you can skip to *Common Options*.

Using completion

In case you don't know what `bash` completion looks like, watch this...

Every time you're unsure what options you have, you can press `TAB` twice to get a menu of choices, and if you already know roughly what you want, you can start typing and save keystrokes by pressing `TAB` once, to complete whatever you provided so far.

So for example, enter a partial command name like `rtco` and then `TAB` to get "`rtcontrol`", then type `--` followed by 2 times `TAB` to get a list of possible command line options.

Activating completion

To add `pyrocore`'s completion definitions to your shell, call these commands:

```
pyroadmin --create-config
touch ~/.bash_completion
grep /\.pyroscope/ ~/.bash_completion >/dev/null || \
    echo >>.bash_completion ". ~/.pyroscope/bash-completion.default"
. /etc/bash_completion
```

After that, completion should work, see the above section for things to try out.

Note: On *Ubuntu*, you need to have the `bash-completion` package installed on your machine. Other Linux systems will have a similar pre-condition.

Common Options

All commands share some common options:

```
--version           show program's version number and exit
-h, --help          show this help message and exit
-q, --quiet         omit informational logging
-v, --verbose       increase informational logging
--debug             always show stack-traces for errors
--config-dir=DIR    configuration directory [~/.pyroscope]
```

Also see the *PyroScope CLI Tools Usage* section for an automatically generated and thus comprehensive listing of all the current options.

mktor

mktor creates `*.torrent` files (metafiles), given the **path to the data** in a file, directory, or named pipe (more on that below) and a **tracker URL or alias name** (see *Setting values in 'config.ini'* on how to define aliases). Optionally, you can also set an additional comment and a different name for the resulting torrent file. Peer exchange and DHT can be disabled by using the `--private` option.

If you want to create metafiles in bulk, use one of the many options a Linux shell offers you, among them:

- *Anything* in the current directory:

```
ls -1 | xargs -d$'\n' -I{} mktor -p -o /tmp "{}" "$ANNOUNCE_URL"
```

- Just for directories:

```
find . -mindepth 1 -maxdepth 1 -type d \! -name ".*" -print0 | sort -z \
    | xargs -0I{} mktor -p "{}" "$ANNOUNCE_URL"
```

If you create torrents for different trackers, they're *automatically enabled for cross-seeding*, i.e. you can load several torrents for exactly the same data into your client. For the technically inclined, this is done by adding a unique key so that the info hash is always different. Use the `--no-cross-seed` option to disable this. You can also set the 'source' field many trackers use for unique info hashes, use `-s info.source=LABEL` for that.

To exclude files stored on disk from the resulting torrent, use the `--exclude` option to extend the list of standard glob patterns that are ignored. These standard patterns are: `core`, `CVS`, `.*`, `*~`, `*.swp`, `*.tmp`, `*.bak`, `[Tt]humbs.db`, `[Dd]esktop.ini`, and `ehthumbs_vista.db`.

The `--fast-resume` option creates a second metafile `*-resume.torrent` that contains special entries which, when loaded into rTorrent, makes it skip the redundant hashing phase (after all, you hashed the files just now). It is **very** important to upload the *other* file without `resume` in its name to your tracker, else you cause leechers using rTorrent problems with starting their download.

As a unique feature, if you want to change the root directory of the torrent to something different than the basename of the data directory, you can do so with the `--root-name` option. This is especially useful if you have hierarchical paths like `documents/2009/myproject/specs` - normally, all the context information but `specs` would be

lost on the receiving side. Just don't forget to provide a symlink in your download directory with the chosen name that points to the actual data directory.

Very few people will ever need that, but another advanced feature is concurrent hashing — if the first argument is a named pipe (see the `mkfifo` man page), the filenames to be hashed are read from that pipe. These names must be relative to the directory the named pipe resides in, or put another way, the named pipe has to be created in the same directory as the files to be hashed. For example, this makes it possible to hash files as they arrive via FTP or are transcoded from one audio format to another, reducing overall latency. See the fifotest script for a demonstration of the concept.

lstor

lstor lists the contents of bittorrent metafiles. The resulting output looks like this:

```
NAME pavement.torrent
SIZE 3.6 KiB (0 * 32.0 KiB + 3.6 KiB)
HASH 2D1A7E443D23907E5118FA4A1065CCA191D62C0B
URL  http://example.com/
PRV  NO (DHT/PEX enabled)
TIME 2009-06-06 00:49:52
BY   PyroScope 0.1.1

FILE LISTING
pavement.py                                                              3.6 KiB

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
NAME tests.torrent
SIZE 2.6 KiB (0 * 32.0 KiB + 2.6 KiB)
HASH 8E37EB6F4D3807EB26F267D3A9D31C4262530AB2
URL  http://example.com/
PRV  YES (DHT/PEX disabled)
TIME 2009-06-06 00:49:52
BY   PyroScope 0.1.1

FILE LISTING
pyroscope tests/
    test_bencode.py                                                      2.6 KiB
```

`lstor` has these options:

```
--reveal       show full announce URL including keys
--raw          print the metafile's raw content in all detail
-V, --skip-validation
               show broken metafiles with an invalid structure
--output=KEY,KEY1.KEY2,...
               select fields to print, output is separated by TABs;
               note that __file__ is the path to the metafile,
               __hash__ is the info hash, and __size__ is the data
               size in byte
```

Starting with v0.3.6, you can select to output specific fields from the metafile, like this:

```
$ lstor -qo __hash__,info.piece\ length,info.name *.torrent
00319ED92914E30C9104DA30BF39AF862513C4C8    262144  Execute My Liberty - The Cursed␣
↪Way -- Jamendo - OGG Vorbis q7 - 2010.07.29 [www.jamendo.com]
```

This can also be used to rename ‹infohash›.torrent metafiles from a session directory to a human readable name, using parts of the hash to ensure unique names:

```
ls -1 *.torrent | egrep '^[0-9a-fA-F]{40}\.torrent' | while read i; do
    humanized="$(lstor -qo info.name,__hash__ "$i" | awk -F$'\t' '{print $1"-"substr(
↪$2,1,7)}')"
    mv "$i" "$humanized.torrent"
done
```

And to see a metafile with all the guts hanging out, use the --raw option:

```
{'announce': 'http://tracker.example.com/announce',
 'created by': 'PyroScope 0.3.2dev-r410',
 'creation date': 1268581272,
 'info': {'length': 10,
          'name': 'lab-rats',
          'piece length': 32768,
          'pieces': '<1 piece hashes>',
          'x_cross_seed': '142e0ae6d40bd9d3bcccdc8a9683e2fb'},
 'libtorrent_resume': {'bitfield': 0,
                       'files': [{'completed': 0,
                                  'mtime': 1283007315,
                                  'priority': 1}],
                       'peers': [],
                       'trackers': {'http://tracker.example.com/announce': {'enabled
↪': 1}}},
 'rtorrent': {'chunks_done': 0,
              'complete': 0,
              'connection_leech': 'leech',
              'connection_seed': 'seed',
              'custom': {'activations': 'R1283007474P1283007494R1283007529P1283007537
↪',
                         'kind': '100%_',
                         'tm_loaded': '1283007442',
                         'tm_started': '1283007474'},
              'custom1': '',
              'custom2': '',
              'custom3': '',
              'custom4': '',
              'custom5': '',
              'directory': '~/rtorrent/work',
              'hashing': 0,
              'ignore_commands': 1,
              'key': 357633323,
              'loaded_file': '~/rtorrent/.session/
↪38DE398D332AE856B509EF375C875FACFA1C939F.torrent',
              'priority': 2,
              'state': 0,
              'state_changed': 1283017194,
              'state_counter': 4,
              'throttle_name': '',
              'tied_to_file': '~/rtorrent/watch/lab-rats.torrent',
              'total_uploaded': 0,
              'views': []}}
```

chtor

chtor is able to change common attributes of a metafile, or clean any non-standard data from them (namely, rTorrent session information).

Note that `chtor` automatically changes only those metafiles whose existing announce URL starts with the scheme and location of the new URL when using `--reannounce`. To change *all* given metafiles unconditionally, use the `--reannounce-all` option and be very sure you provide only those files you actually want to be changed.

`chtor` only rewrites metafiles that were actually changed, and those changes are first written to a temporary file, which is then renamed.

rtcontrol

Purpose

rtcontrol allows you to select torrents loaded into rTorrent using various filter conditions. You can then either display the matches found in any rTorrent view for further inspection, list them to the console using flexible output formatting, or perform some management action like starting and stopping torrents. *Using 'rtxmlrpc'* shows examples for sending commands that don't target a specific item.

For example, the command `rtcontrol up=+0 up=-10k` will list all torrents that are currently uploading any data, but at a rate of below 10 KiB/s. See the *'rtcontrol' Examples* for more real-world examples, and the following section on basics regarding the filter conditions.

Filter Conditions

Filter conditions take the form ‹ `field` ›=‹ `value` ›, and by default all given conditions must be met (AND). If a field name is omitted, `name` is assumed. Multiple values separated by a comma indicate several possible choices (OR). `!` in front of a filter value negates it (NOT). Use uppercase `OR` to combine multiple alternative sets of conditions. And finally brackets can be used to group conditions and alter the default "AND before OR" behaviour; be sure to separate both the opening and closing bracket by white space from surrounding text. `NOT` at the start of a bracket pair inverts the contained condition.

For string fields, the value is a glob pattern which you are used to from shell filename patterns (`*`, `?`, `[a-z]`, `[!a-z]`); glob patterns must match the whole field value, i.e. use `*...*` for 'contains' type searches. To use regex matches instead of globbing, enclose the pattern in slashes (`/regex/`). Since regex can express anchoring the match at the head (`^`) or tail (`$`), they're by default of the 'contains' type. All string comparisons are case-ignoring.

For numeric fields, a leading + means greater than, a leading − means less than (just like with the standard `find` command).

Selection on fields that are lists of tags or names (e.g. `tagged` and `views`) works by just providing the tags you want to search for. The difference to the glob patterns for string fields is that tagged search respects word boundaries (whitespace), and to get a match the given tag just has to appear anywhere in the list (`bar` matches on `foo bar baz`).

In time filtering conditions (e.g. for the `completed` and `loaded` fields), you have three possible options to specify the value:

1. time deltas in the form "`<number><unit>...`", where unit is a single upper- or lower-case letter and one of `Y`ear, `M`onth, `W`eek, `D`ay, `H`our, m`I`nute, or `S`econd. The order is important (`y` before `m`), and a + before the delta means *older than*, while − means *younger than*.

   Example: `-1m2w3d`

2. a certain date and time in human readable form, where the date can be given in ISO (`Y-M-D`), American (`M/D/Y`), or European (`D.M.Y`) format. A date can be followed by a time, with minutes and seconds optional and separated by `:`. Put either a space or a `T` between the date and the time.

   Example: `+2010-08-15t14:50`

3. absolute numerical UNIX timestamp, i.e. what `ls -l --time-style '+%s'` returns.

   Example: `+1281876597`

See *Useful Filter Conditions* for some concrete examples with an explanation of what they do.

Annealing Results

Using the `--anneal` option, you can add some pre-defined post-processing steps that modify the current result set. You can use this option several times to combine processing steps in the order given on the command line. Sorting is done first, and if anything changes, the modified result is sorted again before applying the next step. Note that any `--select` restrictions are applied *after* annealing.

The available processing methods are these:

dupes+ Adds any loaded item that shares the same base directory with any existing result item, or points to the same file. Note that symlinks are followed, but hardlinks are always considered independent (which they are when deleted). This is especially useful in combination with `--cull` to avoid leaving items with some or all of their files gone.

dupes- Removes items from the result that share the same path with any other loaded item, as described for `dupes+`, that is not *also* part of the result. Again, combination with `--cull` is a typical use-case, to avoid deleting data of items that still need to be seeded, when only some of a set of duplicated items meet the deletion criteria.

dupes= Removes any items from the result that are *not* dupes, as defined above, leaving only the dupes. Combine with `invert` to only get singular items.

invert Invert the current selection, i.e. select any item in the *original* result (before any annealing happened) that is not in the *current* selection.

unique Ensures that only the *first* item in the result set having the same name as other items *in the result set* is kept. The others are removed. Note that unlike with 'dupes', the scope here is only the current result set, not *all* loaded items.

> **Warning:** If you use options that cause `rtcontrol` to request only a subset of all loaded items, then all `dupes*` methods will produce results that might be unexpected, since they look at *all* available items, not just the selected ones. And 'all' is different if you change the view, or use the `-Q` option – for that reason, you'll get a warning if you mix `-A` with these.

rtxmlrpc

rtxmlrpc allows you to call raw XMLRPC methods on the rTorrent instance that you have specified in your configuration. See the *usage information* for available options.

The method name and optional arguments are provided using standard shell rules, i.e. where you would use `^X throttle_down=slow,120` in rTorrent you just list the arguments in the usual shell way (`rtxmlrpc throttle_down slow 120`). The rTorrent format is also recognized though, but without any escaping rules (i.e. you cannot have a `,` in your arguments then).

Remember that almost all commands require a 'target' as the first parameter in newer rTorrent versions, and you have to provide that explicitly. Thus, it must be `rtxmlrpc view.size '' main`, with an extra empty argument – otherwise you'll get a `Unsupported target type found` fault.

There are some special ways to write arguments of certain types: `+‹number›` and `-‹number›` send an integer value, `@‹filename›` or `@-` (for stdin) reads the file's content into a XMLRPC binary, and finally `[‹item1›⟨,‹item2›,...⟩]` produces an array of strings. These typed arguments only cover some common use-cases, at some point you have to write Python code to build up more intricate data structures.

To get a list of available methods, just call `rtxmlrpc system.listMethods`. The *Using 'rtxmlrpc'* section shows some typical examples for querying global information and controlling rTorrent behaviour.

rtmv

With *rtmv*, you can move actively seeded data around at will. Currently, it only knows one mode of operation, namely moving the data directory or file and leave a symlink behind in its place (or fixing the symlink if you move data around a second time). Watch this example that shows what's going on internally:

```
~/bt/rtorrent/work$ rtmv lab-rats /tmp/ -v
DEBUG    Found "lab-rats" for 'lab-rats'
INFO     Moving to "/tmp/lab-rats"...
DEBUG    Symlinking "~/bt/rtorrent/work/lab-rats"
DEBUG    rename("~/bt/rtorrent/work/lab-rats", "/tmp/lab-rats")
DEBUG    symlink("/tmp/lab-rats", "~/bt/rtorrent/work/lab-rats")
INFO     Moved 1 path (skipped 0)

$ rtmv /tmp/lab-rats /tmp/lab-mice -v
DEBUG    Item path "~/bt/rtorrent/work/lab-rats" resolved to "/tmp/lab-rats"
DEBUG    Found "lab-rats" for '/tmp/lab-rats'
INFO     Moving to "/tmp/lab-mice"...
DEBUG    Re-linking "~/bt/rtorrent/work/lab-rats"
DEBUG    rename("/tmp/lab-rats", "/tmp/lab-mice")
DEBUG    remove("~/bt/rtorrent/work/lab-rats")
DEBUG    symlink("/tmp/lab-mice", "~/bt/rtorrent/work/lab-rats")
```

From the second example you can see that you can rename actively seeding downloads in mid-flight, i.e. to fix a bad root directory name.

You can use `rtmv` in combination with `rtcontrol --call` for very flexible completion moving. To facilitate this, if there is a double slash `//` in the target path, it is always interpreted as a directory (i.e. you cannot rename the source file in that case), and the partial path after the `//` is automatically created. This can be used in completion moving, to create hierarchies for dynamic paths built from `rtcontrol` fields. Since the part before the `//` has to exist beforehand, this won't go haywire and create directory structures just anywhere.

Note: Future modes of operation will include copying instead of moving, moving and fixing the download directory in rTorrent (like classical rtorrent completion event handling), and moving across devices (i.e. copying and then deleting).

---

rtevent

Not yet implemented

rtevent handles rTorrent events and provides common implementations for them, like completion moving. See EventHandling for details on using it.

'rtcontrol' Examples

Useful Filter Conditions

The following *rtcontrol Filter Conditions* give you a hint on what you can do, and some building blocks for more complex conditions.

`*HDTV*` Anything with "HDTV" in its name

`/s\d+e\d+/` Anything with typical TV episode numbering in its name (regex match)

`ratio=+1` All downloads seeded to at least 1:1

`xfer=+0` All active torrents (transferring data)

`up=+0` All seeding torrents (uploading data)

`down=+0 down=-5k` Slow torrents (downloading, but with < 5 KiB/s)

`down=0 is_complete=no is_open=yes` Stuck torrents

`size=+4g` Big stuff (DVD size or larger)

`is_complete=no` Incomplete downloads

`is_open=y is_active=n` Paused items

`is_ghost=yes` Torrents that have no data (were never started or lost their data; *since v0.3.3*)

`alias=obt` Torrents tracked by *openbittorrent.com* (see *Configuration Guide* on how to add aliases for trackers)

`'path=!'` Has a non-empty *path*

`ratio=+1 realpath=\!/mnt/*` 1:1 seeds not on a mounted path (i.e. likely on localhost)

`completed=+2w` Completed more than 2 weeks ago (*since v0.3.4*)

`tagged=` Not tagged at all (*since v0.3.5*)

`tagged=\!` Has at least one tag (*since v0.3.5*)

`tagged=foo,bar` Tagged with "foo" or "bar" (*since v0.3.5*) — tags are white-space separated lists of names in the field `custom_tags`

`tagged==highlander` *Only* tagged with "highlander" and nothing else (*since v0.3.6*)

`kind=flac,mp3` Music downloads (*since v0.3.6*)

`files=sample/*` Items with a top-level *sample* folder (*since v0.3.6*)

`ratio=+2.5 OR seedtime=+1w` Items seeded to 5:2 **or** for more than a week (*since v0.3.6*)

`alias=foo [ ratio=+2.5 OR seedtime=+7d ]` The same as above, but for one tracker only (*since v0.3.7*)

`traits=avi traits=tv,movies` TV or movies in AVI containers (*since v0.3.7*)

Note that the `!` character has to be escaped in shell commands. For a current full list of all the field names and their meaning, see the output of the `--help-fields` option of *rtcontrol* which gives you a complete list for your installation.

Integrating 'rtcontrol' into the Curses UI

Anyone who ever dreamt about a search box in their rtorrent UI, dream no more...

Note: You already have the following configuration commands, if you followed the *Configuration Guide*.

Just add this to your `.rtorrent.rc`:

```
# VIEW: Use rtcontrol filter (^X s=KEYWORD, ^X t=TRACKER, ^X f="FILTER")
method.insert = s,simple|private,"execute.nothrow=rtcontrol,--detach,-qV,\"$cat=*,
↪$argument.0=,*\""
method.insert = t,simple|private,"execute.nothrow=rtcontrol,--detach,-qV,\"$cat=\\\
↪"alias=\\\",$argument.0=\""
method.insert = f,simple|private,"execute.nothrow=rtcontrol,--detach,-qV,$argument.0="
```

You can of course add as many commands as you like, and include sorting options and whatever else `rtcontrol` offers.

The 'trick' here is the `-V` (`--view-only`) option, which shows the selection result in a rTorrent view instead of on the console. You can add this to any query you execute on the command line, and then interactively work with the result. The above commands are just shortcuts for common use-cases, directly callable from the curses UI.

Reports

Using bash Aliases for Common Reports

You might want to add the following alias definitions to your `~/.bashrc`:

```
alias rt2days="rtcontrol -scompleted -ocompleted,is_open,up.sz,ratio,alias,name␣
↪completed=-2d"
alias rtls="rtcontrol -qo '{{chr(10).join([d.directory+chr(47)+x.path for x in d.
↪files])|h.subst(chr(47)+chr(43),chr(47))}}'"
```

`rt2days` gives the completion history of the last 48 hours, and `rtls` lets you create lists of files just like `ls`:

```
$ rtls /a.boy/ | xargs -d'\n' ls -lgGh
-rw-r----- 1 702M Mar  7 17:42 /var/torrent/work/A_Boy_and_His_Dog.avi
```

If you feed the list of paths into normal `ls` as shown, you have all the usual options available to you.

Note: See the rt-alias.sh file of the pimp-my-box project for these and some more aliases.

Defining and Using Custom Output Formats

Before describing the possible options for output formatting in more details below, here's a short overview of the possible methods, each with an example:

- `size.sz,name` — simple field lists, possibly with format specifiers; in the output, fields are separated by a TAB character.

- `%(size.sz)s %(name)s` — string interpolation, i.e. like the above lists, but interspersed with literal text instead of TABs.
- `{{d.size|sz}} {{d.name}}` — Tempita templates, see *Using Output Templates* for more details.
- `file:template.tmpl` — File URLs that point to a template file, which is especially useful for more complicated templates. The filenames can be absolute (starting with a `/`), relative to your home (starting with a `~`), or relative to `templates` in the configuration directory (anything else).
- «formatname» — A name of a custom format from the `[FORMATS]` configuration section, see `~/.pyroscope/config.ini.default` for the predefined ones (including the special `default` format).

Starting with version 0.3.5, you can define custom output formats and print column headers, the `rt2days` example from the previous section becomes this:

```
alias rt2days="rtcontrol --column-headers -scompleted -ocompletion completed=-2d"
```

You need to define the custom output format used there, so also add this to your `~/.pyroscope/config.ini`:

```
[FORMATS]
# Custom output formats
completion = $(completed.raw.delta)13.13s $(leechtime)9.9s $(is_open)4.4s $(up.sz)10s/
↪s $(ratio.pc)5d$(pc)s $(alias)-8s $(kind_50)-4.4s  $(name)s
```

See PyFormat for a description how the formatting options work, and notice that `$` is used instead of `%` here, because `%` has a special meaning in INI files. For the same reason, a single `%` in the final output becomes `$(pc)s` in the configuration (`pc` is a system field that is simply a percent sign).

You can also append one or more format specifiers to a field name, separated by a `.`. These take the current value and transform it — in the above example `.raw.delta` means "take an unformatted time value and then convert it into a time delta relative to just now." The option `--help-fields` lists the available format specifiers.

Then, calling `rt2days -q` will print something like this:

```
 COMPLETED LEECHTIME IS_O         UP/s RATIO% ALIAS    KIND  NAME
1d 21h ago   10m  2s  OPN    0 bytes/s   100% SeedBox  rar   lab-rats
```

And with version 0.3.6 installed, you can create a full listing of all the files you have loaded into rTorrent using the predefined format "`files`":

```
$ rtcontrol \* -ofiles | less
STP    1970-01-01 01:00:00   25.6 MiB Execute My Liberty - The Cursed Way -- Jamendo -
↪ OGG Vorbis q7 - 2010.07.29 [www.jamendo.com] {Jamendo}
       2010-08-21 01:25:27    2.0 MiB | 01 - Midnight (Intro).ogg
       ...
       2010-08-21 01:25:27   48.7 KiB | [cover] Execute My Liberty - The Cursed Way.
↪jpg
                                     = 9 file(s) [ogg txt]
...
```

And finally, from version 0.4.1 onwards, you can use a full templating language instead of the simple field lists or string interpolation described above, more on that in *Using Output Templates*.

Statistics

Printing Some Statistics to the Terminal

Create a list of all your trackers and how many torrents are loaded for each:

```
rtcontrol -q -o alias -s alias \* | uniq -c
```

You can easily modify this by using conditions other than `*`, e.g. show the count of fully seeded downloads using `ratio=+1`. Or try the same command with `traits` instead of `alias` (version 0.3.7 only).

The total amount of data you have loaded in GiB:

```
rtcontrol -qosize \* | awk '{ SUM += $1} END { print SUM/1024/1024/1024 }'
```

The amount uploaded per tracker:

```
rtcontrol -qo alias,uploaded // \
    | awk '{arr[$1]+=$2} END {for (i in arr) {printf "%20s %7.1f GiB\n",i,arr[i]/1024^
↪3}}' \
    | sort -bnk2
```

Starting with version 0.4.1, you can also request a statistical summary of your numerical output columns, like this:

```
$ rtcontrol -qo size.sz,uploaded.sz,ratio.pc --summary "a*"
      SIZE    UPLOADED      RATIO
  14.5 GiB     9.3 GiB      2592.0 [SUM of 32 item(s)]
 462.4 MiB   298.9 MiB      81.0 [AVG of 32 item(s)]
```

Normalized Histogram of Ratio Distribution

The following will create a normalized histogram of ratio distribution of your loaded torrents. Each bar indicates the percentage of items in a ratio class (i.e. the first bar shows ratios up to 1).

```
rtcontrol alias=* -qo ratio -s ratio >/tmp/data \
    && octave -q --persist --eval \
             "load /tmp/data; hist(data, $(tail -n1 /tmp/data), 100); print -dpng /
↪tmp/ratio.png"
```

You need to have Octave installed, on Debian/Ubuntu all you need is `sudo aptitude install octave3.0`.

Performing Management Tasks

Fixing Items With an Empty "Base Path"

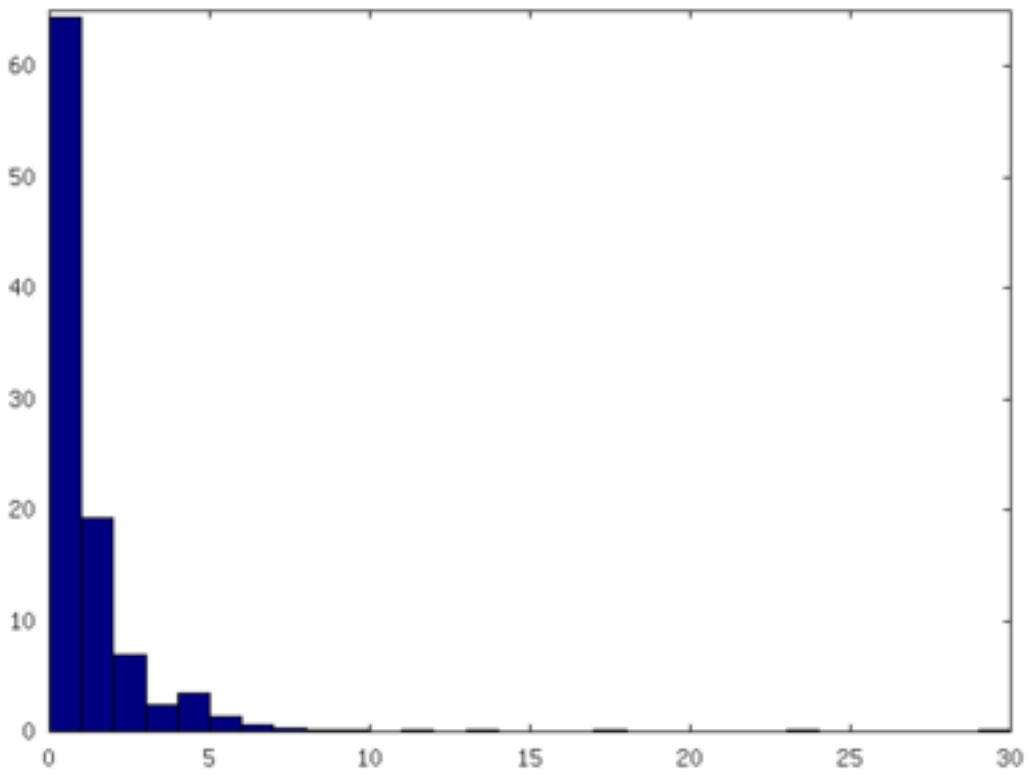
Sometimes rTorrent loses track of where it stores the data for an item, leading to an empty `Base path` in the `Info` panel. You can try to fix this by selectively rehashing those, with these commands:

```
rtcontrol path= is_complete=y -V
rtcontrol path= is_complete=y --hash -i
```

The first command selects the broken items into a rTorrent view, so that you can watch the progress of hashing and the results afterwards. If all of them are finished, you can then start those that were successfully restored like so:

```
rtcontrol path=\! done=100 --from-view rtcontrol --start``
```

(note that the `--from-view` option needs version 0.3.7)



Deleting Download Items and Their Data

Using the option `--cull` of version 0.3.10, an item can be deleted including its data. You can do this either manually, or automatically as a part of ratio management (see the section further below on that topic).

Called from the shell, you will first be presented with the number of items found and then asked for each of them whether you want to delete it (interactive mode is on by default). Therefor, for automatic uses in cron, you should also specify the `--yes` option.

If you define the following command shortcut, you can also delete the current item directly from ncurses (needs version 0.4.1 to work):

```
method.insert = cull,simple|private,"execute.nothrow=rtcontrol,-q,--detach,--cull,--
↪yes,\"$cat=hash=,$d.hash=\""
```

Just select the item you want to annihilate and enter `cull=` into the command prompt (`Ctrl-X`). Note that *you already have that command added* if you followed the *Configuration Guide*.

Pruning Partial Downloads

Starting with version 0.3.10, the `--purge` option (a/k/a `--delete-partial`) allows you to not only delete the selected items from the client, but at the same time delete any incomplete files contained in them (i.e. files that are part of an incomplete chunk).

For technical reasons, rTorrent has to create files that you have deselected from download to save data of chunks that border selected files, and this option can be a great time saver, especially on large torrents containing hundreds of files. So, unless you have filtered out incomplete items by the appropriate conditions, using `--purge` instead of `--delete` is always the better option.

As with `--cull`, a shortcut command to call this from the curses UI is useful:

```
system.method.insert = purge,simple,"execute_nothrow=rtcontrol,-q,--detach,--purge,--
↪yes,\"$cat=hash=,$d.get_hash=\""
```

Note that *you already have that command added* if you followed the *Configuration Guide*.

Performing Periodic Tasks

Simple Queue Management

This is a queue management one-liner (well, logically one line). Before you run it automatically, add a trailing "-n" to test it out, e.g. play with the queue size parameter and check out what would be started. Then put it into a script, crontab that and run it every (few) minute(s).

```
export rt_max_start=6; rtcontrol -q --start --yes hash=$(echo $( \
    rtcontrol -qrs is_active -o is_open,hash is_complete=no is_ignored=no \
    | head -n $rt_max_start | grep ^CLS | cut -f2 ) | tr " " ,)
```

It works by listing all incomplete downloads that heed commands and sorting the already active ones to the top. Then it looks at the first `rt_max_start` entries and starts any closed ones.

Note that this means you can exempt items from queue management easily by using the `I` key in the curses interface. See *rTorrent Queue Manager* for a much better solution.

Move on Completion

The following moves completed downloads *still physically residing* in a `work` directory (change the `realpath` filter when you named your download directory differently), to another directory (note that you can restrict this further, e.g. to a specific tracker by using "alias=NAME"). You don't need any multiple watch folders or other prerequisites for this.

```
rtcontrol --from-view complete 'realpath=*/work/*' -qo '~/bin/rtmv "$(path)s" ~/
↪rtorrent/done --cron' | bash
```

Test it first **without the** `| bash` **part** at the end, to make sure it'll in fact do what you intended.

Another advantage is that in case you ever wanted to switch clients, or exchange the drive you host the data on, you can do so easily since all the active downloads still reside at one place in your download directory (in form of a bunch of symlinks) — even if their data is scattered all over the place in reality.

You can also extend it to create more organized completion structures, e.g. creating a directory tree organized by month and item type, as follows:

```
RT_SOCKET=/home/bt/rtorrent/.scgi_local

# Move completed torrents to "done", organized by month and item type (e.g. "2010-09/
↪tv/avi")
*/15    * * * *         test -S $RT_SOCKET && ~/bin/rtcontrol --from-view complete
↪'realpath=*/work/*' -qo '~/bin/rtmv "$(path)s" ~/rtorrent/done//$(now.iso).7s/
↪$(traits)s --cron' | bash
```

The above is a fully working crontab example, you just have to adapt the paths to your system. If you want to create other organizational hierarchies, like "by tracker", just replace the `$(now.iso).7s/$(traits)s` part by `$(alias)s`. And if you don't want the file type in there (i.e. just "tv"), use `$(traits.pathdir)s` to have it removed.

To get themed trackers specially treated, you can add hints to the `[TRAITS_BY_ALIAS]` section of the config (see `config.ini.default` for examples).

Afterwards, you can always move and rename stuff at will *and still continue seeding*, by using the `rtmv` tool in version 0.3.7 — this will rename the data file or directory at its current location and automatically fix the symlink in the download directory to point at the new path. Example:

```
cd ~/rtorrent/done/2010-09/tv/avi
rtmv foo.avi bar.avi
```

Ratio Management

While rTorrent has a built-in form of ratio management since a few versions, it's hard to use after-the-fact and also hard to understand — you need to have different watch directories and complex settings in your `.rtorrent.rc` to use that.

It can be much simpler — a basic form of ratio management using `rtcontrol` looks like this:

```
rtcontrol is_complete=yes is_open=yes ratio=+1.1 alias=sometracker,othertracker --stop
```

You will always want to have the `is_complete=yes is_open=yes ratio=+1.1` part, which excludes all torrents that are still downloading, closed or not having the necessary ratio. Another basic filter is `is_ignored=no`, which excludes items that have their *ignore commands* flag set (via the `I` key) from ratio management.

To that you can add anything you think fits your needs, and also use several commands with different minimum ratios for different trackers by selecting them using `alias` or `tracker`, like in the example above. Assuming you have your original seeds in a directory named `seed` and don't want to ratio-limit them, one thing you might add is `'datapath=!*/seed/*'` to prevent them from being stopped. Only your imagination (and the available fields) are the limit here.

If you then put these commands into a script that runs every few minutes via `cron`, you have a very flexible form of ratio management that can be changed on a whim.

Note: For cron use, you'll want to add the `--cron --yes` options to any `rtcontrol` commands. The first one redirects logging to a special logfile `~/.pyroscope/log/cron.log`, and the second positively answers any prompts that would appear when using `--delete` or `--cull`.

To complete your command line, you add the action you want to take on the torrents found, in the above example `--stop`; `--delete` is another possibility, which removes the item from the client, but leaves the data intact. Starting with version 0.3.10, you can also delete the downloaded data by using the `--cull` option.

Bandwidth Management

Say you want to have torrents that are already seeded back take a back-seat when other torrents with a ratio less than 100% are active — but when they're not, all torrents should take full advantage of the available bandwidth. The last part is not possible with the built-in throttle groups, but here's a fix that works by setting the maximum rate on the `seed` throttle dynamically.

Put this into your `.rtorrent.rc`:

```
throttle_up=seed,900
```

Then save the dynamic seed throttle script into `~/bin/rt_cron_throttle_seed`.

Finally, extend your crontab with these lines (`crontab -e`):

```
RT_SOCKET=/home/bt/rtorrent/.scgi_local
BW_SEED_MAX=900
BW_SEED_SLOW=200

# Throttle torrents that are seeded 1:1 when there are other active ones
*   * * * *         test -S $RT_SOCKET && ~/bin/rt_cron_throttle_seed seed $BW_SEED_
↪MAX $BW_SEED_SLOW --cron

# Put torrents seeded above 1:1 into the seed throttle
*/10        * * * *         test -S $RT_SOCKET && rtcontrol ratio=+1.05 is_complete=1␣
↪is_ignored=0 throttle=none -q -T seed --yes --cron
```

The `900` and `200` in the above examples are the bandwidth limits in KiB/s, you need to adapt them to your connection of course, and all paths need to be changed to fit your system. Each time the throttle rate is changed, a line like the following will be appended to the file `~/.pyroscope/log/cron.log`:

```
2010-08-30 14:16:01 INFO     THROTTLE 'seed' up=200.0 KiB/s [2 prioritized] [__main__.
↪SeedThrottle]
```

Automatic Stop of Items Having Problems

This job takes away a lot of manual monitoring work you had to do previously:

```
HOME=/home/rtorrent
RT_SOCKET=/var/torrent/.scgi_local

# Stops any torrent that isn't known by the tracker anymore,
# or has other authorization problems, or lost its data
* * * * *   test -S $RT_SOCKET && sleep 21 && nice ~/bin/_cron_rt_invalid_items --
↪stop --cron
```

Just call `crontab -e` as the `rtorrent` user and add the above lines. You also need to install the _cron_rt_invalid_items script into `~/bin`.

The `prio=-3` in the script's list of conditions enables you to keep items running in case of errors, by setting their priority to `high`, e.g. when only some trackers in a longer list return errors. The `is_complete=yes is_ghost=yes` part means you can simply stop torrents by removing their data, it won't take more than a minute for the related item to be force-stopped.

Using Output Templates

Introduction

One of the output formatting options described in the *'rtcontrol' Examples* section are Tempita templates. Compared to the other options, they offer more versatile formatting because you can use conditionals and loops, e.g. coloring the output based on some value thresholds (see the example below). A full description of the Tempita language can be found in its documentation.

Note that in order for them to be recognized as such, Tempita templates **MUST** start with two braces `{{`, use `{{#}}` (an empty template comment) if you want to start the output with some literal text.

Using Tempita to format single items

The most common form of using Tempita for formatting a single output item of a `rtcontrol` result is probably by defining it in the configuration as a custom format, so it can be simply used by its name.

The `colored` predefined format is a typical example:

```
[FORMATS]

colored     = {{default ESC = '\x1B'}}{{d.size|sz}} {{d.uploaded|sz}} {{#
    }}{{if d.seedtime < 8*7*86400}}{{ESC}}[36m{{d.seedtime|duration}}{{ESC}}[0m{{else}
↪}}{{d.seedtime|duration}}{{endif}}{{#
    }}{{if d.ratio < 0.8}}{{ESC}}[1m{{ESC}}[31m{{elif d.ratio < 1.0}}{{ESC}}[36m{
↪{elif type(d.ratio) is float}}{{ESC}}[32m{{endif}}{{#
    }} {{str(pc(d.ratio)).rjust(8)}}{{chr(37)}}{{if type(d.ratio) is float}}{{ESC}}[0m
↪{{endif}}{{#
    }} {{(d.alias or '').ljust(8)}} {{d.name or ''}}
```

The main reason to use Tempita here are the if conditions that color the output depending on threshold values, for the ratio and seed time columns. Additionally to what Tempita provides, the global namespace of the template contains the usual format specifiers (see the output of the `--help-fields` option), and the current result item as `d` (think **d**ownload item).

If you look at some of the if conditions, you might find them peculiar, especially the `{{if type(d.ratio) is float}}` one. This is so that the column headers, which are obviously not the usual float values but strings, are exempt from any special coloring. Similarly, the `{{d.name or ''}}` caters for the fact that when you use the `rtcontrol --summary` option, fields that could normally never be `None` suddenly are — because what's the average of a string, really?

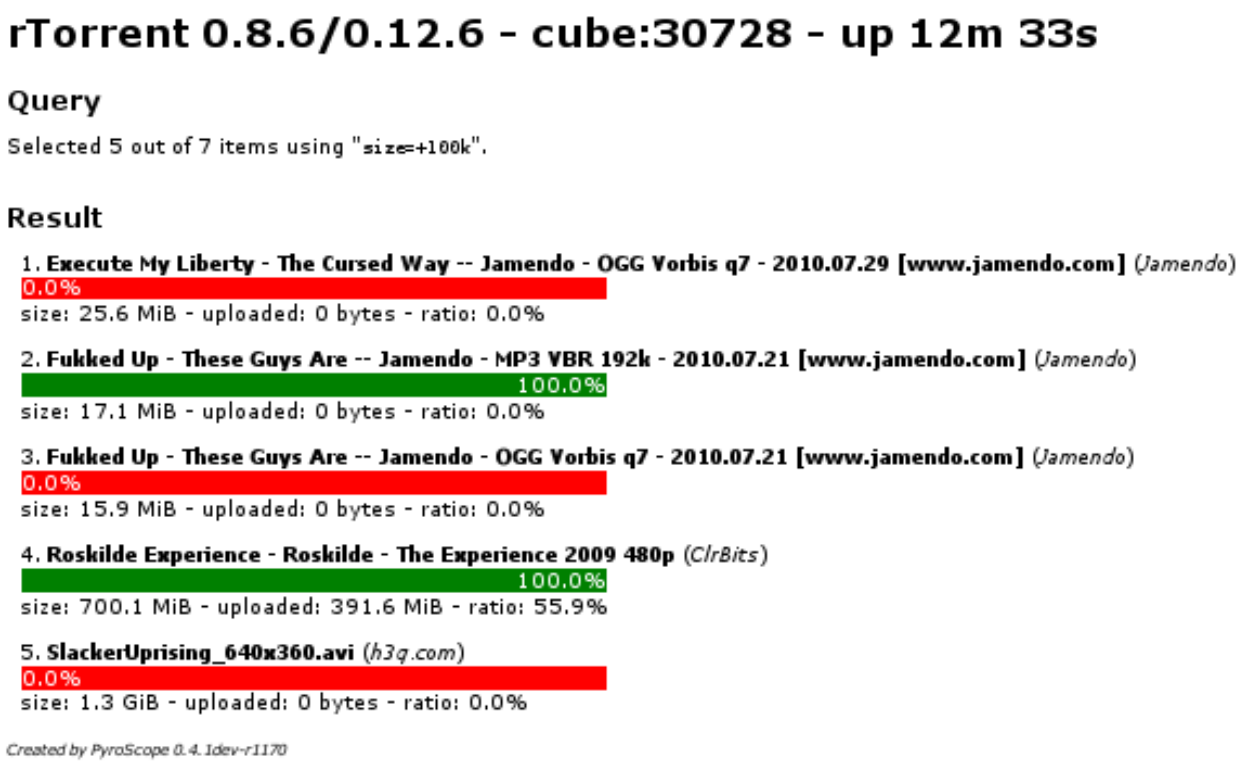
Notable here is also the use of a named default value `ESC`, and using template comments `{{#}}` to escape the line endings we don't want to have in the final output, which looks like this:

SIZE	UPLOADED	SEEDTIME	RATIO%
723.8 MiB	0 bytes	15w 3d	0.0%
401.0 MiB	7.2 MiB	15w 3d	1.7%
282.5 MiB	29.5 MiB	4w 1d	10.4%
558.6 MiB	76.3 MiB	7w 4d	13.6%
348.8 MiB	90.8 MiB	15w 3d	26.0%
729.1 MiB	723.2 MiB	7w 4d	99.2%
254.6 MiB	442.4 MiB	15w 3d	173.7%
362.7 MiB	656.0 MiB	15w 3d	180.8%

Using Tempita for full output control

If you use the `--output-template` option of `rtcontrol`, flow control of presenting the query result is passed fully to a Tempita template. That means that in addition to iterating over the query result, you can also show any value available via the rTorrent XMLRPC connection, since the proxy object that allows access to the client is passed to the template.

This example shows the output of such a template that resembles the rtorstat output:



rTorrent 0.8.6/0.12.6 - cube:30728 - up 12m 33s

Query

Selected 5 out of 7 items using "size=+100k".

Result

1. **Execute My Liberty - The Cursed Way -- Jamendo - OGG Vorbis q7 - 2010.07.29 [www.jamendo.com]** *(Jamendo)*
0.0%
size: 25.6 MiB - uploaded: 0 bytes - ratio: 0.0%

2. **Fukked Up - These Guys Are -- Jamendo - MP3 VBR 192k - 2010.07.21 [www.jamendo.com]** *(Jamendo)*
100.0%
size: 17.1 MiB - uploaded: 0 bytes - ratio: 0.0%

3. **Fukked Up - These Guys Are -- Jamendo - OGG Vorbis q7 - 2010.07.21 [www.jamendo.com]** *(Jamendo)*
0.0%
size: 15.9 MiB - uploaded: 0 bytes - ratio: 0.0%

4. **Roskilde Experience - Roskilde - The Experience 2009 480p** *(ClrBits)*
100.0%
size: 700.1 MiB - uploaded: 391.6 MiB - ratio: 55.9%

5. **SlackerUprising_640x360.avi** *(h3q.com)*
0.0%
size: 1.3 GiB - uploaded: 0 bytes - ratio: 0.0%

Created by PyroScope 0.4.1dev-r1170

To generate a similar result with your installation, follow these steps after updating it:

1. Call `pyroadmin --create-config` to copy the builtin `rtorstat.html` template to your configuration.
2. Call `rtcontrol -qO rtorstat.html done=-100 OR xfer=+0 -sdone >/var/www/cron/rtorrent.html` to create a HTML page.
3. Open that page in your browser.

You can add the command from step #2 as a cronjob and always have a current status display; instead of copying to the local web server space, you could also put the output into your Dropbox folder to have a status display on your mobile gear.

The namespace of these templates is populated with the following objects:

- `version` = the version of PyroScope
- `proxy` = the client proxy (you can call any XMLRPC method on that)
- `view` = the view that was queried
- `query` = the query conditions
- `matches` = the query result (a list of RtorrentItem objects)

Running a rtorstat-like template as a cgi-bin

To get the output of the above example template on-demand, which likely puts less stress on the system and also gives you current information, you can add a cgi-bin wrapper to your webserver. We assume a Debian or Ubuntu Apache standard installation here, and put the cgi-bin into the file `/usr/lib/cgi-bin/rtorstat` with the following content:

```
#! /bin/bash
echo "Content-Type: text/html; charset=UTF-8"
echo

export HOME=/home/bt
$HOME/bin/rtcontrol -qO rtorstat.html done=-100 OR xfer=+0 -sdone
```

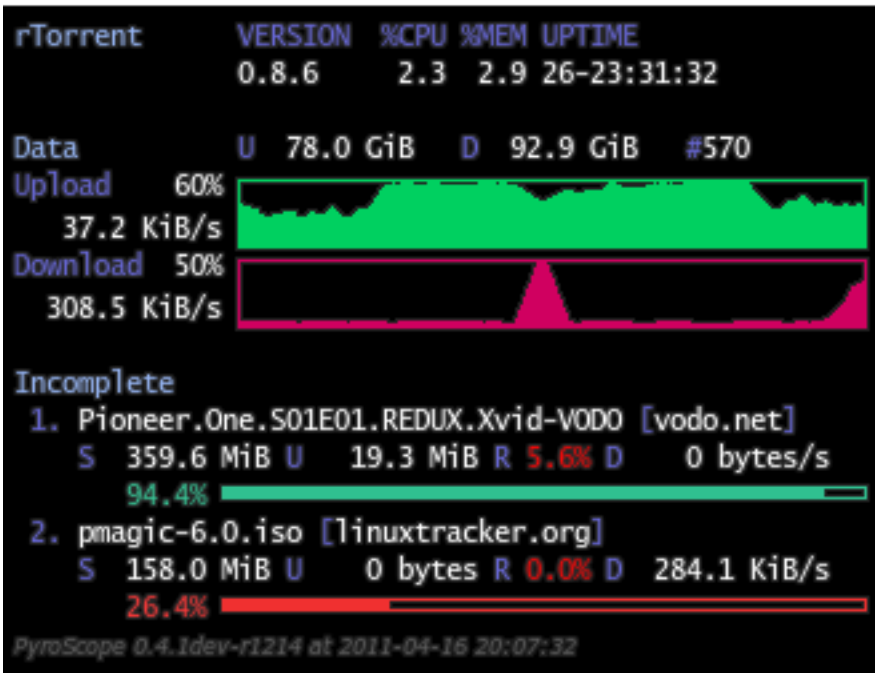
This will only work if permissions are given to the webserver user (normally `www-data`) to access the configuration files belonging to the `bt` user. In case you use a `scgi_local` connection (i.e. a UNIX domain socket), this also applies the the XMLRPC socket file.

That can be done by making all things group-readable, and add `www-data` to the `bt` group. Also, the socket file must be group-writeable when you use one (TCP sockets are available to all users on the machine anyway).

Finally, you can put a `<meta http-equiv="refresh" content="60">` into the template to automatically refresh the page every minute.

Adding a rTorrent status display to `conky`

You can add a status display to the well-known `conky` system monitor tool by using the conky rtorstat template together with a matching conkyrc:



To display the example, run these commands, assuming you have conky already installed:

1. `pyroadmin --create-config`
2. `conky -c ~/.pyroscope/templates/conky/conkyrc`

If you change the `execpi` in the conky configuration to call a remotely installed `rtcontrol` via `ssh`, you can also beam the status of a remote rTorrent instance onto your desktop. It is advisable to increase the poll intervall to at least 15 seconds in that case. Note that this setup means you have the `.conkyrc` on your local host, but the template used is on the remote host!

```
...
${execp ssh -o ConnectTimeout=15 -o SetupTimeOut=15 -T REMOTEHOST "~/bin/rtcontrol -
↪qO conky/rtorstat.txt --from-view incomplete is_open=yes is_ignored=no"}
```

Change `REMOTEHOST` to the name of the remote host, and make sure you have public key login enabled.

Listing all orphans in your download directory

This example shows how easily you can use templates to extract some information out of the client that is otherwise not directly available. The orphans.txt template lists all paths in the download directory *not* loaded into the client, and can be called like this:

```
rtcontrol -qO orphans.txt \*
```

Tips & How-Tos

Adding Category Views to the rTorrent UI

Version `0.5.1` enables you to easily add category views, that also play nice with *ruTorrent* labels in `custom_1`. Since this relies on key bindings, it only works using *rTorrent-PS*.

First, you need to define your category names and watches, like in this example:

```
cd ~/rtorrent
~/.local/pyroscope/src/scripts/add-categories.sh books hdtv movies
```

It is recommended to stick to alphanumeric category names, and use _ for word separation.

The watches put loaded items into the given category, and they expect metafiles in `~/rtorrent/watch/‹category-name›`.

To remove a category, just edit it out of the `rtorrent.d/categories.rc` file, and then call the `add-categories.sh` script without any arguments to clean things up.

On an existing installation, to auto-create categories for all the *ruTorrent* labels you already have (and that also fit the *alphanumeric* constraint), call this:

```
cd ~/rtorrent
~/.local/pyroscope/src/scripts/add-categories.sh \
    $(rtcontrol custom_1=\! -qo custom_1 | egrep '^[_a-zA-Z0-9]+$' | sort -u)
```

Note: After these configuration changes, don't forget to restart *rTorrent*.

In the *rTorrent-PS* user interface, you can now work with the following keys:

- Rotate through category views using < and >.
- The | key updates the current category view, i.e. filters for new or removed items.

The sort order of these views is the same as `main`, and if you switch to any other view and back to categories, you always start at the first category view (from the sorted list of category names).

Dumping Items as a JSON Array

If you want to access rTorrent item data in machine readable form via `rtcontrol`, you can use its `--json` option and feed the output into another script parsing the JSON data for further processing.

Here's an example:

```
$ rtcontrol --json -qo name,is_ghost,directory,fno foo
[
  {
    "directory": "/var/torrent/load/foo",
    "fno": 1,
    "is_ghost": false,
    "name": "foo"
  }
]
```

Note: When using `--json`, the list of fields given with `-o` must consist only of plain field names, i.e. format specifiers aren't supported. If you need derived values, the process parsing the output needs to calculate them.

Working With Several rTorrent Instances

Switching to the 'rtorrent.rc' of an Instance

Both `rtcontrol` and `rtxmlrpc` read the existing rTorrent configuration to extract some settings, so that you don't need to maintain them twice – most importantly the details of the XMLRPC connection. That is why `config.ini` has the `rtorrent_rc` setting, and changing that is the key to select a different instance you have running.

Just pass the option `-D rtorrent_rc=PATH_TO/rtorrent.rc` to either `rtcontrol` or `rtxmlrpc`, to read the configuration of another instance than the default one. For convenient use on the command line, you can add shell aliases to you profile.

Customizing the Default Configuration per Instance

Since version `0.5.1`, the extensions to the rTorrent configuration are loaded via the commands in `~/.pyroscope/rtorrent-pyro.rc.default`, importing snippets found in the `~/.pyroscope/rtorrent.d/` directory. The `commands.rc.default` file located there contains commands that use `rtcontrol` behind the scenes.

As shown in the previous section, these commands must use `-D` to load the right configuration. Instead of switching to importing the `*.rc` variants wholesale, with all the work that comes with that after updates, you can simply ignore just the `commands.rc.default` file, and replace it with an adapted copy in your *main* configuration file.

So, in summary, to customize a `~/rtorrent1` instance:

```
echo >>~/.pyroscope/rtorrent.d/.rcignore "commands.rc.default"
sed -r -e 's:--detach:--detach,-D,"rtorrent_rc=~/rtorrent1/rtorrent.rc":' \
    ~/.pyroscope/rtorrent.d/commands.rc.default \
    >>~/rtorrent1/rtorrent.rc
```

Now commands like `s=` are defined in `~/rtorrent1/rtorrent.rc`, and `commands.rc.default` is not imported, so no duplicate definition errors occur.

Moving All Data for Selected Items to a New Location

This shows how to move the *data* of all items for a specific tracker (identified by the alias `TRK`) from `~/rtorrent/data/` to `~/rtorrent/data/tracker/`. Note that you can do that in *ruTorrent* too, but with too many items, or items too big, the results vary (data is not or only partially moved).

This sequence of commands will stop and relocate the loaded items, move their data, and finally start everything again.

```
mkdir -p ~/rtorrent/data/tracker
rtcontrol --to-view tagged alias=TRK realpath=$HOME/rtorrent/data
rtcontrol --from-view tagged // --stop
rtcontrol --from-view tagged // --exec "directory.set=$HOME/rtorrent/data/tracker" --
↪yes
rtcontrol --from-view tagged // --spawn "mv {{item.path}} $HOME/rtorrent/data/tracker"
rtcontrol --from-view tagged // --start
```

By changing the first `rtcontrol` command that populates the `tagged` view, you can change this to move data for any criteria you can think of — within the limits of `rtcontrol` *Filter Conditions*. Also, if you run *rTorrent-PS*, you can manually remove items from the `tagged` view by using the `.` key, before applying the rest of the commands.

Also see the *Advanced 'rtcontrol'* section that explains the `--spawn` and `--exec` options in more depth.

Note: The `tagged` view is used here solely for the purpose of allowing manual manipulation of the search result after step 1, when using *rTorrent-PS*. It is *not* related to the `tagged` *field* in any way.

They're just different ways to tag items, one of them visually in the *rTorrent-PS* UI.

Tag Episodes in rT-PS, Then Delete Their Whole Season

The command below allows you to delete all items that belong to the same season of a TV series, where single episodes were tagged as a stand-ins for that season. The tagging can be done interactively in rTorrent-PS, using the `.` key.

```
rtcontrol --from tagged -s* -qoname "/\\.S[0-9][0-9]E[0-9][0-9]\\./" \
    | sed -re 's/(.+\.[sS]..[eE])..\..+/\1/' | uniq | \
    | xargs -I# -d$'\n' rtcontrol '/^#/' --cull --yes -A dupes- loaded=+2w
```

The culling command call also protects any item younger than 2 weeks.

Using Tags or Flag Files to Control Item Processing

If you want to perform some actions on download items exactly once, you can use tags or flag files to mark them as handled. The basic pattern works like this:

```
#! /usr/bin/env bash
guard="handled"
...

rtcontrol --from-view complete -qohash tagged=\!$guard | \
while read hash; do
    ...

    # Mark item as handled
    rtcontrol -q --from-view $hash // --tag "$guard" --flush --yes --cron
done
```

The `--from-view $hash //` is an efficient way to select a specific item by hash, in case you wondered. `hash=‹infohash›` in contrast loads all items, then filters out just one.

A variant of this is to use a flag file in the download's directory – such a file can be created and checked by simply poking the file system, which can have advantages in some situations. To check for the existance of that file, add a custom field to your `config.py` as follows:

```
def is_synced(obj):
    "Check for .synced file."
    pathname = obj.path
    if pathname and os.path.isdir(pathname):
        return os.path.exists(os.path.join(pathname, '.synced'))
    else:
        return False if pathname else None

yield engine.DynamicField(engine.untyped, "is_synced", "does download have a .synced␣
↪flag file?",
    matcher=matching.BoolFilter, accessor=is_synced,
    formatter=lambda val: "SYNC" if val else "????" if val is None else "!SYN")
```

The condition `is_synced=no` is then used instead of the `tagged` one in the bash snippet above, and setting the flag is a simple `touch`. Add a `rsync` call to the `while` loop in the example and you have a cron job that can be used to transfer completed items to another host *exactly once*. Note that this only works for multi-file items, since a data directory is assumed – supporting single-file items is left as an exercise for the reader. See *Defining Custom Fields* for more details regarding custom fields.

Advanced Features

Note: Using these features requires some knowledge in the area Linux, Bash, and Python beyond a novice level, but they enable you to customize your setup even further and handle very specific use-cases.

Advanced 'rtcontrol'

Executing OS commands

The `--call` and `--spawn` options can be used to call an OS level command and feed it with data from the selected items. The argument to both options is a template, i.e. you can have things like `{{item.hash}}` in them.

When using `--call`, the command is passed to the shell for parsing – with obvious implications regarding the quoting of arguments, thus `--call` only makes sense if you need I/O redirection or similar shell features.

In contrast, the `--spawn` option splits its argument list according to shell rules *before* expanding the template placeholders, and then calls the resulting sequence of command name and arguments directly. Consider `--spawn 'echo "name: {{item.name}}"'` vs. `--spawn 'echo name: {{item.name}}'` – the first form passes one argument to `/bin/echo`, the second form two arguments. Note that in both cases, spaces or shell meta characters contained in the item name are of no relevance, since the argument list is split according to the template, *not* its expanded value.

To list all the fields available in the first five items, try this command:

```
rtcontrol // -/5 --spawn "echo -e '\\n'{{item}}" | sed -re 's/, /,\n    /g'
```

Unlike `--call`, where you can use shell syntax to call several commands, `--spawn` can be passed several times for executing a sequence of commands. If any called command fails, the `rtcontrol` call is aborted with an error.

Copy Session Metafiles by Category

Here's a practical example for using `--spawn`, it copies all your loaded metafiles from the session directory into a folder structure categorized by the *ruTorrent* label. Unlabelled items go to the `_NOLABEL` folder.

```
target="/tmp/metafiles"
rm -rf "$target"
rtcontrol // \
    --spawn "mkdir -p \"$target/"'{{item.fetch(1) or \"_NOLABEL\"}}"' \
    --spawn 'cp {{item.sessionfile}} "'"$target"'/{{item.fetch(1) or \"_NOLABEL\"}}/{
↪{item.name}}-{{item.hash[:7]}}.torrent"'
```

The copied metafiles themselves are renamed to the contained name of the item's data, plus a small part of the infohash to make these names unique.

Replace the `item.fetch(1)` by `item.‹fieldname›` to categorize by other values, e.g. `item.alias` for 'by tracker'.

Executing XMLRPC commands

If you want to apply some custom XMLRPC commands against a set of download items, the `--exec` option of `rtcontrol` allows you to do that. For global commands not referring to specific items, see the next section about the `rtxmlrpc` tool. Read through the following examples to understand how `--exec` works, features are explained as they are used there. Also make sure you understand basic things like *Using Output Templates* beforehand, it's assumed here that you do.

Examples for using `--exec`

- *Repairing Stuck Items*
- *Relocating Download Data*
- *Making Shared Data Paths Unique*
- *Changing Announce URLs in Bulk*

Note: Previously, the common way to handle use-cases covered by `--exec` was to pipe `rtxmlrpc` commands generated via templating into `bash`. Don't do that anymore, it's quite inferior to using `--exec`.

Repairing Stuck Items

Let's start with an easy example of using `--exec`, where no templating is needed:

```
rtcontrol --exec 'stop= ; close= ; f.multicall=,f.set_create_queued=0,f.set_resize_
↪queued=0 ; check_hash=' \
          --from stopped -/1
```

This command simulates pressing `^K^E^R` in the curses UI (which cleans the state of stuck / damaged items) and only affects the first stopped item. Use different filter arguments after `--exec` to select other items. Afterwards, use `--start` to start these items again.

Relocating Download Data

The most simple variant of changing the download path is setting a new fixed location for all selected items, as follows:

```
rtcontrol --exec 'directory.set=/mnt/data/new/path' directory=/mnt/data/old/path
```

This replaces the location of items stored at `/mnt/data/old/path` with a new path. But to be really useful, we'd want to shift *any* path under a given base directory to a new location – the next command does this by using templating and calculating the new path based on the old one:

```
rtcontrol \
    --exec 'directory.set={{item.directory|subst("^/mnt/data/","/var/data/")}} ; >
↪directory=' \
    directory=/mnt/data/\*
```

This selects any item stored under `/mnt/data` and relocates it to the new base directory `/var/data`. Fields of an item can be used via a `item.‹field-name›` reference. Adding `>directory=` prints the new location to the console – a semicolon with spaces on both sides delimits several commands, and the `>` prints the result of a XMLRPC command. Also note that the `d.` prefix to download item commands is implied.

Making Shared Data Paths Unique

Another example regarding data paths is this:

```
rtcontrol --from stopped // --anneal dupes= --exec 'directory.set={{item.directory}}-{
↪{item.hash}}'
```

That command ensures that items that would download into the same path get a unique name by appending the info hash, and assumes those items weren't started yet (i.e. added via `load.normal`).

Changing Announce URLs in Bulk

The next example replaces an active announce URL with a new one, which is necessary after a domain or passkey change. Compared to other methods like using `sed` on the files in your session directory, this does not require a client restart, and is also safer (the `sed` approach can easily make your session files unusable). This disables all old announce URLs in group 0 using a `t.multicall`, and then adds a new one:

```
rtcontrol \
    --exec 't.multicall=0,t.disable= ; tracker.insert=0,"http://new.example.com/
↪announce" ; save_full_session=' \
    "tracker=http://old.example.com/announce"
```

The `tracker.insert` also shows that arguments to commands can be quoted.

Using 'rtxmlrpc'

Querying system information

The `rtuptime` script shows you essential information about your *rTorrent* instance:

```
#! /bin/bash
SCGI_SOCKET=~/rtorrent/.scgi_local

if test ! -S $SCGI_SOCKET; then
    echo >&2 "rTorrent is not running (no socket $SCGI_SOCKET)"
    exit 1
fi

echo -n rTorrent $(rtxmlrpc system.client_version)/$(rtxmlrpc system.library_version)
echo -n , up $(rtxmlrpc to_elapsed_time $(ls -l --time-style '+%s' $SCGI_SOCKET | awk
↪'{print $6}'))
echo -n \ [$(rtcontrol -qo"1 %(uploaded)s %(size)s" \* | \
    awk '{ TOT += $1; UP += $2; SUM += $3} END { print TOT " loaded; U: " UP/1024/
↪1024/1024 " GiB; S: " SUM/1024/1024/1024 }') GiB]
echo -n , D: $(rtxmlrpc to_xb $(rtxmlrpc throttle.global_down.total))
echo -n \ @ $(rtxmlrpc to_xb $(rtxmlrpc throttle.global_down.rate))/s
echo -n \ of $(rtxmlrpc to_xb $(rtxmlrpc throttle.global_down.max_rate))/s
echo -n , U: $(rtxmlrpc to_xb $(rtxmlrpc throttle.global_up.total))
echo -n \ @ $(rtxmlrpc to_xb $(rtxmlrpc throttle.global_up.rate))/s
echo -n \ of $(rtxmlrpc to_xb $(rtxmlrpc throttle.global_up.max_rate))/s
echo
```

When called, it prints something like this:

```
$ rtuptime
rTorrent 0.9.6/0.13.6, up 189:00:28 [315 loaded; U: 177.292 GiB; S: 891.781 GiB],
D: 27.3 GB @ 0.0 KB/s of 520.0 KB/s, U: 36.8 MB @ 0.0 KB/s of 52.0 KB/s
```

And yes, doing the same in a *Python script* would be much more CPU efficient. ;)

If you connect via `network.scgi.open_port`, touch a file in `/tmp` in your startup script and use that for uptime checking.

General maintenance tasks

Here are some commands that can help with managing your rTorrent instance:

```
# Flush ALL session data NOW, use this before you make a backup of your session␣
↪directory
rtxmlrpc session.save
```

Setting and checking throttles

To set the speed of the `slow` throttle, and then check your new limit and print the current download rate, use:

```
rtxmlrpc throttle.down '' slow 120
# 0
rtxmlrpc throttle.down.max '' slow
# 122880
rtxmlrpc throttle.down.rate '' slow
# 0
```

Note that the speed is specified in KiB/s as a string when setting it but returned in bytes/s as an integer on queries.

The following script makes this available in an easy usable form, e.g. `throttle slow 42` – it also shows the current rate and settings of all defined throttles when called without arguments:

```
#! /bin/bash
# Set speed of named throttle

#
# CONFIGURATION
#
throttle_name="seed" # default name
unit=1024 # KiB/s

#
# HERE BE DRAGONS!
#
down=false
if test "$1" = "-d"; then
    down=true
    shift
fi

if test -n "$(echo $1 | tr -d 0-9)"; then
    # Non-numeric $1 is a name
    throttle_name=$1
    shift
fi

if test -z "$1"; then
    echo >&2 "Usage: ${0/$HOME/~} [-d] [<throttle-name=$throttle_name>] <rate>"

    rtorrent_rc=~/.rtorrent.rc
    test -e "$rtorrent_rc" || rtorrent_rc="$(rtxmlrpc system.get_cwd)/rtorrent.rc"
    if test -e "$rtorrent_rc"; then
        throttles="$(egrep '^throttle[._](up|down)' $rtorrent_rc | tr ._=, ' ' | cut -
↪f3 -d" " | sort | uniq)"
        echo
        echo "CURRENT THROTTLE SETTINGS"
        for throttle in $throttles; do
            echo -e "  $throttle\t" \
                "U: $(rtxmlrpc to_kb $(rtxmlrpc throttle.up.rate $throttle)) /" \
                "$(rtxmlrpc to_kb $(rtxmlrpc throttle.up.max $throttle | sed 's/^-1$/
↪0/')) KiB/s\t" \
                "D: $(rtxmlrpc to_kb $(rtxmlrpc throttle.down.rate $throttle)) /" \
                "$(rtxmlrpc to_kb $(rtxmlrpc throttle.down.max $throttle | sed 's/^-1
↪$/0/')) KiB/s"
        done
    fi
    exit 2
fi

rate=$(( $1 * $unit ))

# Set chosen bandwidth
if $down; then
    if test $(rtxmlrpc throttle.down.max $throttle_name) -ne $rate; then
        rtxmlrpc -q throttle.down $throttle_name $(( $rate / 1024 ))
        echo "Throttle '$throttle_name' download rate changed to" \
             "$(( $(rtxmlrpc throttle.down.max $throttle_name) / 1024 )) KiB/s"
    fi
else
    if test $(rtxmlrpc throttle.up.max $throttle_name) -ne $rate; then
```

```
        rtxmlrpc -q throttle.up $throttle_name $(( $rate / 1024 ))
        echo "Throttle '$throttle_name' upload rate changed to" \
             "$(( $(rtxmlrpc throttle.up.max $throttle_name) / 1024 )) KiB/s"
    fi
fi
```

Global throttling when other computers are up

If you want to be loved by your house-mates, try this:

```
#! /bin/bash
# Throttle bittorrent when certain hosts are up

#
# CONFIGURATION
#
hosts_to_check="${1:-mom dad}"
full_up=62
full_down=620
nice_up=42
nice_down=123
unit=1024 # KiB/s

#
# HERE BE DRAGONS!
#

# Check if any prioritized hosts are up
up=$(( $full_up * $unit ))
down=$(( $full_down * $unit ))
hosts=""

for host in $hosts_to_check; do
    if ping -c1 $host >/dev/null 2>&1; then
        up=$(( $nice_up * $unit ))
        down=$(( $nice_down * $unit ))
        hosts="$hosts $host"
    fi
done

reason="at full throttle"
test -z "$hosts" || reason="for$hosts"

# Set chosen bandwidth
if test $(rtxmlrpc throttle.global_up.max_rate) -ne $up; then
    echo "Setting upload rate to $(( $up / 1024 )) KiB/s $reason"
    rtxmlrpc -q throttle.global_up.max_rate.set_kb $(( $up / 1024 ))
fi
if test $(rtxmlrpc throttle.global_down.max_rate) -ne $down; then
    echo "Setting download rate to $(( $down / 1024 )) KiB/s $reason"
    rtxmlrpc -q throttle.global_down.max_rate.set_kb $(( $down / 1024 ))
fi
```

Add it to your crontab and run it every few minutes.

Throttling rTorrent for a limited time

If you want to slow down *rTorrent* to use your available bandwidth on foreground tasks like browsing, but usually forget to return the throttle settings back to normal, then you can use the provided rt-backseat script. It will register a job via `at`, so that command must be installed on the machine for it to work. The default throttle speed and timeout can be set at the top of the script.

rTorrent Queue Manager

Introduction

The `pyrotorque` command is a daemon that handles background jobs. At first, it was just a flexible torrent queue manager for starting items one at a time (thus the name `pyro-tor-que`), but it can now manage any job that does some background processing for rTorrent, including custom ones that you can add yourself.

It runs in the background parallel to rTorrent and has its own scheduler to run automation jobs similar to rTorrent's `schedule` command — one of the jobs does start stopped items in a controlled fashion, that is the queue manager part.

Besides the queue manager, the most important job type is `TreeWatch`. It reacts to file system events (via `inotify`) to load new metafiles on the spot, if you add the necessary configuration and activate it. This way you have no delays at all, and no polling of watch directories in short intervals, most often with no tangible result and just wasted CPU cycles. Also, you can place the metafiles in arbitrary folders and sub-folders, with just one configuration entry for the root folder to watch. The queue is able to start items loaded via `inotify`, i.e. both jobs can work together.

If you want to know about the gory details of the machinery behind this, read *Writing Custom Jobs*.

Initial Setup

Before you start configuring the daemon, you have to install some additional Python dependencies it needs to do its work, also depending on what jobs you activate in your configuration. The following is how to install the *full* set of dependencies:

```
~/.local/pyroscope/bin/pip install -r ~/.local/pyroscope/requirements-torque.txt
```

Watch out for any errors, since this installs several Python extensions that *might* need some `*-dev` OS packages available that you don't have on your machine.

The `pyrotorque` queue manager daemon relies on certain additions to `rtorrent.rc`, these are included in the standard `pyrocore` includes that you added when you followed the *Configuration Guide*. If for whatever reason you need to add these manually, the file `~/.pyroscope/rtorrent.d/torque.rc.default` holds these settings.

The daemon itself is configured by an additional configuration file `~/.pyroscope/torque.ini` containing the `[TORQUE]` section. Most settings are already covered in `torque.ini.default`, including some short explanation what each setting does. The next section shows how to customize these defaults.

Configuration

Minimal Example

The following is a **minimal** `~/.pyroscope/torque.ini` **configuration example**, only changing a few values from the defaults to demonstrate key features:

```
# "pyrotorque" configuration file
#
# For details, see https://pyrocore.readthedocs.io/en/latest/advanced.html#torque-
↪config
#

[TORQUE]
# Queue manager
job.queue.active            = True
job.queue.schedule          = second=*/5
job.queue.intermission      = 60
job.queue.downloading_max   = 3
job.queue.startable         = is_ignored=0 message= prio>0
        [ prio>2 OR [ NOT [ traits=audio kind_25=jpg,png,tif,bmp ] ] ]
job.queue.downloading       = [ prio>1 [ down>3 OR started<2i ] ]

# Tree watch (works together with the queue)
job.treewatch.active        = True
job.treewatch.load_mode     = start
job.treewatch.queued        = True
job.treewatch.path          = /var/torrent/watch
job.treewatch.cmd.nfo       = f.multicall=*.nfo,f.priority.set=2
job.treewatch.cmd.jpg       = f.multicall=*.jpg,f.priority.set=2
job.treewatch.cmd.png       = f.multicall=*.png,f.priority.set=2
job.treewatch.cmd.tif       = f.multicall=*.tif,f.priority.set=0
job.treewatch.cmd.target    = {{# set target path
    }}d.custom.set=targetdir,/var/torrent/done/{{label}}/{{relpath}}
```

Having a minimal configuration with just your changes is recommended, so you get new defaults in later releases automatically.

See the default configuration for **more parameters and what they mean**.

Warning: If the folder tree specified in the `path` setting overlaps with the paths used in existing 'watch' schedules of `rtorrent.rc`, then please either keep those paths apart, or disable those schedules (comment them out), *before* activating tree watch.

Anything else will lead to confusing and inconsistent results.

Queue Settings Explained

In the above example for the `queue` job, `downloading_max` counts started-but-incomplete items including those that ignore commands. Only if there are fewer of these items in the client than that number, a new item will be started. This is the queue's length and thus the most important parameter.

The queue *never* stops any items, i.e. `downloading_max` is not enforced and you can manually start more items than that if you want to. That is also the reason items that should be under queue control must be loaded in 'normal' mode, i.e. stopped.

Other queue parameters are the minimum number of items in 'downloading' state named `downloading_min`, which trumps `start_at_once`, the maximum number of items to start in one run of the job. Both default to `1`. Since the default schedule is `second=*/15`, that means at most one item would be started every 15 seconds.

But that default is changed using the following two lines:

```
job.queue.schedule          = second=*/5
job.queue.intermission      = 60
```

This makes the queue manager check more often whether there is something startable, but prevents it from starting the next batch of items when the last start was less than `intermission` seconds ago.

The `startable` condition (repeated below for reference) prevents ignored items, ones having a non-empty message, and those with the lowest priority from being started. Note that tree watch sets the priority of items loaded in 'normal' mode to zero – that `prio>0` condition then excludes them from being started automatically some time later, until you press + to increase that priority. You can also delay not-yet-started items using the − key until the item has a priority of zero (a/k/a `off`).

```
job.queue.startable = is_ignored=0 message= prio>0
        [ prio>2 OR [ NOT [ traits=audio kind_25=jpg,png,tif,bmp ] ] ]
```

This sample condition also adds the extra hurdle that audio downloads that don't stay below a 25% threshold regarding contained images are **not** started automatically. *Unless* you raise the priority to 3 (`high`) using the + key, then they're fair game for the queue. Go do all that with a plain rTorrent watch dir, in one line of configuration.

The parameter `sort_fields` is used to determinate in what order startable items are handled. By default, higher priority items are started first, and age is used within each priority class.

Above, it was mentioned `downloading_max` counts started-but-incomplete items. The exact definition of that classification can be changed using the `downloading` condition. A given condition is *always* extended with `is_active=1 is_complete=0`, i.e. the started-but-incomplete requirement.

```
job.queue.downloading = [ prio>1 [ down>3 OR started<2i ] ]
```

In plain English, this example says we only count items that have a normal or high priority, and transfer data or were started in the last 2 minutes. The priority check means you can 'hide' started items from the queue by setting them to `low`, e.g. because they're awfully slow and prevent your full bandwidth from being used.

The second part automatically ignores stalled items unless just started. This prevents disk trashing when a big item is still creating its files and thus has no data transfer – it looks stalled, but we do not want yet another item to be started and increasing disk I/O even more, so the manager sees those idle but young items as occupying a slot in the queue.

Tree Watch Details

The `treewatch` job is set to co-operate with the queue as previously explained, and load items as ready to be started (i.e. in stopped state, but with normal priority). Any `load_mode` that is not either `start` or `started` is considered as equivalent to `load.normal`.

```
job.treewatch.active        = True
job.treewatch.load_mode     = start
job.treewatch.queued        = True
```

The configuration settings for `load_mode` and `queued` can also be changed on a case-by-case basis. For that, one of the 'flags' `load`, `start`, or `queued` has to appear in the path of the loaded metafile – either as a folder name, or else delimited by dots in the file name. These examples should help with understanding how to use that:

```
 load and start these, ignoring what 'load_mode' says
.../tv/start/foo.torrent
.../movies/foo.start.torrent

 just load these, ignoring what 'load_mode' says
.../tv/load/foo.torrent
```

```
.../movies/foo.load.torrent

 always queue these, using the configured 'load_mode'
.../tv/queue/foo.torrent
.../movies/foo.queue.torrent
```

Should you have both `start` and `load` in a path, then `start` wins.

`path` determines the root of the folder tree to watch for new metafiles via registration with the `inotify` mechanism of Linux. That means they are loaded milliseconds after they're written to disk, without any excessive polling.

```
job.treewatch.path          = /var/torrent/watch
```

You can provide more that one tree to watch, by separating the root folders with `:`.

The `cmd.`«`name`» settings can be used to provide additional load commands, executed during loading the new item, *before* it is started (in case it is started at all). This is equivalent to the commands you can append to a rTorrent `load.*` command. They're added in the alphabetic order of their names.

```
job.treewatch.cmd.nfo       = f.multicall=*.nfo,f.priority.set=2
job.treewatch.cmd.jpg       = f.multicall=*.jpg,f.priority.set=2
job.treewatch.cmd.png       = f.multicall=*.png,f.priority.set=2
job.treewatch.cmd.tif       = f.multicall=*.tif,f.priority.set=0
job.treewatch.cmd.target    = {{# set target path
    }}d.custom.set=targetdir,/var/torrent/done/{{label}}/{{relpath}}
```

The above example shows how to set any NFO files and JPG/PNG images to high priority, and prevent downloading any TIF images by default.

Commands can be templates, see *Using the Tree Watch Job* for further details on the `target` command.

Note: In case no files are loaded after you activated tree watch, you can set `trace_inotify` to `True` to get detailed logs of all file system events as they are received.

Also keep in mind that for now, if you add metafiles while the `pyrotorque` daemon is not running, you have to `touch` them manually after you have restarted it to load them.

Testing Your Configuration

After having completed your configuration, you're ready to **test it, by following these steps**:

1. Execute `rm ~/.pyroscope/run/pyrotorque` to **prevent the watchdog from starting the manager** in the background.
2. **Stop any running daemon** process using `pyrotorque --stop`, just in case.
3. Run `pyrotorque --fg -v` in a terminal, this will **start the job scheduler in the foreground** with verbose logging directly to that terminal, exactly what you need to check out if your configuration does what you intended. It also helps you to understand what goes on "under the hood".
4. If you applied **changes to your configuration**, stop the running scheduler by pressing CTRL-C, then **restart it**. Wash, rinse, repeat.
5. Press CTRL-C for the last time and call `pyrotorque --status`, it should show that no daemon process is running.

6. Execute `touch ~/.pyroscope/run/pyrotorque` — this does **create the guard file again**, which must always exist if you want `pyrotorque` to run in the background (otherwise you'll just get an error message on the console or in the log, if you try to launch it).
7. **Wait up to 300 seconds**, and if your *rTorrent* configuration has the `pyro_watchdog` schedule as it should have, `pyrotorque --status` will show that a daemon process was automatically started by that *rTorrent* schedule.
8. Enjoy, and **check** `~/.pyroscope/log/torque.log` for feedback from the daemon process.

If you want to restart the daemon running in the background immediately, e.g. to **reload** `torque.ini` or after a software update, use `pyrotorque --cron --restart`.

Built-in Jobs

The `QueueManager` is just one kind of job that can be run by `pyrotorque`. It has an embedded scheduler that can run any number of additional jobs, the following sections explain the built-in ones. Since these jobs can be loaded from any available Python package, you can also easily *write your own*.

Jobs and their configuration are added in the `[TORQUE]` section, by providing at least the parameters `job.«NAME».handler` and `job.«NAME».schedule`. Depending on the handler, additional parameters can/must be provided (see below for a list of built-in handlers and what they do).

Details on the `schedule` parameter can be found here. Multiple fields must be separated by spaces, so if a field value contains a space, it must be quoted, e.g. `hour=12 "day=3rd sun"`. The `handler` parameter tells the system where to look for the job implementation, see the handler descriptions below for the correct values.

QueueManager

`pyrocore.torrent.queue:QueueManager` manages queued downloads (i.e. starts them in a controlled manner), it is described in detail *further up on this page*.

TreeWatch (beta, not feature-complete)

`pyrocore.torrent.watch:TreeWatch` watches a folder tree, which can be nested arbitrarily. Loading of new `.torrent` files is immediate (using `libnotify`).

TODO Each sub-directory can contain a `watch.ini` configuration file for parameters like whether to start new items immediately, and for overriding the completion path.

See the explanation of the example configuration above and *Using the Tree Watch Job* for further details.

EngineStats

`pyrocore.torrent.jobs:EngineStats` runs once per minute, checks the connection to rTorrent, and logs some statistical information.

You can change it to run only hourly by adding this to the configuration: `job.connstats.schedule = hour=*`

Using the Tree Watch Job

Introduction

As mentioned in *rTorrent Queue Manager*, commands configured to be executed during item loading can be templates. This can be used to support all sorts of tricks, the most common ones are explained here, including fully dynamic completion moving. If the following explanation of the inner workings is too technical and nerdy for you, skip to the *Tree Watch Examples* section below, and just adapt one of the prepared use cases to your setup.

So how does this work? When a `.torrent` file is notified for loading via `inotify`, it's parsed and contained data is put into variables that can be used in the command templates. In order to get an idea what variables are available, you can dump the templating namespace for a metafile to the console, by calling the `watch` job directly.

Consider this example:

```
$ date >example.dat
$ mktor -q example.dat http://tracker.example.com/
$ python-pyrocore -m pyrocore.torrent.watch -v example.dat.torrent
...
DEBUG    Tree watcher created with config Bunch(active=False, ...
    cmd.target='{{# set target path\n}}d.custom.set=targetdir,/var/torrent/done/{
↪{label}}/{{relpath}}',
    dry_run=True, handler='pyrocore.torrent.watch:TreeWatch', job_name='treewatch',
    load_mode='start', path='/var/torrent', queued='True', quiet='False', schedule=
↪'hour=*')
DEBUG    custom commands = {'target': <Template 2d01990 name=None>, 'nfo': f.
↪multicall=*.nfo,f.set_priority=2, ...}
INFO     Templating values are:
    commands=[..., 'd.custom.set=targetdir,/var/torrent/done//pyrocore', ...]
    filetype='.dat'
    ...
    info_hash='8D59E3FD8E78CC9896BDE4D65B0DC9BDBA0ADC70'
    info_name='example.dat'
    label=''
    pathname='/var/torrent/pyroscope/example.dat.torrent'
    relpath='pyrocore'
    tracker_alias='tracker.example.com'
    traits=Bunch(kind=None)
    watch_path=set(['/var/torrent'])
```

Things to take note of:

1. the `target` custom command is expanded to set the `targetdir` rTorrent attribute to the completion path (which can then be used in a typical `event.download.finished` handler), using the `relpath` variable which is obtained from the full `.torrent` path, relative to the watch dir root.
2. all kinds of other information is made available, like the torrent's info hash and the tracker alias; thus you can write conditional templates based on tracker, or use the tracker name in a completion path.
3. for certain types of downloads, `traits` provides parsed information to build specific target paths, e.g. for the `Pioneer.One.S01E06.720p.x264-VODO` TV episode, you'll get this:

   ```
   label='tv/mkv'
   traits=Bunch(aspect=None, codec='x264', episode='06', extension=None, format='720p
   ↪',
       group='VODO', kind='tv', pattern='Normal TV Episodes', release=None,
       release_tags=None, season='01', show='Pioneer.One', sound=None, title=None)
   ```

Tree Watch Examples

- *Completion Moving*
- *Tree Watch with Sorting*

Completion Moving

Since the templating namespace automatically includes the path of a loaded `.torrent` file relative to the watch root (in `relpath`, see above example namespace output and the config example further down), you can set the "move on completion" target using that value.

```
job.treewatch.cmd.target    = {{# set target path
    }}d.custom.set=targetdir,/var/torrent/done/{{label}}/{{relpath}}
```

Note that this still needs a typical completion event handler that takes the custom variable that is set, and moves the data based on its value.

Tree Watch with Sorting

This example adds a *second* job for a `sorted` tree that directly saves the data into a path based on the loaded metafile's location.

```
# Tree watch with location
job.watch-sorted.handler        = pyrocore.torrent.watch:TreeWatch
job.watch-sorted.schedule       = hour=*
job.watch-sorted.active         = True

job.watch-sorted.load_mode      = normal
job.watch-sorted.queued         = True
job.watch-sorted.path           = /var/torrent/sorted/watch
job.watch-sorted.cmd.setdir     = {{# set download path
    }}{{if '/music/' in pathname}}{{# add metafile basename to path
        }}d.directory_base.set="/var/torrent/sorted/{{relpath}}/{{pathname|h.pathname}
↪}"{{#
    }}{{elif traits.kind == 'tv'}}{{# store TV content into separate show folders
        }}d.directory.set="/var/torrent/sorted/{{relpath}}/{{traits.get('show', '_
↪UNKNOWN').replace('.',' ').title()}}"{{#
    }}{{else}}{{# just use the relative metafile location
        }}d.directory.set="/var/torrent/sorted/{{relpath}}"{{#
    }}{{endif}}
```

Change the values in the second block to suit your needs. As given, an item loaded from `.../sorted/watch/movies/*.torrent` would end up in the `.../sorted/movies` directorey (with the filename coming from inside the metafile as usual), and it won't start by itself.

Also, paths containing `music` use the metafile's basename as the data directory, and metafiles recognized as TV content get separated into show directories.

Experimental Features

Warning: The features described here are *unfinished* and in an alpha or beta stage.

Query Optimization

You can provide the `--fast-query` option of `rtcontrol` to set a level of optimization to use when querying *rTorrent* for items. The default for that option is set via the `fast_query` config parameter, and is `0` if not changed. That means optimization is normally off, and can be activated via `-Q1`. It is recommended to keep it that way for now,

and use `-Q1` explicitly in scripts and other background processing to reduce the load they generate. Only activating it in scripts usually means the filters used don't change that much, i.e. you can be pretty sure the optimization does what you expect it to do.

Level 1 is less aggressive and safe by definition (i.e. produces correct results in all cases, unless there's a bug), while `-Q2` is highly experimental and in some circumstances likely produces results that are too small or empty.

Optimization works by giving a *pre-filter* condition to *rTorrent*, to reduce the overhead involved in sending items over XMLRPC and processing them, only to be then discarded in the `rtcontrol` filter machinery. That pre-filter evaluation needs features of *rTorrent-PS* 1.1 or later, and will produce errors when used with anything else.

This goal of reducing the number of items sent to `rtcontrol` is best achieved if you put a highly selective condition first in a series of conditions combined by `AND`. For cron-type jobs, this can often be achieved by looking at recent items only – older items should already be processed by previous runs. Even a very lenient window like "last week" drastically reduces items that need to be processed.

Consider this example:

```
$ rtcontrol loaded=-6w is_ignored=0 -o- -v -Q0
DEBUG    Matcher is: loaded=-6w is_ignored=no
DEBUG    Got 131 items with 20 attributes ...
INFO     Filtered 13 out of 131 torrents.
DEBUG    XMLRPC stats: 25 req, out 5.6 KiB [1.4 KiB max], in 104.9 KiB [101.5 KiB␣
↪max], ...
INFO     Total time: 0.056 seconds.

$ rtcontrol loaded=-6w is_ignored=0 -o- -v -Q1
INFO     !!! pre-filter: greater=value=$d.custom=tm_loaded,value=1488920876
DEBUG    Got 17 items with 20 attributes ...
INFO     Filtered 13 out of 131 torrents.
DEBUG    XMLRPC stats: 25 req, out 5.7 KiB [1.5 KiB max], in 16.6 KiB [13.2 KiB max],␣
↪...
INFO     Total time: 0.028 seconds.
```

You can see that the 2nd command executes faster (the effect is larger with more overall items), and only looks at 17 items to select the final 13 ones, while with `-Q0` all 131 items need to be looked at, and thus transferred via XMLRPC. That means 105 KiB instead of only 16.6 KiB need to be serialized, read, and parsed again.

Putting the right condition first is quite important, as you can see when the conditions are swapped and the less selective one is used for the pre-filter:

```
$ rtcontrol is_ignored=0 loaded=-6w -o- -v -Q1
INFO     !!! pre-filter: equal=d.ignore_commands=,value=0
DEBUG    Got 117 items with 20 attributes ...
```

Be careful when mixing `--anneal` and `--fast-query`, since most of the post-processing steps also look at deselected items, and produce unexpected results if they are missing due to pre-filtering. Put another way, always include `-Q0` when you use `--anneal`, to be on the safe side.

Connecting via SSH

Starting with version 0.4.1, you can use URLs of the form

```
scgi+ssh://[«user»@]«host»[:«port»]«/path/to/unix/domain/socket»
```

to connect securely to a remote rTorrent instance. For this to work, the following preconditions have to be met:

- the provided account has to have full permissions (`rwx`) on the given socket.

- you have to use either public key authentication via `authorized_keys`, or a SSH agent that holds your password.
- the remote host needs to have the `socat` executable available (on Debian/Ubuntu, install the `socat` package).

You also need to extend the `rtorrent.rc` of the remote instance with this snippet:

```
# COMMAND: Return startup time (can be used to calculate uptime)
method.insert = startup_time,value|const,$system.time=
```

For example, the following queries the remote instance ID using `rtxmlrpc`:

```
rtxmlrpc -v -Dscgi_url=scgi+ssh://user@example.com/var/torrent/.scgi_local session.
↪name
```

This typically takes several seconds due to the necessary authentication.

Using the Monitoring Web Service

Overview

Note: This feature is not finished and should not be considered stable at this time (i.e. it might change drastically).

The monitoring subsystem is an optional part of `pyrotorque` and includes a web service that creates the monitoring pages which can be viewed in your browser. There is a live view that continuously updates current performance indicators of rTorrent and the host it runs on, something similar to this:

Fig. 1.1: Screenshot of the Monitoring View

What can you see here?

- rTorrent and host uptimes.
- rTorrent upload and download activity.
- number of rTorrent items in total (), active (), having a message (), complete (), incomplete (), seeding (), downloading (). started (), stopped ().
- and key host performance indicators.

The web interface follows *responsive web design* (RWD) principles, which means it'll adapt to different devices and their display size.

(This is not yet implemented. . .) Also, the `StatsArchiver` job of the `pyrotorque` demon writes a lot of statistical data to RRD archives (*round robin database*) in 1 minute intervals. See http://oss.oetiker.ch/rrdtool/doc/rrdtool.en.html for the theory behind RRD, and the standard implementation used in a lot of systems.

Installation & Configuration

As previously mentioned, monitoring is an optional part of `pyrotorque`, so *first* see *rTorrent Queue Manager* on how to set it up in case you didn't do that already. *After* `pyrotorque` is successfully running, follow these additional steps to activate the web server.

A few additional Python libraries and external CSS/Javascript resources need to be installed, which are not part of the core distribution.

1. Install current code and dependencies:

```
~/.local/pyroscope/update-to-head.sh
~/.local/pyroscope/bin/pip install -r ~/.local/pyroscope/requirements-torque.txt
```

2. Activate the web server option by adding this to your `~/.pyroscope/torque.ini`:

```
httpd.active = True
```

3. Download resources to `~/.pyroscope/htdocs`:

```
pyroadmin --create-config
```

4. Finally, restart the demon:

```
pyrotorque --cron --restart
# use "pyrotorque --fg --restart -v" instead, in case something doesn't work,
# so you can directly read the log
```

If you didn't change the defaults, the web interface is now available using the URL http://localhost:8042/, which will show you something similar to the screen shot further above,

Additional Configuration Options

As with other config files, `~/.pyroscope/torque.ini.default` lists all the available options and a short description. The following just lists those that are quite often changed from the defaults.

`httpd.waitress.host` The address the web server listens on. The default is `127.0.0.1` (i.e. `localhost`), and can be changed to `0.0.0.0` to listen to *any* interface. Note that the latter is only safe in your home LAN, behind a firewall or NAT. Add a *reverse proxy* to your Apache/nginx/... for exposing the web service to the internet, ideally adding password protection and using SSL.

`httpd.waitress.port` TCP port the web server listens on, default is `8042`.

`httpd.json.disk_usage_path` Path used to get disk used/total, this can also be a list of paths to different partitions, separated by `:`. The default is your home directory `~`.

Sensors

The following values are gathered. Most (*all?*) of them are also available per tracker (*and per media type?*).

Item Numbers `d.total`, `d.started`, `d.stopped`, `d.complete`, `d.incomplete`, `d.seeding`, `d.leeching`, `d.active`, `d.messages`

These are the associated view sizes; could be sampled more often, and the average values taken.

Item Size `d.size_bytes`, `d.left_bytes`, `d.size_files`

Traffic `d.up_rate`, `d.down_rate`, `d.skip_rate`

Resources `open_sockets`, `cputime`, `pcpu`, `pmem`, `sz`, `rsz`, `vsz`

See `man ps` for most of these.

Also, the usual machine statistics (CPU load, disk usage and I/O, network traffic) are sampled (by `collectd`, or using `collectd` plugins, or some system stats package?).

Later Extensions

These are probably not sampled that often, or we need to define an extra view to allow efficient sampling.

Ratios *As histogram counters?*

Events `event_closed`, ...

Counters for all `event.download.*` events.

Peers `peers_total`, `peers_encrypted`, `peers_incoming`, `peers_obfuscated`, `peers_preferred`, `peers_snubbed`, `peers_unwanted`

Files ...

With some patches compiled into rTorrent, the additional values `network.http.open`, and `network.open_files` are available.

Event Handling

TODO – see the old docs for anything not yet moved.

Queue Manager: Planned Features

These aren't implemented yet...

`ExecCommand` (planned)

TODO `pyrocore.torrent.jobs:ExecCommand` runs an external command in a shell, i.e. it simply runs cron jobs. The reasons for not using cron instead are these: 1. You can have all your rTorrent-related background processing at one place, and the commands see the same environment as `pyrotorque`. 1. `pyrotorque` offers more flexible scheduling, including the ability to run jobs at sub-minute intervals.

`RemoteWatch` (planned)

TODO `pyrocore.torrent.watch:RemoteWatch` polls a (S)FTP source for new `.torrent` files, creates a local copy, and loads that into the client.

`ItemPoller` (planned)

TODO `pyrocore.torrent.:` maintains an updated copy of all rTorrent items, as a service for the other jobs.

`ActionRule` (planned)

TODO `pyrocore.torrent.filter:ActionRule` is `rtcontrol` in form of a house-keeping job, and using this is way more efficient than an equivalent `rtcontrol` cron job; due to that, they can be run a lot more frequently.

`TorrentMirror` (planned)

TODO `pyrocore.torrent.filter:TorrentMirror` allows you to transfer a torrent's data from the local client to other remote clients using a specified tracker (at the start, a locally running "bttrack"). In a nutshell, it allows you to transfer any filtered item automatically to a remote location via bittorrent.

`CompletionHandler` (planned)

TODO `pyrocore.torrent.:` moves completed data to a target directory, according to flexible rules.

`StatsArchiver` (planned)

TODO `pyrocore.torrent.:` keeps a continuous archive of some statistical values (like bandwidth) so they can later be rendered into graphs.

See RtorrentMonitoring for more details.

Custom Python Code

You can write your own code for `pyrocore` implementing custom features, by adding fields, your own command line scripts, or `pyrotorque` jobs. You probably need a solid grasp of Python for this.

Defining Custom Fields

Introduction

As mentioned in the *Configuration Guide*, the `config.py` script can be used to add custom logic to your setup. The most common use for this file is adding custom fields.

To add user-defined fields you can put code describing them into your `~/.pyroscope/config.py` file. You can then use your custom field just like any built-in one, e.g. issue a command like `rtcontrol --from-view incomplete \* -qco partial_done,name` (see below examples). They're also listed when you call `rtcontrol --help-fields`.

Basic Custom Field Code

The following is the framework you need to add before putting in your field definitions:

```
def _custom_fields():
    """ Yield custom field definitions.
    """
    # Import some commonly needed modules
    import os
    from pyrocore.torrent import engine, matching
    from pyrocore.util import fmt

    # PUT CUSTOM FIELD CODE HERE

# Register our factory with the system
custom_field_factories.append(_custom_fields)
```

In place of the `# PUT CUSTOM FIELD CODE HERE` comment you can add any combination of the examples below, or your own code. Be sure to do so at the correct indent level, the example snippets are left-aligned and need to be indented by 4 spaces.

Custom Field Examples

Adding rTorrent fields not supported by default

```
# Add rTorrent attributes not available by default
def get_tracker_field(obj, name, aggregator=sum):
    "Get an aggregated tracker field."
    return aggregator(obj._engine._rpc.t.multicall(obj._fields["hash"], 0, "t.%s=" %␣
↪name)[0])

yield engine.OnDemandField(int, "peers_connected", "number of connected peers",␣
↪matcher=matching.FloatFilter)
yield engine.DynamicField(int, "downloaders", "number of completed downloads",␣
↪matcher=matching.FloatFilter,
    accessor=lambda o: get_tracker_field(o, "get_scrape_downloaded"))
yield engine.DynamicField(int, "seeds", "number of seeds", matcher=matching.
↪FloatFilter,
    accessor=lambda o: get_tracker_field(o, "get_scrape_complete"))
yield engine.DynamicField(int, "leeches", "number of leeches", matcher=matching.
↪FloatFilter,
    accessor=lambda o: get_tracker_field(o, "get_scrape_incomplete"))
yield engine.DynamicField(engine.untyped, "lastscraped", "time of last scrape",␣
↪matcher=matching.TimeFilter,
    accessor=lambda o: get_tracker_field(o, "get_scrape_time_last", max),
    formatter=lambda dt: fmt.human_duration(float(dt), precision=2, short=True))

# Add peer attributes not available by default
def get_peer_data(obj, name, aggregator=None):
    "Get some peer data via a multicall."
    aggregator = aggregator or (lambda _: _)
    result = obj._engine._rpc.p.multicall(obj._fields["hash"], 0, "p.%s=" % name)
    return aggregator([i[0] for i in result])

yield engine.DynamicField(set, "peers_ip", "list of IP addresses for connected peers",
    matcher=matching.TaggedAsFilter, formatter=", ".join,
    accessor=lambda o: set(get_peer_data(o, "address")))
```

Checking that certain files are present

```
# Add file checkers
def has_nfo(obj):
    "Check for .NFO file."
    pathname = obj.path
```

```
    if pathname and os.path.isdir(pathname):
        return any(i.lower().endswith(".nfo") for i in os.listdir(pathname))
    else:
        return False if pathname else None

def has_thumb(obj):
    "Check for folder.jpg file."
    pathname = obj.path
    if pathname and os.path.isdir(pathname):
        return any(i.lower() == "folder.jpg" for i in os.listdir(pathname))
    else:
        return False if pathname else None

yield engine.DynamicField(engine.untyped, "has_nfo", "does download have a .NFO file?
↪",
    matcher=matching.BoolFilter, accessor=has_nfo,
    formatter=lambda val: "NFO" if val else "!DTA" if val is None else "----")
yield engine.DynamicField(engine.untyped, "has_thumb", "does download have a folder.
↪jpg file?",
    matcher=matching.BoolFilter, accessor=has_thumb,
    formatter=lambda val: "THMB" if val else "!DTA" if val is None else "----")
```

Calculating information about partial downloads

Note that the `partial_done` value can be a little lower than it actually should be, when chunks shared by different files are not yet complete; but it will eventually reach `100` when all selected chunks are downloaded in full.

```
# Fields for partial downloads
def partial_info(obj, name):
    "Helper for partial download info"
    try:
        return obj._fields[name]
    except KeyError:
        f_attr = ["get_completed_chunks", "get_size_chunks", "get_range_first", "get_
↪range_second"]
        chunk_size = obj.fetch("chunk_size")
        prev_chunk = -1
        size, completed, chunks = 0, 0, 0
        for f in obj._get_files(f_attr):
            if f.prio: # selected?
                shared = int(f.range_first == prev_chunk)
                size += f.size
                completed += f.completed_chunks - shared
                chunks += f.size_chunks - shared
                prev_chunk = f.range_second - 1

        obj._fields["partial_size"] = size
        obj._fields["partial_missing"] = (chunks - completed) * chunk_size
        obj._fields["partial_done"] = 100.0 * completed / chunks if chunks else 0.0

        return obj._fields[name]

yield engine.DynamicField(int, "partial_size", "bytes selected for download",
    matcher=matching.ByteSizeFilter,
    accessor=lambda o: partial_info(o, "partial_size"))
yield engine.DynamicField(int, "partial_missing", "bytes missing from selected chunks
↪",
```

```
    matcher=matching.ByteSizeFilter,
    accessor=lambda o: partial_info(o, "partial_missing"))
yield engine.DynamicField(float, "partial_done", "percent complete of selected chunks
↪",
    matcher=matching.FloatFilter,
    accessor=lambda o: partial_info(o, "partial_done"))
```

Extract TV data from item name

This defines the `tv_series` and `tv_episode` fields, that are non-empty when the item name follows the "usual" naming conventions. Try it using something like `rtcontrol loaded=-2w traits=tv -co tv_series,tv_episode,name`.

```
# Map name field to TV series name, if applicable, else an empty string
from pyrocore.util import traits

def tv_mapper(obj, name, templ):
    "Helper for TV name mapping"
    try:
        return obj._fields[name]
    except KeyError:
        itemname = obj.name
        result = ""

        kind, info = traits.name_trait(itemname, add_info=True)
        if kind == "tv":
            try:
                info["show"] = ' '.join([i.capitalize() for i in info["show"].replace(
↪'.',' ').replace('_',' ').split()])
                result = templ % info
            except KeyError, exc:
                #print exc
                pass

        obj._fields[name] = result
        return result

yield engine.DynamicField(fmt.to_unicode, "tv_series", "series name of a TV item",
    matcher=matching.PatternFilter, accessor= lambda o: tv_mapper(o, "tv_series", "
↪%(show)s"))
yield engine.DynamicField(fmt.to_unicode, "tv_episode", "series name and episode␣
↪number of a TV item",
    matcher=matching.PatternFilter, accessor= lambda o: tv_mapper(o, "tv_episode", "
↪%(show)s.S%(season)sE%(episode)s"))
```

Only start items that you have disk space for

This works together with *rTorrent Queue Manager*, so that only items that pass a disk space check are actually started.

The first step is to add a custom field that checks whether an item has room on the target device. As with the other examples, place this in your `config.py` (read the 1st two sections, before the "Examples" one).

```
# Disk space check
def has_room(obj):
```

```
    "Check disk space."
    pathname = obj.path
    if pathname and os.path.exists(pathname):
        stats = os.statvfs(pathname)
        return stats.f_bavail * stats.f_frsize - int(diskspace_threshold_mb) *␣
↪1024**2 > obj.size * (1.0 - obj.done / 100.0)
    else:
        return None

yield engine.DynamicField(engine.untyped, "has_room", "check whether the download␣
↪will fit on its target device",
    matcher=matching.BoolFilter, accessor=has_room,
    formatter=lambda val: "OK" if val else "??" if val is None else "NO")
globals().setdefault("diskspace_threshold_mb", "500")
```

Note that you can set the threshold of space to keep free (in MiB) in your configuration, and the default is 500MiB. You should **keep** your `close_low_diskspace` schedule for rTorrent as a fallback, and set `diskspace_threshold_mb` **higher** than the limit given there (so that normally, it never triggers).

And now, all you need is to add `has_room=y` to your `job.queue.startable` conditions. Done.

Adding Custom Template Helpers

In templating contexts, there is an empty `c` namespace (think `custom` or `config`), just like `h` for helpers. You can populate that namespace with your own helpers as you need them, from simple string transformations to calling external programs or web interfaces.

The following example illustrates the concept, and belongs into `~/.pyroscope/config.py`.

```
def _hostname(ip):
    """Helper to e.g. look up peer IPs."""
    import socket

    return socket.gethostbyaddr(ip)[0] if ip else ip

custom_template_helpers.hostname = _hostname
```

This demonstrates the call of that helper using a custom field, a real use-case would be to resolve peer IPs and the like.

```
$ rtcontrol -qo '{{d.fetch("custom_ip")}} → {{d.fetch("custom_ip") | c.hostname}}' //␣
↪-/1
8.8.8.8 → google-public-dns-a.google.com
```

Writing Your Own Scripts

Introduction

The `pyrocore` Python package contains powerful helper classes that make remote access to *rTorrent* child's play (see *API Documentation*). And your tools get the same *Look & Feel* like the built-in *PyroScope* commands, as long as you use the provided base class *`pyrocore.scripts.base.ScriptBaseWithConfig`*.

See for yourself:

```
#! /usr/bin/env python-pyrocore
# -*- coding: utf-8 -*-
```

```
# Enter the magic kingdom
from pyrocore import config
from pyrocore.scripts import base


class UserScript(base.ScriptBaseWithConfig):
    """
        Just some script you wrote.
    """

    # argument description for the usage information
    ARGS_HELP = "<arg_1>... <arg_n>"

    # set your own version
    VERSION = '1.0'

    # (optionally) define your licensing
    COPYRIGHT = u'Copyright (c) ...'

    def add_options(self):
        """ Add program options.
        """
        super(UserScript, self).add_options()

        # basic options
        ##self.add_bool_option("-n", "--dry-run",
        ##    help="don't do anything, just tell what would happen")


    def mainloop(self):
        """ The main loop.
        """
        # Grab your magic wand
        proxy = config.engine.open()

        # Wave it
        torrents = list(config.engine.items())

        # Abracadabra
        print "You have loaded %d torrents tracked by %d trackers." % (
            len(torrents),
            len(set(i.alias for i in torrents)),
        )

        self.LOG.info("XMLRPC stats: %s" % proxy)


if __name__ == "__main__":
    base.ScriptBase.setup()
    UserScript().run()
```

Another full example is the dynamic seed throttle script.

Note: If you wondered about the first line referring to a `python-pyrocore` command, that is an alias the installation scripts create for the Python interpreter of the *pyrocore* virtualenv. This way, your script will always use the

correct environment that actually offers the right packages.

For simple calls, you can also use the `rtxmlrpc` command on a shell prompt, see *Using 'rtxmlrpc'* for that. For a reference of the *rTorrent* XMLRPC interface, see *rTorrent XMLRPC*. Another common way to add your own extensions is *Defining Custom Fields*, usable by `rtcontrol` just like built-in ones.

Interactive use in a Python shell

You can also access rTorrent interactively, like this:

```
>>> from pyrocore import connect
>>> rt = connect()
>>> len(set(i.tracker for i in rt.items()))
2
>>> rt.engine_software
'rTorrent 0.9.2/0.13.2'
>>> rt.uptime
1325.6771779060364
>>> proxy = rt.open()
>>> len(proxy.system.listMethods())
1033
```

Using `pyrocore` as a library in other projects

The example in the first section is an easy way to create user-defined scripts. If you want to use `pyrocore`'s features in another runtime environment, you just have to load the configuration manually (what *`pyrocore.scripts.base.ScriptBaseWithConfig`* does for you otherwise).

```
# Details depend on the system you want to extend, of course
from some_system import plugin
from pyrocore import error
from pyrocore.util import load_config

def my_rtorrent_plugin():
    """ Initialize plugin.
    """
    try:
        load_config.ConfigLoader().load()
    except error.LoggableError as exc:
        # Handle accordingly...
    else:
        # Do some other stuff...

plugin.register(my_rtorrent_plugin)
```

Code snippets

Note: The following snippets are meant to be placed and executed within the `mainloop` of the script skeleton found in *Introduction*.

Accessing the files in a download item

To get all the files for several items at once, we combine `system.multicall` and `f.multicall` to one big efficient mess.

```
from pprint import pprint, pformat

# The attributes we want to fetch
methods = [
    "f.get_path",
    "f.get_size_bytes",
    "f.get_last_touched",
    "f.get_priority",
    "f.is_created",
    "f.is_open",
]

# Build the multicall argument
f_calls = [method + '=' for method in methods]
calls = [{"methodName": "f.multicall", "params": [infohash, 0] + f_calls}
    for infohash in self.args
]

# Make the calls
multicall = proxy.system.multicall
result = multicall(calls)

# Print the results
for infohash, (files,) in zip(self.args, result):
    print ("~~~ %s [%d file(s)] " % (infohash, len(files))).ljust(78, '~')
    pprint(files)
self.LOG.info("Multicall stats: %s" % multicall)
```

Writing Custom Jobs

First off, you really need to know a good amount of Python to be able to do this. But if you do, you can easily add your own background processing, more versatile and more efficient than calling `rtcontrol` in a cron job. The description here is terse, and mostly just tells you where to look for code examples, and the basics of how a job implementation interacts with the core system.

Note: While some effort will be spent on keeping the API backwards compatible, there is no guarantee of a stable API. Follow the commit log and changelogs of releases to get notified when you need to adapt your code.

Jobs are created during `pyrotorque` startup and registered with the scheduler. Configuration is taken from the `[TORQUE]` section of `torque.ini`, and any `job.«job-name».«param-name»` setting contributes to a job named `job-name`. The `handler`, `schedule`, and `active` settings are used by the core, the rest is passed to the `handler` class for customization and depends on the job type.

To locate the job implementation, `handler` contains a `module.path:ClassName` coordinate of its class. So `job.foo.handler = my.code::FooJob` registers `FooJob` under the name `foo`. This means a job can be scheduled several times, given the right configuration and if the job implementation is designed for it. The given module must be importable of course, i.e. `pip install` it into your `pyrocore` virtualenv.

The `schedule` defines the call frequency of the job's `run` method, and `active` allows to easily disable a job without removing its configuration – which is used to provide all the default jobs and their settings. A job with `active =`

`False` is simply ignored and not added to the scheduler on startup.

The most simple of jobs is the *`EngineStats`* one. Click on the link and then on `[source]` to see its source code. Some noteworthy facts:

- the initializer gets passed a `config` parameter, holding all the settings from `torque.ini` for a particular job instance, with the `job.`«`name`» prefix removed.
- `pyrocore.config` is imported as `config_ini`, to not clash with the `config` dict passed into jobs.
- create a `LOG` attribute as shown, for your logging needs.
- to interact with *rTorrent*, open a proxy connection in `run`.
- the `InfluxDB` job shows how to access config parameters, e.g. `self.config.dbname`.
- raise *`UserError`* in the initializer to report configuration mishaps and prevent `pyrotorque` from starting.

More complex jobs that you can look at are the *`pyrocore.torrent.watch.TreeWatch`* and *`pyrocore.torrent.queue.QueueManager`* ones.

Trouble-Shooting Guide

Reporting Problems

If you have any trouble during *pyrocore* installation and configuration, or using any of the commands, join the pyroscope-users mailing list or the inofficial `##rtorrent` channel on `irc.freenode.net`. IRC will generally provide a faster resolution.

If you are sure there is a bug, then open an issue on *GitHub*. Make sure that nobody else reported the same problem before you, there is a search box you can use (after the **Filter** button). Please note that the *GitHub* issue tracker is not a support platform, use the mailing list or IRC for that.

Note: Please **describe your problem clearly**, and provide any pertinent information. What are the **version numbers** of software and OS? What did you do? What was the **unexpected result**? If things worked and 'suddenly' broke, **what did you change**?

On IRC, don't ask if somebody is there, just describe your problem. Eventually, someone will notice you – IRC is a global medium, and people *do* live in different time zones than you.

Put up any logs on 0bin or any other pastebin service, and **make sure you removed any personal information** you don't want to be publically known. Copy the pastebin link into IRC or into your post.

The following helps with querying your system environment, e.g. the version of Python and your OS.

Providing Diagnostic Information

Python Diagnostics

Execute the following command to be able to provide some information on your Python installation:

```
deactivate 2>/dev/null; /usr/bin/virtualenv --version; python <<'.'
import sys, os, time, pprint
pprint.pprint(dict(
    version=sys.version,
    prefix=sys.prefix,
```

```
    os_uc_names=os.path.supports_unicode_filenames,
    enc_def=sys.getdefaultencoding(),
    maxuchr=sys.maxunicode,
    enc_fs=sys.getfilesystemencoding(),
    tz=time.tzname,
    lang=os.getenv("LANG"),
    term=os.getenv("TERM"),
    sh=os.getenv("SHELL"),
))
.
```

If `enc_fs` is **not** `UTF-8`, then call `dpkg-reconfigure locales` (on Debian type systems) and choose a proper locale (you might also need `locale-gen en_US.UTF-8`), and make sure `LANG` is set to `en_US.UTF-8` (or another locale with UTF-8 encoding).

OS Diagnostics

Similarly, execute this in a shell prompt:

```
uname -a; echo $(lsb_release -as 2>/dev/null); grep name /proc/cpuinfo | uniq -c; \
free -m | head -n2; uptime; \
strings $(which rtorrent) | grep "client version"; \
ldd $(which rtorrent) | egrep "lib(torrent|curses|curl|xmlrpc.so|cares|ssl|crypto)"; \
ps auxw | egrep "USER|/rtorrent" | grep -v grep
```

Common Problems & Solutions

"rTorrent-PS features NOT active!" during rTorrent startup

As mentioned several times in the installation guides, you *must* tell the standard `rtorrent.rc` configuration include that it's loaded into a *rTorrent-PS* installation.

The way to do that is setting the `pyro.extended` constant to `1`:

```
method.insert = pyro.extended, value|const, 1
```

If you do that and are *not* actually running *rTorrent-PS*, you *will* get errors about missing commands during startup.

"Input failed: ExecFile::execute(...) Fork failed." during searches

This is always a host setup problem, check your resource limits, namely `nofile` (max. number of open file handles) and `nproc` (max. number of processes). Also look into `~/rtorrent/log/execute.log` if it contains any hints.

"Scheduled command failed: bind_home: Bad key definition." during startup

Your rTorrent-PS is too old to support the new default key bindings for `Home`, `End`, and so on. Either compile the HEAD of rTorrent-PS to update your installation, or else do this:

```
echo >>~/.pyroscope/rtorrent.d/.rcignore "bind-navigation-keys.rc.default"
```

Software Updates

Warning: The syntax of XMLRPC commands changed with rTorrent version 0.8.9, and continues to change. Make sure that the versions of rTorrent and PyroScope you plan to install or update to are actually compatible. There are compensation mechanisms in both projects, but there are limits to those — scan the respective changelogs for breaking changes.

pyrocore 0.5+ will no longer support the old syntax, and thus not work with *rTorrent* 0.8.x versions. *rTorrent* 0.9.6 has the old commands disabled by default, and only a special command line switch will enable them again, *for now*. Also, this documentation uses the new syntax (mostly).

Making Backups

Since repairing broken files resulting from faulty updates usually is either a lot of work or simply impossible, always **make a backup**. Backups should be made when *either* PyroScope or rTorrent is changed to a new release version or git revision.

These steps should make a copy of pretty much anything important:

1. Copy your *rTorrent* session data and configuration (`rtorrent` needs to be running):

```
rtxmlrpc -q session.save
tar cvfz /tmp/instance-backup-$USER-$(date +'%Y-%m-%d').tgz \
    $(echo $(rtxmlrpc session.path)/ | tr -s / /)*.torrent \
    ~/rtorrent/*.rc ~/rtorrent/rtorrent.d ~/rtorrent/start
```

2. Backup your current *PyroScope* virtualenv and configuration (use `~/lib` instead of `~/.local` for installations before `0.5.1`):

```
tar cvfz /tmp/pyroscope-backup-$USER-$(date +'%Y-%m-%d').tgz \
    ~/.pyroscope/ ~/.local/pyroscope/
```

3. Depending on how you installed *rTorrent*, make a copy of the `rtorrent` executable and `libtorrent*.so*`. Note that the rTorrent-PS build script installs into versioned directories, i.e. using that you don't have to worry if changing to a new *rTorrent* version — the old one is still available, and you can switch back easily.

Updating the Software

Before adapting and extending your configuration to make use of new features, you first have to update the software itself. How to do that depends on the way you initially installed it, so follow **one** of the following sections, depending on whether you did a *release installation* or one *from source*.

How to Do a Release Version Software Update

Remember to read the **migration instructions** further below, and the changelog, **BEFORE** installing any new version.

Then to **update** an existing installation, use these commands (but note the `0.5.1` update is different, see below):

```
cd ~/.local/pyroscope
bin/pip install -U "pyrocore[templating]"
ln -nfs $(egrep -l '(from.pyrocore.scripts|entry_point.*pyrocore.*console_scripts)'
↪$PWD/bin/*) ~/bin
```

If you used `pip install --user -U pyrocore` without creating a virtualenv, just repeat that command.

Now **skip** the next section describing a source installation upgrade, and go to the *configuration update* further below.

How to Update a Source Installation to the Newest Code

BEFORE any update, remember to read the **migration instructions** further below, the changelog and the list of commits.

Then to **update** an existing installation, use this command:

```
~/.local/pyroscope/update-to-head.sh
```

Continue with any tasks regarding configuration changes from the next section.

Updating Your Configuration

After you installed a new version of the software, you have to check for necessary changes to the default configuration, after calling the `pyroadmin --create-config` or the `update-to-head.sh` command.

Note that only the `*.default` files (`config.ini.default`, `config.py.default`, and so on) will be overwritten, they are a literal copy of the defaults packaged into the software, and are there for informational purposes only. You can then use the `diff` tool to check for the differences between your current configuration and the new default one, and add any changes you want to adopt.

Also note that sections of the configuration you leave out, and keys that you do not overwrite, are automatically taken from the defaults, which greatly simplifies any update. That is the reason why it is recommended to have a minimal configuration with just your customizations, in addition to the defaults.

The file `~/.pyroscope/rtorrent-pyro.rc.default`, and those contained in `~/.pyroscope/rtorrent.d`, are a different story. They change quite often, and since there is no merging of `*.rc.default` with `*.rc` files, the default ones are normally used. You can still disable those default files one by one using the `rtorrent.d/.rcignore` file, in order to provide your own versions or simply disable certain features. That is way better than switching altogether to `*.rc` files, again for the reason updates become way more painless. See the comments at the start of files in `rtorrent.d` for details.

And remember to **always** read the changelog!

Migrating to Version 0.5.x

The `0.5.x` release line adds a queue manager, watching a directory tree for loading metafiles, and removes support for ancient versions of *Python* and *rTorrent*. More details on the contained changes can be found at GitHub releases and the changelog. Install at least version `0.5.3`, which has a few important fixes.

To upgrade your existing installation, follow these steps:

1. For people that run a source code installation, just use the `update-to-head.sh` script as described in *Installing from GitHub*. When your old installation is still in `~/lib`, you'll be presented with the necessary commands to move to `~/.local` *after* calling `~/lib/pyroscope/update-to-head.sh`. Since all the documentation now points to `~/.local` paths, you should switch over.

   For PyPI installs, just do a *fresh install* to the new location at `~/.local`.

2. Call `pyroadmin --create-config` to update the `.default` configuration examples, and create the new `rtorrent.d` directory.

3. In your *rTorrent* instance, update the start script (and save a copy of the old one before that).

4. You also **MUST** change the `import` command in your `rtorrent.rc` that loads the PyroScope configuration include:

```
# Remove the ".default" if you want to change something (else your changes
# get over-written on update, when you put them into ``*.default`` files).
import = ~/.pyroscope/rtorrent-pyro.rc.default
```

5. Read the *rTorrent Queue Manager* section if you plan to use item queueing and/or the tree watch feature; both are inactive by default and need to be enabled. You also need to add the new `pyro_watchdog` schedule into your configuration, as shown in the *Configuration Guide*.
6. Remember to restart *rTorrent* after any configuration changes.

When you have a rather aged configuration, also consider switching to the new set of configuration files as found in the `pimp-by-box` project, that use the new command names through-out and are thus way more future-proof.

There is an easy to use `make-rtorrent-config.sh` script, see rTorrent Configuration on how to use it. At the same time, update the start script. Note that these configuration files also work with a plain vanilla *rTorrent* version, you do **not** need *rTorrent-PS* for them to work.

In any case, **make a backup** of your configuration and scripts, as mentioned at the start of this chapter. After creating the new configuration, merge in what's missing from your old configuration, but migrate to the new syntax first. For adding your custom settings, you can use your own files in the `~/rtorrent/rtorrent.d` directory.

Migrating to Version 0.6.1 (UNRELEASED)

...

References

PyroScope CLI Tools Usage

This section is automatically generated and shows the options available in the *development* version of the code (git HEAD). See *User's Manual* for more details on how to use these commands.

- *chtor*
- *hashcheck*
- *lstor*
- *mktor*
- *pyroadmin*
- *pyrotorque*
- *rtcontrol*
- *rtevent*
- *rtmv*
- *rtxmlrpc*

Note: The help output presented here applies to version `0.5.1` of the tools.

chtor

```
Usage: chtor [options] <metafile>...

Change attributes of a bittorrent metafile.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
  -n, --dry-run         don't write changes to disk, just tell what would happen
  -V, --no-skip         do not skip broken metafiles that fail the integrity check
  -o PATH, --output-directory=PATH
                        optional output directory for the modified metafile(s)
  -p, --make-private    make torrent private (DHT/PEX disabled)
  -P, --make-public     make torrent public (DHT/PEX enabled)
  -s KEY=VAL [-s ...], --set=KEY=VAL [-s ...]
                        set a specific key to the given value; omit the '=' to delete␣
↪a key
  -r KEYcREGEXcSUBSTc [-r ...], --regex=KEYcREGEXcSUBSTc [-r ...]
                        replace pattern in a specific key by the given substitution
  -C, --clean           remove all non-standard data from metafile outside the info␣
↪dict
  -A, --clean-all       remove all non-standard data from metafile including inside␣
↪the info dict
  -X, --clean-xseed     like --clean-all, but keep libtorrent resume information
  -R, --clean-rtorrent  remove all rTorrent session data from metafile
  -H DATAPATH, --hashed=DATAPATH, --fast-resume=DATAPATH
                        add libtorrent fast-resume information (use {} in place of␣
↪the torrent's name in DATAPATH)
  -a URL, --reannounce=URL
                        set a new announce URL, but only if the old announce URL␣
↪matches the new one
  --reannounce-all=URL  set a new announce URL on ALL given metafiles
  --no-ssl              force announce URL to 'http'
  --no-cross-seed       when using --reannounce-all, do not add a non-standard field␣
↪to the info dict ensuring unique info hashes
  --comment=TEXT        set a new comment (an empty value deletes it)
  --bump-date           set the creation date to right now
  --no-date             remove the 'creation date' field
```

hashcheck

```
Usage: hashcheck [options] <metafile> [<data-dir-or-file>]

Check a bittorrent metafile.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
```

lstor

```
Usage: lstor [options] <metafile>...

List contents of a bittorrent metafile.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --reveal              show full announce URL including keys
  --raw                 print the metafile's raw content in all detail
  -V, --skip-validation
                        show broken metafiles with an invalid structure
  -o KEY,KEY1.KEY2,..., --output=KEY,KEY1.KEY2,...
                        select fields to print, output is separated by TABs; note␣
↪that __file__ is the path to the metafile,
                        __hash__ is the info hash, and __size__ is the data size in␣
↪bytes
```

mktor

```
Usage: mktor [options] <dir-or-file> <tracker-url-or-alias>... | <magnet-uri>

Create a bittorrent metafile.
```

```
If passed a magnet URI as the only argument, a metafile is created
in the directory specified via the configuration value 'magnet_watch',
loadable by rTorrent. Which means you can register 'mktor' as a magnet:
URL handler in Firefox.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
  -p, --private         disallow DHT and PEX
  --no-date             leave out creation date
  -o PATH, --output-filename=PATH
                        optional file name (or target directory) for the metafile
  -r NAME, --root-name=NAME
                        optional root name (default is basename of the data path)
  -x PATTERN [-x ...], --exclude=PATTERN [-x ...]
                        exclude files matching a glob pattern from hashing
  --comment=TEXT        optional human-readable comment
  -s KEY=VAL [-s ...], --set=KEY=VAL [-s ...]
                        set a specific key to the given value; omit the '=' to delete␣
↪a key
  --no-cross-seed       do not automatically add a field to the info dict ensuring␣
↪unique info hashes
  -X LABEL, --cross-seed=LABEL
                        set additional explicit label for cross-seeding (changes info␣
↪hash)
  -H, --hashed, --fast-resume
                        create second metafile containing libtorrent fast-resume␣
↪information
```

pyroadmin

```
Usage: pyroadmin [options]

Support for administrative tasks.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
```

```
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
  --create-config       create default configuration
  --remove-all-rc-files
                        write new versions of BOTH .rc and .rc.default files, and␣
↪remove stale ones
  --dump-config         pretty-print configuration including all defaults
  --create-import=GLOB-PATTERN
                        create import file for a '.d' directory
  --dump-rc             pretty-print dynamic commands defined in 'rtorrent.rc'
  -o KEY,KEY1.KEY2=DEFAULT,..., --output=KEY,KEY1.KEY2=DEFAULT,...
                        select fields to print, output is separated by TABs; default␣
↪values can be provided after the key
  --reveal              show config internals and full announce URL including keys
  --screenlet           create screenlet stub
```

pyrotorque

```
Usage: pyrotorque [options]

rTorrent queue manager & daemon.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
  -n, --dry-run         advise jobs not to do any real work, just tell what would␣
↪happen
  --no-fork, --fg       Don't fork into background (stay in foreground and log to␣
↪console)
  --stop                Stop running daemon
  --restart             Stop running daemon, then fork into background
  -?, --status          Check daemon status
  --pid-file=PATH       file holding the process ID of the daemon, when running in␣
↪background
  --guard-file=PATH     guard file for the process watchdog
```

rtcontrol

```
Usage: rtcontrol [options] <filter>...

Control and inspect rTorrent from the command line.

Filter expressions take the form "<field>=<value>", and all expressions must
be met (AND). If a field name is omitted, "name" is assumed. You can also use
uppercase OR to build a list of alternative conditions.

For numeric fields, a leading "+" means greater than, a leading "-" means less
than. For string fields, the value is a glob pattern (*, ?, [a-z], [!a-z]), or
a regex match enclosed by slashes. All string comparisons are case-ignoring.
Multiple values separated by a comma indicate several possible choices (OR).
"!" in front of a filter value negates it (NOT).

See https://pyrocore.readthedocs.io/en/latest/usage.html#rtcontrol for more.

Examples:
  - All 1:1 seeds         ratio=+1
  - All active torrents   xfer=+0
  - All seeding torrents  up=+0
  - Slow torrents         down=+0 down=-5k
  - Older than 2 weeks    completed=+2w
  - Big stuff             size=+4g
  - 1:1 seeds not on NAS  ratio=+1 'realpath=!/mnt/*'
  - Music                 kind=flac,mp3

Use --help to get a list of all options.
Use --help-fields to list all fields and their description.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
  --help-fields         show available fields and their description
  -n, --dry-run         don't commit changes, just tell what would happen
  --detach              run the process in the background
  -i, --interactive     interactive mode (prompt before changing things)
  --yes                 positively answer all prompts (e.g. --delete --yes)
  -S, --shell           escape output following shell rules
  -0, --nul, --print0   use a NUL character instead of a linebreak after items
  -c, --column-headers  print column headers
  -+, --stats           add sum / avg / median of numerical fields
  --summary             print only statistical summary, without the items
  --json                dump all items as JSON (use '-o f1,f2,...' to specify fields)
  -o FORMAT, --output-format=FORMAT
```

```
                        specify display format (use '-o-' to disable item display)
  -O FILE, --output-template=FILE
                        pass control of output formatting to the specified template
  -s [-]FIELD[,...] [-s...], --sort-fields=[-]FIELD[,...] [-s...]
                        fields used for sorting, descending if prefixed with a '-'; '-
↪s*' uses output field list
  -r, --reverse-sort    reverse the sort order
  -A MODE [-A...], --anneal=MODE [-A...]
                        modify result set using some pre-defined methods
  -/ [N-]M, --select=[N-]M
                        select result subset by item position (counting from 1)
  -V, --view-only       show search result only in default ncurses view
  --to-view=NAME        show search result only in named ncurses view
  --tee-view            ADDITIONALLY show search results in ncurses view (modifies -V␣
↪and --to-view behaviour)
  --from-view=NAME      select only items that are on view NAME (NAME can be an info␣
↪hash to quickly select a single item)
  -M NAME, --modify-view=NAME
                        get items from given view and write result back to it (short-
↪cut to combine --from-view and --to-view)
  -Q LEVEL, --fast-query=LEVEL
                        enable query optimization (=: use config; 0: off; 1: safe; 2:␣
↪danger seeker) [=]
  --call=CMD            call an OS command pattern in the shell
  --spawn=CMD [--spawn ...]
                        execute OS command pattern(s) directly
  --start               start torrent
  --close, --stop       stop torrent
  -H, --hash-check      hash-check torrent (implies -i)
  --delete              remove torrent from client (implies -i)
  --purge, --delete-partial
                        delete PARTIAL data files and remove torrent from client␣
↪(implies -i)
  --cull, --exterminate, --delete-all
                        delete ALL data files and remove torrent from client (implies␣
↪-i)
  -T NAME, --throttle=NAME
                        assign to named throttle group (NULL=unlimited, NONE=global)␣
↪(implies -i)
  --tag="TAG +TAG -TAG..."
                        add or remove tag(s)
  --custom=KEY=VALUE    set value of 'custom_KEY' field (KEY might also be 1..5)
  --exec=CMD, --xmlrpc=CMD
                        execute XMLRPC command pattern (implies -i)
  --ignore=0|1          set 'ignore commands' status on torrent
  --prio=0|1|2|3        set priority of torrent
  -F, --flush           flush changes immediately (save session data)

Fields are:
  active                last time a peer was connected
  alias                 tracker alias or domain
  completed             time download was finished
  custom_KEY            named rTorrent custom attribute, e.g. 'custom_completion_
↪target'
  directory             directory containing download data
  done                  completion in percent
  down                  download rate
  files                 list of files in this item
```

```
  fno                   number of files in this item
  hash                  info hash
  is_active             download active?
  is_complete           download complete?
  is_ghost              has no data file or directory?
  is_ignored            ignore commands?
  is_multi_file         single- or multi-file download?
  is_open               download open?
  is_private            private flag set (no DHT/PEX)?
  kind                  ALL kinds of files in this item (the same as kind_0)
  kind_N                file types that contribute at least N% to the item's total␣
↪size
  leechtime             time taken from start to completion
  loaded                time metafile was loaded
  message               current tracker message
  metafile              path to torrent file
  name                  name (file or root directory)
  path                  path to download data
  prio                  priority (0=off, 1=low, 2=normal, 3=high)
  ratio                 normalized ratio (1:1 = 1.0)
  realpath              real path to download data
  seedtime              total seeding time after completion
  sessionfile           path to session file
  size                  data size
  started               time download was FIRST started
  stopped               time download was last stopped or paused
  tagged                has certain tags? (not related to the 'tagged' view)
  throttle              throttle group name (NULL=unlimited, NONE=global)
  tracker               first in the list of announce URLs
  traits                automatic classification of this item (audio, video, tv,␣
↪movie, etc.)
  up                    upload rate
  uploaded              amount of uploaded data
  views                 views this item is attached to
  xfer                  transfer rate

Format specifiers are:
  delta                 Format a UNIX timestamp to a delta (relative to now).
  duration              Format a duration value in seconds to a readable form.
  iso                   Format a UNIX timestamp to an ISO datetime string.
  json                  JSON serialization.
  mtime                 Modification time of a path.
  pathbase              Base name of a path.
  pathdir               Directory containing the given path.
  pathext               Extension of a path (including the '.').
  pathname              Base name of a path, without its extension.
  pc                    Scale a ratio value to percent.
  raw                   Switch off the default field formatter.
  strip                 Strip leading and trailing whitespace.
  subst                 Replace regex with string.
  sz                    Format a byte sized value.

Append format specifiers using a '.' to field names in '-o' lists,
e.g. 'size.sz' or 'completed.raw.delta'.
```

rtevent

```
Usage: rtevent [options] <event> <infohash> [<args>...]

Handle rTorrent events.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
  --no-fork, --fg       Don't fork into background (stay in foreground, default for␣
↪terminal use)
```

rtmv

```
Usage: rtmv [options] <source>... <target>

Move data actively seeded in rTorrent.

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
  -n, --dry-run         don't move data, just tell what would happen
  -F, --force-incomplete
                        force a move of incomplete data
```

rtxmlrpc

```
Usage: rtxmlrpc [options] <method> <args>...

Perform raw rTorrent XMLRPC calls, like "rtxmlrpc throttle.up.rate ''".
```

```
Start arguments with "+" or "-" to indicate they're numbers (type i4 or i8).

For more details, see the full documentation at

    https://pyrocore.readthedocs.io/

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -q, --quiet           omit informational logging
  -v, --verbose         increase informational logging
  --debug               always show stack-traces for errors
  --cron                run in cron mode (with different logging configuration)
  --config-dir=DIR      configuration directory [~/.pyroscope]
  --config-file=PATH    additional config file(s) to read
  -D KEY=VAL [-D ...], --define=KEY=VAL [-D ...]
                        override configuration attributes
  -r, --repr            show Python pretty-printed response
  -x, --xml             show XML response
  -i, --as-import       execute each argument as a private command using 'import'
```

rTorrent XMLRPC

TODO

XMLRPC Migration

TODO

External Links

- User Mailing List
- The rTorrent and libtorrent projects
- rTorrent Community Wiki and the rTorrent Handbook
- Open HUB
- free(code)
- Bintray
- pyrobase

BitTorrent Protocol

Wikipedia:

- Protocol
- bencode

BitTorrent standards:

- Index of BitTorrent Enhancement Proposals

API Documentation

This is the full pyrocore API documentation, generated from source.

Packages & Modules

pyrocore package

Python Torrent Tools Core Package.

Copyright (c) 2010 The PyroScope Project <pyroscope.project@gmail.com>

pyrocore.**connect**(*config_dir=None*, *optional_config_files=None*, *cron_cfg='cron'*)
Initialize everything for interactive use.

Returns a ready-to-use RtorrentEngine object.

Subpackages

pyrocore.daemon package

Background Daemon Package.

Copyright (c) 2012 The PyroScope Project <pyroscope.project@gmail.com>

Submodules

pyrocore.daemon.webapp module

rTorrent web apps.

Copyright (c) 2013 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.daemon.webapp.**JsonController**(***kwargs*)
Bases: object

Controller for generating JSON data.

ERRORS_LOGGED = set([])

guarded(*func*, **args*, ***kwargs*)
Call a function, return None on errors.

json_charts(*req*)
Return charting data.

json_engine(*req*)
Return torrent engine data.

class pyrocore.daemon.webapp.**Router**
Bases: object

URL router middleware.

See http://docs.webob.org/en/latest/do-it-yourself.html

ROUTES_RE = <_sre.SRE_Pattern object>

add_route(*template*, *controller*, ***kwargs*)
Add a route definition

controller can be either a controller instance, or the name of a callable that will be imported.

classmethod parse_route(*template*)
Parse a route definition, and return the compiled regex that matches it.

class pyrocore.daemon.webapp.**StaticFolders**(*paths*, *fileapp=None*, ***kw*)
Bases: object

An application that serves up the files in a list of given directories.

Non-existent paths are ignored. Pass a *fileapp* factory to change the default file serving app.

pyrocore.daemon.webapp.**make_app**(*httpd_config*)
Factory for the monitoring webapp.

pyrocore.daemon.webapp.**module_test**()
Quick test using...

python -m pyrocore.daemon.webapp

pyrocore.scripts package

Basic Command Line Scripts.

Copyright (c) 2009 The PyroScope Project <pyroscope.project@gmail.com>

Submodules

pyrocore.scripts.base module

Command Line Script Support.

Copyright (c) 2009, 2010 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.base.**PromptDecorator**(*script_obj*)
Bases: object

Decorator for interactive commands.

QUIT_RC = 75

add_options()
Add program options, must be called in script's addOptions().

ask_bool(*question*, *default=True*)
Ask the user for Y)es / N)o / Q)uit.

If "Q" ist entered, this method will exit with RC=3. Else, the user's choice is returned.

Note that the options –non-interactive and –defaults also influence the outcome.

quit()
Exit the program due to user's choices.

class pyrocore.scripts.base.**ScriptBase**
Bases: object

Base class for command line interfaces.

ADDITIONAL_HELP = []

ARGS_HELP = '<log-base>...'

COPYRIGHT = 'Copyright (c) 2009 - 2017 Pyroscope Project'

LOGGING_CFG = '~/.pyroscope/logging.%s.ini'

STD_LOG_LEVEL = 20

VERSION = None

add_bool_option(**args*, ***kwargs*)
Add a boolean option.

@keyword help: Option description.

add_options()
Add program options.

add_value_option(**args*, ***kwargs*)
Add a value option.

@keyword dest: Destination attribute, derived from long option name if not given. @keyword action: How to handle the option. @keyword help: Option description. @keyword default: If given, add this value to the help string.

fatal(*msg*, *exc=None*)
Exit on a fatal error.

get_options()
Get program options.

handle_completion()
Handle shell completion stuff.

help_completion_options()
Return options of this command.

mainloop()
The main loop.

run()
The main program skeleton.

classmethod setup(*cron_cfg='cron'*)
Set up the runtime environment.

class pyrocore.scripts.base.**ScriptBaseWithConfig**
Bases: *pyrocore.scripts.base.ScriptBase*

CLI tool with configuration support.

OPTIONAL_CFG_FILES = []

add_options()
Add configuration options.

get_options()
Get program options.

pyrocore.scripts.chtor module

Metafile Editor.

Copyright (c) 2010 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.chtor.**MetafileChanger**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Change attributes of a bittorrent metafile.

ARGS_HELP = '<metafile>...'

RT_RESUMT_KEYS = ('libtorrent_resume', 'log_callback', 'err_callback', 'rtorrent')

add_options()
Add program options.

mainloop()
The main loop.

pyrocore.scripts.chtor.**replace_fields**(*meta*, *patterns*)
Replace patterns in fields.

pyrocore.scripts.chtor.**run**()
The entry point.

pyrocore.scripts.hashcheck module

Metafile Checker.

Copyright (c) 2011 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.hashcheck.**MetafileChecker**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Check a bittorrent metafile.

ARGS_HELP = '<metafile> [<data-dir-or-file>]'

add_options()
Add program options.

mainloop()
The main loop.

pyrocore.scripts.hashcheck.**run**()
The entry point.

pyrocore.scripts.lstor module

Metafile Lister.

Copyright (c) 2009, 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.lstor.**MetafileLister**
Bases: *pyrocore.scripts.base.ScriptBase*

List contents of a bittorrent metafile.

ARGS_HELP = '<metafile>...'

add_options()
Add program options.

mainloop()
The main loop.

pyrocore.scripts.lstor.**run**()
The entry point.

pyrocore.scripts.mktor module

Metafile Creator.

Copyright (c) 2009, 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.mktor.**MetafileCreator**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Create a bittorrent metafile.

If passed a magnet URI as the only argument, a metafile is created in the directory specified via the configuration value 'magnet_watch', loadable by rTorrent. Which means you can register 'mktor' as a magnet: URL handler in Firefox.

ARGS_HELP = '<dir-or-file> <tracker-url-or-alias>... | <magnet-uri>'

ENTROPY_BITS = 512

add_options()
Add program options.

mainloop()
The main loop.

make_magnet_meta(*magnet_uri*)
Create a magnet-uri torrent.

pyrocore.scripts.mktor.**run**()
The entry point.

pyrocore.scripts.pyroadmin module

Administration Tool.

Copyright (c) 2010 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.pyroadmin.**AdminTool**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Support for administrative tasks.

ARGS_HELP = ''

CONFIG_DIRS = ['log', 'data', 'run', 'htdocs']

OPTIONAL_CFG_FILES = ['torque.ini']

RC_CONTINUATION_THRESHOLD = 55

add_options()
Add program options.

download_resource(*download_url*, *target*, *guard*)
Helper to download and install external resources.

mainloop()
The main loop.

pyrocore.scripts.pyroadmin.**run**()
The entry point.

pyrocore.scripts.pyrotorque module

rTorrent queue manager & daemon.

Copyright (c) 2012 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.pyrotorque.**RtorrentQueueManager**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

rTorrent queue manager & daemon.

ARGS_HELP = ''

OPTIONAL_CFG_FILES = ['torque.ini']

POLL_TIMEOUT = 1.0

add_options()
Add program options.

mainloop()
The main loop.

pyrocore.scripts.pyrotorque.**run**()
The entry point.

pyrocore.scripts.rtcontrol module

rTorrent Control.

Copyright (c) 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.rtcontrol.**FieldStatistics**(*size*)
Bases: object

Collect statistical values for the fields of a search result.

add(*field*, *val*)
Add a sample

average
Calculate average

class pyrocore.scripts.rtcontrol.**RtorrentControl**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Control and inspect rTorrent from the command line.

Filter expressions take the form "<field>=<value>", and all expressions must be met (AND). If a field name is omitted, "name" is assumed. You can also use uppercase OR to build a list of alternative conditions.

For numeric fields, a leading "+" means greater than, a leading "-" means less than. For string fields, the value is a glob pattern (*, ?, [a-z], [!a-z]), or a regex match enclosed by slashes. All string comparisons are case-ignoring. Multiple values separated by a comma indicate several possible choices (OR). "!" in front of a filter value negates it (NOT).

See https://pyrocore.readthedocs.io/en/latest/usage.html#rtcontrol for more.

Examples:

- All 1:1 seeds ratio=+1
- All active torrents xfer=+0
- All seeding torrents up=+0
- Slow torrents down=+0 down=-5k
- Older than 2 weeks completed=+2w
- Big stuff size=+4g
- 1:1 seeds not on NAS ratio=+1 'realpath=!/mnt/*'
- Music kind=flac,mp3

ACTION_MODES = (Bunch(help='start torrent', name='start', options=('–start',)), Bunch(help='stop torrent', method='st

ADDITIONAL_HELP = ['', '', 'Use –help to get a list of all options.', 'Use –help-fields to list all fields and their description.

ARGS_HELP = '<filter>...'

FORMATTER_DEFAULTS = {'now': 1497295940.250278}

IGNORE_OPTIONS = ('0', '1')

PRIO_OPTIONS = ('0', '1', '2', '3')

add_options()
Add program options.

anneal(*mode*, *matches*, *orig_matches*)
Perform post-processing.

Return True when any changes were applied.

emit(*item*, *defaults=None*, *stencil=None*, *to_log=False*, *item_formatter=None*)
Print an item to stdout, or the log on INFO level.

format_item(*item*, *defaults=None*, *stencil=None*)
Format an item.

get_output_fields()
Get field names from output template.

help_completion_fields()
Return valid field names.

mainloop()
The main loop.

show_in_view(*sourceview*, *matches*, *targetname=None*)
Show search result in ncurses view.

validate_output_format(*default_format*)
Prepare output format for later use.

validate_sort_fields()
Take care of sorting.

pyrocore.scripts.rtcontrol.**print_help_fields**()
Print help about fields and field formatters.

pyrocore.scripts.rtcontrol.**run**()
The entry point.

pyrocore.scripts.rtevent module

Rtorrent event handler.

class pyrocore.scripts.rtevent.**RtorrentEventHandler**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Handle rTorrent events.

ARGS_HELP = '<event> <infohash> [<args>...]'

add_options()
Add program options.

mainloop()
The main loop.

pyrocore.scripts.rtevent.**run**()
The entry point.

pyrocore.scripts.rtmv module

Move seeding data.

class pyrocore.scripts.rtmv.**RtorrentMove**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Move data actively seeded in rTorrent.

ARGS_HELP = '<source>... <target>'

PREFETCH_FIELDS = ['hash', 'name', 'size', 'path', 'is_complete']

add_options()
Add program options.

guarded(*call*, **args*)
Catch exceptions thrown by filesystem calls, and don't really execute them in dry-run mode.

mainloop()
The main loop.

resolve_slashed(*path*)
Resolve symlinked directories if they end in a '/', remove trailing '/' otherwise.

pyrocore.scripts.rtmv.**pretty_path**(*path*)
Prettify path for logging.

pyrocore.scripts.rtmv.**run**()
The entry point.

pyrocore.scripts.rtxmlrpc module

Perform raw XMLRPC calls.

Copyright (c) 2010 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.scripts.rtxmlrpc.**RtorrentXmlRpc**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Perform raw rTorrent XMLRPC calls, like "rtxmlrpc throttle.global_up.max_rate".

Start arguments with "+" or "-" to indicate they're numbers (type i4 or i8). Use "[1,2,..." for arrays.

ARGS_HELP = '<method> <args>...'

STD_LOG_LEVEL = 10

add_options()
Add program options.

mainloop()
The main loop.

pyrocore.scripts.rtxmlrpc.**run**()
The entry point.

pyrocore.torrent package

Torrent Backend Engines Package.

Copyright (c) 2010 The PyroScope Project <pyroscope.project@gmail.com>

Submodules

pyrocore.torrent.engine module

Torrent Engine Interface.

Copyright (c) 2009, 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.torrent.engine.**ConstantField**(*valtype*, *name*, *doc*, *accessor=None*, *matcher=None*, *formatter=None*, *engine_name=None*)
Bases: *pyrocore.torrent.engine.ImmutableField*

Read-only download item field with constant value.

class pyrocore.torrent.engine.**DynamicField**(*valtype*, *name*, *doc*, *accessor=None*, *matcher=None*, *formatter=None*, *engine_name=None*)
Bases: *pyrocore.torrent.engine.ImmutableField*

Read-only download item field with dynamic value.

class pyrocore.torrent.engine.**FieldDefinition**(*valtype*, *name*, *doc*, *accessor=None*, *matcher=None*, *formatter=None*, *engine_name=None*)

Bases: object

Download item field.

FIELDS = {u'uploaded': <OnDemandField(<type 'int'>, u'uploaded', u'amount of uploaded data')>, u'tagged': <Dynam

classmethod lookup(*name*)

Try to find field C{name}.

@return: Field descriptions, see C{matching.ConditionParser} for details.

class pyrocore.torrent.engine.**ImmutableField**(*valtype*, *name*, *doc*, *accessor=None*, *matcher=None*, *formatter=None*, *engine_name=None*)

Bases: *pyrocore.torrent.engine.FieldDefinition*

Read-only download item field.

class pyrocore.torrent.engine.**MutableField**(*valtype*, *name*, *doc*, *accessor=None*, *matcher=None*, *formatter=None*, *engine_name=None*)

Bases: *pyrocore.torrent.engine.FieldDefinition*

Writable download item field

class pyrocore.torrent.engine.**OnDemandField**(*valtype*, *name*, *doc*, *accessor=None*, *matcher=None*, *formatter=None*, *engine_name=None*)

Bases: *pyrocore.torrent.engine.DynamicField*

Field that is fetched on first access only.

class pyrocore.torrent.engine.**TorrentEngine**

Bases: object

A torrent backend.

group_by(*fields*, *items=None*)

Returns a dict of lists of items, grouped by the given fields.

`fields` can be a string (one field) or an iterable of field names.

items(*view=None*, *prefetch=None*, *cache=True*)

Get list of download items.

load_config(*namespace=None*, *rcfile=None*)

Load engine configuration file.

log(*msg*)

Log a message in the torrent client.

open()

Open connection.

show(*items*, *view=None*)

Visualize a set of items (search result), and return the view name.

view(*viewname*, *matcher=None*)

Get list of download items.

class pyrocore.torrent.engine.**TorrentProxy**

Bases: object

A single download item.

active
last time a peer was connected

classmethod add_custom_fields(**args*, ***kw*)
Add any custom fields defined in the configuration.

classmethod add_manifold_attribute(*name*)
Register a manifold engine attribute.

@return: field definition object, or None if "name" isn't a manifold attribute.

alias
tracker alias or domain

announce_urls(*default=[]*)
Get a list of all announce URLs.

completed
time download was finished

datapath()
Get an item's data path.

delete()
Remove torrent from client.

directory
directory containing download data

done
completion in percent

down
download rate

fetch(*name*, *engine_name=None*)
Get a field on demand.

"engine_name" is the internal name of the client engine.

files
list of files in this item

flush()
Write volatile data to disk.

fno
number of files in this item

hash
info hash

hash_check()
Hash check a download.

ignore(*flag*)
Set ignore status.

is_active
download active?

is_complete
download complete?

is_ghost
Shining a light on the naming and paths mess:

hash=xxx for i in d.name d.base_filename d.base_path d.directory d.directory_base d.is_multi_file; do

echo -n "$(printf '%20.20s ' $i)"; rtxmlrpc $i $hash

done

Basics:

- d.base_filename is always the basename of d.base_path
- d.directory_base and d.directory are always the same
- d.base_filename and d.base_path are empty on closed items, after a restart, i.e. not too useful (since 0.9.1 or so)

Behaviour of d.directory.set + d.directory_base.set (tested with 0.9.4):

- d.base_path always remains unchanged, and item gets closed
- d.start sets d.base_path if resume data ok
- **single:**
  - d.directory[_base].set → d.name NEVER appended (only in d.base_path)
  - after start, d.base_path := d.directory/d.name
- **multi:**
  - d.directory.set → d.name is appended
  - d.directory_base.set → d.name is NOT appended (i.e. item renamed to last path part)
  - after start, d.base_path := d.directory

Making sense of it (trying to at least):

- d.directory is *always* a directory (thus, single items auto-append d.name in d.base_path and cannot be renamed)
- d.directory_base.set means set path PLUS basename together for a multi item (thus allowing a rename)
- only d.directory.set behaves consistently for single+multi, regarding the end result in d.base_path

is_ignored
ignore commands?

is_multi_file
single- or multi-file download?

is_open
download open?

is_private
private flag set (no DHT/PEX)?

kind
ALL kinds of files in this item (the same as kind_0)

leechtime
time taken from start to completion

loaded
time metafile was loaded

message
current tracker message

metafile
path to torrent file

name

path
path to download data

prio
priority (0=off, 1=low, 2=normal, 3=high)

ratio
normalized ratio (1:1 = 1.0)

realpath
real path to download data

seedtime
total seeding time after completion

sessionfile
path to session file

set_custom(*key*, *value=None*)
Set a custom value. C{key} might have the form "key=value" when value is C{None}.

set_throttle(*name*)
Assign to throttle group.

size
data size

start()
(Re-)start downloading or seeding.

started
time download was FIRST started

stop()
Stop and close download.

stopped
time download was last stopped or paused

tag(*tags*)
Add or remove tags.

tagged
has certain tags? (not related to the 'tagged' view)

throttle
throttle group name (NULL=unlimited, NONE=global)

tracker
first in the list of announce URLs

traits
automatic classification of this item (audio, video, tv, movie, etc.)

up
upload rate

uploaded
amount of uploaded data

views
views this item is attached to

xfer
transfer rate

class pyrocore.torrent.engine.**TorrentView**(*engine*, *viewname*, *matcher=None*)
Bases: object

A view on a subset of torrent items.

items()
Get list of download items.

size()
Total unfiltered size of view.

pyrocore.torrent.engine.**detect_traits**(*item*)
Build traits list from attributes of the passed item. Currently, "kind_51", "name" and "alias" are considered.

See pyrocore.util.traits:dectect_traits for more details.

pyrocore.torrent.engine.**percent**(*floatval*)
Convert float ratio to a percent value.

pyrocore.torrent.engine.**ratio_float**(*intval*)
Convert scaled integer ratio to a normalized float.

pyrocore.torrent.engine.**untyped**(*val*)
A type specifier for fields that does nothing.

pyrocore.torrent.filter module

rTorrent Item Filter Jobs.

Copyright (c) 2012 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.torrent.filter.**ActionRule**(*config=None*)
Bases: *pyrocore.torrent.filter.FilterJobBase*

Perform an action on selected items.

run_filter(*items*)
Perform configured action on filtered items.

class pyrocore.torrent.filter.**FilterJobBase**(*config=None*)
Bases: object

Base class for filter rule jobs.

run()
Filter job callback.

run_filter(*items*)
Perform job on filtered items.

class pyrocore.torrent.filter.**TorrentMirror**(*config=None*)
Bases: *pyrocore.torrent.filter.FilterJobBase*

Mirror selected items via a specified tracker.

run_filter(*items*)
Load filtered items into remote client via tracker / watchdir.

pyrocore.torrent.formatting module

Torrent Item Formatting and Filter Rule Parsing.

class pyrocore.torrent.formatting.**OutputMapping**(*obj*, *defaults=None*)
Bases: *pyrocore.util.algo.AttributeMapping*

Map item fields for displaying them.

classmethod formatter_help()
Return a list of format specifiers and their documentation.

pyrocore.torrent.formatting.**expand_template**(*template*, *namespace*)
Expand the given (preparsed) template. Currently, only Tempita templates are supported.

@param template: The template, in preparsed form, or as a string (which then will be preparsed). @param namespace: Custom namespace that is added to the predefined defaults

and takes precedence over those.

@return: The expanded template. @raise LoggableError: In case of typical errors during template execution.

pyrocore.torrent.formatting.**fmt_delta**(*timestamp*)
Format a UNIX timestamp to a delta (relative to now).

pyrocore.torrent.formatting.**fmt_duration**(*duration*)
Format a duration value in seconds to a readable form.

pyrocore.torrent.formatting.**fmt_iso**(*timestamp*)
Format a UNIX timestamp to an ISO datetime string.

pyrocore.torrent.formatting.**fmt_json**(*val*)
JSON serialization.

pyrocore.torrent.formatting.**fmt_mtime**(*val*)
Modification time of a path.

pyrocore.torrent.formatting.**fmt_pathbase**(*val*)
Base name of a path.

pyrocore.torrent.formatting.**fmt_pathdir**(*val*)
Directory containing the given path.

pyrocore.torrent.formatting.**fmt_pathext**(*val*)
Extension of a path (including the '.').

pyrocore.torrent.formatting.**fmt_pathname**(*val*)
Base name of a path, without its extension.

pyrocore.torrent.formatting.**fmt_pc**(*floatval*)
Scale a ratio value to percent.

pyrocore.torrent.formatting.**fmt_strip**(*val*)
Strip leading and trailing whitespace.

pyrocore.torrent.formatting.**fmt_subst**(*regex*, *subst*)
Replace regex with string.

pyrocore.torrent.formatting.**fmt_sz**(*intval*)
Format a byte sized value.

pyrocore.torrent.formatting.**format_item**(*format_spec*, *item*, *defaults=None*)
Format an item according to the given output format. The format can be gioven as either an interpolation string, or a Tempita template (which has to start with "E{lb}E{lb}"),

@param format_spec: The output format. @param item: The object, which is automatically wrapped for interpolation. @param defaults: Optional default values.

pyrocore.torrent.formatting.**preparse**(*output_format*)
Do any special processing of a template, and return the result.

pyrocore.torrent.formatting.**validate_field_list**(*fields*, *allow_fmt_specs=False*, *name_filter=None*)
Make sure the fields in the given list exist.

@param fields: List of fields (comma-/space-separated if a string). @type fields: list or str @return: validated field names. @rtype: list

pyrocore.torrent.formatting.**validate_sort_fields**(*sort_fields*)
Make sure the fields in the given list exist, and return sorting key.

If field names are prefixed with '-', sort order is reversed for that field (descending).

pyrocore.torrent.jobs module

rTorrent Daemon Jobs.

Copyright (c) 2012 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.torrent.jobs.**EngineStats**(*config=None*)
Bases: object

rTorrent connection statistics logger.

run()
Statistics logger job callback.

class pyrocore.torrent.jobs.**InfluxDBStats**(*config=None*)
Bases: object

Push rTorrent and host statistics to InfluxDB.

run()
Statistics feed job callback.

pyrocore.torrent.jobs.**module_test**()
Quick test using...

python -m pyrocore.torrent.jobs

pyrocore.torrent.queue module

rTorrent Queue Manager.

Copyright (c) 2012 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.torrent.queue.**QueueManager**(*config=None*)
Bases: object

rTorrent queue manager implementation.

VIEWNAME = 'pyrotorque'

run()
Queue manager job callback.

pyrocore.torrent.rtorrent module

rTorrent Proxy.

Copyright (c) 2009, 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.torrent.rtorrent.**CommaLexer**(*text*)
Bases: shlex.shlex

Helper to split argument lists.

class pyrocore.torrent.rtorrent.**RtorrentEngine**
Bases: *pyrocore.torrent.engine.TorrentEngine*

The rTorrent backend proxy.

CONSTANT_FIELDS = set(['tracker_size', 'name', 'is_multi_file', 'is_private', 'size_bytes', 'hash'])

CORE_FIELDS = set(['tied_to_file', 'tracker_size', 'name', 'size_bytes', 'is_private', 'hash', 'is_multi_file', 'complete'])

PREFETCH_FIELDS = set(['up_total', 'down_rate', 'tracker_size', 'name', 'ratio', 'custom=m_alias', 'is_active', 'custom

PYRO2RT_MAPPING = {'custom_tm_completed': 'custom=tm_completed', 'throttle': 'throttle_name', 'prio': 'priority', '

RT2PYRO_MAPPING = {'down_rate': 'down', 'throttle_name': 'throttle', 'complete': 'is_complete', 'custom=m_alias': 'c

RTORRENT_RC_ALIASES = {'network.scgi.open_port': 'scgi_port', 'network.scgi.open_local': 'scgi_local'}

RTORRENT_RC_KEYS = ('scgi_local', 'scgi_port', 'log.execute')

item(*infohash*, *prefetch=None*, *cache=False*)
Fetch a single item by its info hash.

items(*view=None*, *prefetch=None*, *cache=True*)
Get list of download items.

@param view: Name of the view. @param prefetch: OPtional list of field names to fetch initially. @param cache: Cache items for the given view?

load_config(*namespace=None*, *rcfile=None*)
Load file given in "rcfile".

log(*msg*)
Log a message in the torrent client.

open()
Open connection.

show(*items*, *view=None*)
Visualize a set of items (search result), and return the view name.

uptime
rTorrent's uptime.

class pyrocore.torrent.rtorrent.**RtorrentItem**(*engine_*, *fields*)
Bases: *pyrocore.torrent.engine.TorrentProxy*

A single download item.

announce_urls(*default=[]*)
Get a list of all announce URLs. Returns *default* if no trackers are found at all.

as_dict()
Return known fields.

cull(*file_filter=None*, *attrs=None*)
Delete ALL data files and remove torrent from client.

@**param file_filter: Optional callable for selecting a subset of all files.** The callable gets a file item as described for RtorrentItem._get_files and must return True for items eligible for deletion.

@param attrs: Optional list of additional attributes to fetch for a filter.

datapath()
Get an item's data path.

delete()
Remove torrent from client.

execute(*commands*)
Execute XMLRPC command(s).

fetch(*name*, *engine_name=None*)
Get a field on demand.

flush()
Write volatile data to disk.

hash_check()
Hash check a download.

ignore(*flag*)
Set ignore status.

purge()
Delete PARTIAL data files and remove torrent from client.

set_custom(*key*, *value=None*)
Set a custom value. C{key} might have the form "key=value" when value is C{None}.

set_prio(*prio*)
Set priority (0-3).

set_throttle(*name*)
Assign to throttle group.

start()
(Re-)start downloading or seeding.

stop()
Stop and close download.

tag(*tags*)
Add or remove tags.

pyrocore.torrent.rtorrent.**run**()
Module level test.

pyrocore.torrent.watch module

rTorrent Watch Jobs.

class pyrocore.torrent.watch.**MetafileHandler**(*job*, *pathname*)
Bases: object

Handler for loading metafiles into rTorrent.

addinfo()
Add known facts to templating namespace.

handle()
Handle metafile.

load()
Load metafile into client.

parse()
Parse metafile and check pre-conditions.

class pyrocore.torrent.watch.**RemoteWatch**(*config=None*)
Bases: object

rTorrent remote torrent file watch.

run()
Check remote watch target.

class pyrocore.torrent.watch.**TreeWatch**(*config=None*)
Bases: object

rTorrent folder tree watch via inotify.

run()
Regular maintenance and fallback task.

setup()
Set up inotify manager.

See https://github.com/seb-m/pyinotify/.

class pyrocore.torrent.watch.**TreeWatchCommand**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Use tree watcher directly from cmd line, call it like this: python -m pyrocore.torrent.watch <DIR>

If the argument is a file, the templating namespace for that metafile is dumped (for testing and debugging purposes).

ARGS_HELP = '<directory>'

OPTIONAL_CFG_FILES = ['torque.ini']

STD_LOG_LEVEL = 10

classmethod main()
The entry point.

mainloop()
The main loop.

class pyrocore.torrent.watch.**TreeWatchHandler**(*pevent=None*, ***kargs*)
Bases: pyinotify.ProcessEvent

inotify event handler for rTorrent folder tree watch.

See https://github.com/seb-m/pyinotify/.

METAFILE_EXT = ('.torrent', '.load', '.start', '.queue')

handle_path(*event*)
Handle a path-related event.

my_init(***kw*)

process_IN_CLOSE_WRITE(*event*)
File written.

process_IN_MOVED_TO(*event*)
File moved into tree.

process_default(*event*)
Fallback.

pyrocore.ui package

Curses UI helpers and extensions.

Copyright (c) 2017 The PyroScope Project <pyroscope.project@gmail.com>

Submodules

pyrocore.ui.categories module

Category management.

Copyright (c) 2017 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.ui.categories.**CategoryManager**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Rotate through category views.

ARGS_HELP = ''

PREFIX = 'category_'

PREFIX_LEN = 9

add_options()
Add program options.

mainloop()
Manage category views.

pyrocore.ui.categories.**run**()
The entry point.

pyrocore.ui.theming module

Color theme support.

Copyright (c) 2017 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.ui.theming.**ThemeSwitcher**
Bases: *pyrocore.scripts.base.ScriptBaseWithConfig*

Rotate through color themes.

ARGS_HELP = ''

add_options()
Add program options.

mainloop()
Handle theme selection changes, or rotate through selection.

pyrocore.ui.theming.**run**()
The entry point.

pyrocore.util package

Utility Modules.

Copyright (c) 2010 The PyroScope Project <pyroscope.project@gmail.com>

Submodules

pyrocore.util.algo module

Helper Algorithms.

Copyright (c) 2009, 2010 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.util.algo.**AttributeMapping**(*obj*, *defaults=None*)
Bases: object

Wrap an object's dict so that it can be accessed by the mapping protocol.

pyrocore.util.algo.**flatten**(*nested*, *containers=(<type 'list'>, <type 'tuple'>)*)
Flatten a nested list in-place and return it.

pyrocore.util.load_config module

Configuration Loader.

For details, see https://pyrocore.readthedocs.io/en/latest/setup.html

Copyright (c) 2009, 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.util.load_config.**ConfigLoader**(*config_dir=None*)
Bases: object

Populates this module's dictionary with the user-defined configuration values.

CONFIG_INI = 'config.ini'

CONFIG_PY = 'config.py'

INTERPOLATION_ESCAPE = <_sre.SRE_Pattern object>

create(*remove_all_rc_files=False*)
Create default configuration files at either the default location or the given directory.

load(*optional_cfg_files=None*)
Actually load the configuation from either the default location or the given directory.

pyrocore.util.load_config.**validate**(*key*, *val*)
Validate a configuration value.

pyrocore.util.load_config.**walk_resources**(*package_or_requirement*, *resource_name*, *recurse=True*, *base=''*)
Yield paths of files in the given resource directory, all paths start with '/'.

pyrocore.util.matching module

Torrent Item Filters.

class pyrocore.util.matching.**BoolFilter**(*name*, *value*)
Bases: *pyrocore.util.matching.FieldFilter*

Filter boolean values.

match(*item*)
Return True if filter matches item.

pre_filter()
Return rTorrent condition to speed up data transfer.

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**ByteSizeFilter**(*name*, *value*)
Bases: *pyrocore.util.matching.NumericFilterBase*

Filter size and bandwidth values.

UNITS = {'k': 1024, 'b': 1, 'm': 1048576, 'g': 1073741824}

pre_filter()
Return rTorrent condition to speed up data transfer.

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**CompoundFilterAll**
Bases: *pyrocore.util.matching.CompoundFilterBase*

List of filters that must all match (AND).

match(*item*)
Return True if filter matches item.

pre_filter()
Return rTorrent condition to speed up data transfer.

class pyrocore.util.matching.**CompoundFilterAny**
Bases: *pyrocore.util.matching.CompoundFilterBase*

List of filters where at least one must match (OR).

match(*item*)
Return True if filter matches item.

pre_filter()
Return rTorrent condition to speed up data transfer.

class pyrocore.util.matching.**CompoundFilterBase**
Bases: *pyrocore.util.matching.Filter*, list

List of filters.

class pyrocore.util.matching.**ConditionParser**(*lookup*, *default_field=None*, *ident_re='[_A-Za-z][_A-Za-z0-9]*'*)

Bases: object

Filter condition parser.

classmethod AMENABLE(_)
Prefined lookup mode for typeless access to any field name.

COMPARISON_OPS = {'>=': '!-%s', '<>': '!%s', '<=': '!+%s', '~': '/%s/', '!=': '!%s', '<': '-%s', '>': '+%s'}

parse(*conditions*)
Parse filter conditions.

@param conditions: multiple conditions. @type conditions: list or str

class pyrocore.util.matching.**DurationFilter**(*name*, *value*)

Bases: *pyrocore.util.matching.TimeFilter*

Filter durations in seconds.

match(*item*)
Return True if filter matches item.

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**EqualsFilter**(*name*, *value*)

Bases: *pyrocore.util.matching.FieldFilter*

Filter fields equal to the given value.

match(*item*)
Return True if filter matches item.

class pyrocore.util.matching.**FieldFilter**(*name*, *value*)

Bases: *pyrocore.util.matching.Filter*

Base class for all field filters.

PRE_FILTER_FIELDS = {'uploaded': 'd.up.total=', 'custom_tm_completed': 'd.custom=tm_completed', 'tagged': 'd.cu

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**FilesFilter**(*name*, *value*)

Bases: *pyrocore.util.matching.PatternFilter*

Case-insensitive pattern filter on filenames in a torrent.

match(*item*)
Return True if filter matches item.

class pyrocore.util.matching.**Filter**

Bases: object

Base class for all filters.

match(*item*)
Return True if filter matches item.

pre_filter()
Return rTorrent condition to speed up data transfer.

exception pyrocore.util.matching.**FilterError**
Bases: *pyrocore.error.UserError*

(Syntax) error in filter.

class pyrocore.util.matching.**FloatFilter**(*name*, *value*)
Bases: *pyrocore.util.matching.NumericFilterBase*

Filter float values.

FIELD_SCALE = {'ratio': 1000}

pre_filter()
Return rTorrent condition to speed up data transfer.

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**MagicFilter**(*name*, *value*)
Bases: *pyrocore.util.matching.FieldFilter*

Filter that looks at the comparison value and automatically decides what type of filter to use.

match(*item*)
Return True if filter matches item.

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**NegateFilter**(*inner*)
Bases: *pyrocore.util.matching.Filter*

Negate result of another filter (NOT).

match(*item*)
Return True if filter matches item.

pre_filter()
Return rTorrent condition to speed up data transfer.

class pyrocore.util.matching.**NumericFilterBase**(*name*, *value*)
Bases: *pyrocore.util.matching.FieldFilter*

Base class for numerical value filters.

match(*item*)
Return True if filter matches item.

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**PatternFilter**(*name*, *value*)
Bases: *pyrocore.util.matching.FieldFilter*

Case-insensitive pattern filter, either a glob or a /regex/ pattern.

CLEAN_PRE_VAL_RE = <_sre.SRE_Pattern object>

SPLIT_PRE_GLOB_RE = <_sre.SRE_Pattern object>

SPLIT_PRE_VAL_RE = <_sre.SRE_Pattern object>

match(*item*)
Return True if filter matches item.

pre_filter()
Return rTorrent condition to speed up data transfer.

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**TaggedAsFilter**(*name*, *value*)
Bases: *pyrocore.util.matching.FieldFilter*

Case-insensitive tags filter. Tag fields are white-space separated lists of tags.

match(*item*)
Return True if filter matches item.

pre_filter()
Return rTorrent condition to speed up data transfer.

validate()
Validate filter condition (template method).

class pyrocore.util.matching.**TimeFilter**(*name*, *value*)
Bases: *pyrocore.util.matching.NumericFilterBase*

Filter UNIX timestamp values.

TIMEDELTA_RE = <_sre.SRE_Pattern object at 0x34c95f0>

TIMEDELTA_UNITS = {'y': <function <lambda>>, 's': <function <lambda>>, 'd': <function <lambda>>, 'w': <function

pre_filter()
Return rTorrent condition to speed up data transfer.

validate()
Validate filter condition (template method).

validate_time(*duration=False*)
Validate filter condition (template method) for timestamps and durations.

class pyrocore.util.matching.**TimeFilterNotNull**(*name*, *value*)
Bases: *pyrocore.util.matching.TimeFilter*

Filter UNIX timestamp values, ignore unset values unless compared to 0.

validate()
Validate filter condition (template method).

pyrocore.util.matching.**truth**(*val*, *context*)
Convert truth value in "val" to a boolean.

pyrocore.util.matching.**unquote_pre_filter**(*pre_filter*, *_regex=<_sre.SRE_Pattern object>*)
Unquote a pre-filter condition.

pyrocore.util.metafile module

Metafile Support.

Copyright (c) 2009, 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.util.metafile.**MaskingPrettyPrinter**(*indent=1*, *width=80*, *depth=None*, *stream=None*)
Bases: pprint.PrettyPrinter

A PrettyPrinter that masks strings in the object tree.

format (*obj*, *context*, *maxlevels*, *level*)
Mask obj if it looks like an URL, then pass it to the super class.

class pyrocore.util.metafile.**Metafile** (*filename*, *datapath=None*)
Bases: object

A torrent metafile.

IGNORE_GLOB = [u'core', u'CVS', u'.*', u'*~', u'*.swp', u'*.tmp', u'*.bak', u'[Tt]humbs.db', u'[Dd]esktop.ini', u'ehthu

check (*metainfo*, *datapath*, *progress=None*)
Check piece hashes of a metafile against the given datapath.

create (*datapath*, *tracker_urls*, *comment=None*, *root_name=None*, *created_by=None*, *private=False*, *no_date=False*, *progress=None*, *callback=None*)
Create a metafile with the path given on object creation. Returns the last metafile dict that was written (as an object, not bencoded).

datapath
Get a valid datapath, else raise an exception.

listing (*masked=True*)
List torrent info & contents. Returns a list of formatted lines.

walk ()
Generate paths in "self.datapath".

pyrocore.util.metafile.**add_fast_resume** (*meta*, *datapath*)
Add fast resume data to a metafile dict.

pyrocore.util.metafile.**assign_fields** (*meta*, *assignments*)
Takes a list of C{key=value} strings and assigns them to the given metafile. If you want to set nested keys (e.g. "info.source"), you have to use a dot as a separator. For exotic keys *containing* a dot, double that dot ("dotted..key").

Numeric values starting with "+" or "-" are converted to integers.

If just a key name is given (no '='), the field is removed.

pyrocore.util.metafile.**check_info** (*info*)
Validate info dict.

Raise ValueError if validation fails.

pyrocore.util.metafile.**check_meta** (*meta*)
Validate meta dict.

Raise ValueError if validation fails.

pyrocore.util.metafile.**checked_open** (*filename*, *log=None*, *quiet=False*)
Open and validate the given metafile. Optionally provide diagnostics on the passed logger, for invalid metafiles, which then just cause a warning but no exception. "quiet" can supress that warning.

pyrocore.util.metafile.**clean_meta** (*meta*, *including_info=False*, *logger=None*)
Clean meta dict. Optionally log changes using the given logger.

@param logger: If given, a callable accepting a string message. @return: Set of keys removed from C{meta}.

pyrocore.util.metafile.**console_progress** ()
Return a progress indicator for consoles if stdout is a tty.

pyrocore.util.metafile.**data_size** (*metadata*)
Calculate the size of a torrent based on parsed metadata.

pyrocore.util.metafile.**info_hash**(*metadata*)
Return info hash as a string.

pyrocore.util.metafile.**mask_keys**(*announce_url*)
Mask any passkeys (hex sequences) in an announce URL.

pyrocore.util.metafile.**sanitize**(*meta*)
Try to fix common problems, especially transcode non-standard string encodings.

pyrocore.util.osmagic module

Platform Specific Incantations.

Copyright (c) 2011 The PyroScope Project <pyroscope.project@gmail.com>

pyrocore.util.osmagic.**check_process**(*pidfile*)
Read pid file and check process status. Return (running, pid).

pyrocore.util.osmagic.**daemonize**(*pidfile=None*, *logfile=None*, *sync=True*)
Fork the process into the background.

@param pidfile: Optional PID file path. @param sync: Wait for parent process to disappear? @param logfile: Optional name of stdin/stderr log file or stream.

pyrocore.util.osmagic.**guard**(*pidfile*, *guardfile=None*)
Raise an EnvironmentError when the "guardfile" doesn't exist, or the process with the ID found in "pidfile" is still active.

pyrocore.util.pymagic module

Python Utility Functions.

Copyright (c) 2009, 2010 The PyroScope Project <pyroscope.project@gmail.com>

class pyrocore.util.pymagic.**JSONEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)
Bases: json.encoder.JSONEncoder

Custon JSON encoder.

default(*o*)
Support more object types.

pyrocore.util.pymagic.**get_class_logger**(*obj*)
Get a logger specific for the given object's class.

pyrocore.util.pymagic.**get_lazy_logger**(*name*)
Return a logger proxy that is lazily initialized.

This avoids the problems associated with module-level loggers being created early (on import), *before* the logging system is properly initialized.

pyrocore.util.pymagic.**import_name**(*module_spec*, *name=None*)
Import identifier C{name} from module C{module_spec}.

If name is omitted, C{module_spec} must contain the name after the module path, delimited by a colon (like a setuptools entry-point).

@param module_spec: Fully qualified module name, e.g. C{x.y.z}. @param name: Name to import from C{module_spec}. @return: Requested object. @rtype: object

pyrocore.util.stats module

Statistics data.

Copyright (c) 2014 The PyroScope Project <pyroscope.project@gmail.com>

pyrocore.util.stats.**engine_data**(*engine*)
Get important performance data and metadata from rTorrent.

pyrocore.util.traits module

Classification.

Copyright (c) 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

pyrocore.util.traits.**detect_traits**(*name=None*, *alias=None*, *filetype=None*)
Build traits list from passed attributes.

The result is a list of hierarchical classifiers, the top-level consisting of "audio", "movie", "tv", "video", "document", etc. It can be used as a part of completion paths to build directory structures.

pyrocore.util.traits.**get_filetypes**(*filelist*, *path=None*, *size=<function <lambda>>*)
Get a sorted list of file types and their weight in percent from an iterable of file names.

@return: List of weighted file extensions (no '.'), sorted in descending order @rtype: list of (weight, filetype)

pyrocore.util.traits.**name_trait**(*name*, *add_info=False*)
Determine content type from name.

pyrocore.util.xmlrpc module

RTorrent client proxy.

Copyright (c) 2011 The PyroScope Project <pyroscope.project@gmail.com>

exception pyrocore.util.xmlrpc.**HashNotFound**(*msg*, **args*)
Bases: *pyrocore.util.xmlrpc.XmlRpcError*

Non-existing or disappeared hash.

class pyrocore.util.xmlrpc.**RTorrentMethod**(*proxy*, *method_name*)
Bases: object

Collect attribute accesses to build the final method name.

NEEDS_FAKE_TARGET = set(['ui.current_view.set', 'view_filter'])

class pyrocore.util.xmlrpc.**RTorrentProxy**(*url*, *mapping=None*)
Bases: object

Proxy to rTorrent's XMLRPC interface.

Method calls are built from attribute accesses, i.e. you can do something like C{proxy.system.client_version()}.

exception pyrocore.util.xmlrpc.**XmlRpcError**(*msg*, **args*)
Bases: exceptions.Exception

Base class for XMLRPC protocol errors.

Submodules

pyrocore.config module

Configuration.

For details, see https://pyrocore.readthedocs.io/en/latest/setup.html

Copyright (c) 2009, 2010, 2011 The PyroScope Project <pyroscope.project@gmail.com>

`pyrocore.config.`**`lookup_announce_alias`**(*name*)
Get canonical alias name and announce URL list for the given alias.

`pyrocore.config.`**`map_announce2alias`**(*url*)
Get tracker alias for announce URL, and if none is defined, the 2nd level domain.

pyrocore.error module

Exception Classes.

Copyright (c) 2010 The PyroScope Project <pyroscope.project@gmail.com>

exception `pyrocore.error.`**`EngineError`**
Bases: *pyrocore.error.LoggableError*

Connection or other backend error.

exception `pyrocore.error.`**`LoggableError`**
Bases: `exceptions.Exception`

An exception that is intended to be logged instead of passing it to the runtime environment which will likely produce a full stacktrace.

exception `pyrocore.error.`**`NetworkError`**
Bases: *pyrocore.error.LoggableError*

External connection errors.

exception `pyrocore.error.`**`UserError`**
Bases: *pyrocore.error.LoggableError*

Yes, it was your fault!

UML Diagrams

All Classes

Exceptions

rTorrent API

Filter Rules

Scripts

Configuration

Metafile

Tree Watch

Contributing Guidelines

See contribution-guide.org for the basics on contributing to an open source project.

Reporting an Issue, or Requesting a Feature

Any defects and feature requests are managed using GitHub's *issue tracker*. If you never opened an issue on GitHub before, consult the Mastering Issues guide.

Before creating a bug report, please read the Trouble-Shooting Guide and also see `contribution-guide.org`'s Submitting Bugs.

Performing a Release

1. Check for and fix `pylint` violations:

```
paver lint -m
```

2. Verify `debian/changelog` for completeness and the correct version, and bump the release date:

```
dch -r
```

3. Check Travis CI status at https://travis-ci.org/pyroscope/pyrocore

4. Remove 'dev' version tagging from `setup.cfg`, and perform a release check:

```
sed -i -re 's/^(tag_[a-z ]+=)/##\1/' setup.cfg
paver release
```

5. Commit and tag the release:

```
git status  # check all is committed
tag="v$(dpkg-parsechangelog | grep '^Version:' | awk '{print $2}')"
git tag -a "$tag" -m "Release $tag"
```

6. Build the final release and upload it to PyPI:

```
paver dist_clean sdist bdist_wheel
twine upload dist/*.{zip,whl}
```

License

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc., <http://fsf.org/> 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with

modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   > a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
   >
   > b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
   >
   > c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   > a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   >
   > b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable

copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among

countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

    NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

    END OF TERMS AND CONDITIONS

    How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

{description} Copyright (C) {year} {fullname}

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.
>
> {signature of Ty Coon}, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

CHAPTER 2

Indices & Tables

- genindex
- modindex
- search

Python Module Index

p

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X