deluge Documentation

Release 2.0.3

Deluge Team

June 12, 2019

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 Getting started with Deluge

This is a starting point if you are new to Deluge where we will walk you through getting up and running with our BitTorrent client.

1.1.1 Installing Deluge

These are the instructions for installing Deluge. Consider them a work-in-progress and feel free to make suggestions for improvement.

Ubuntu

PPA

Until the stable PPA is updated, the development version of Deluge can be used:

```
sudo add-apt-repository -u ppa:deluge-team/stable
sudo apt install deluge
```

PyPi

To install from Python PyPi, Deluge requires the following system installed packages:

```
sudo apt install python3-pip python3-libtorrent python3-gi python3-gi-cairo gir1.2-
↪gtk-3.0 gir1.2-appindicator3
```

Install with pip:

```
pip install deluge
```

Windows

Unfortuately due to move to GTK3 and Python 3 there is no installer package currently available for Windows.

Intrepid users can install Deluge from seperate packages as detailed in issue #3201.

macOS

There is no `.app` package currently for macOS, but can try Deluge with homebrew.

1. Install Homebrew
2. Open a terminal.
3. Run the following to install required packages:

```
brew install pygobject3 gtk+3 adwaita-icon-theme
brew install libtorrent-rasterbar
```

4. To fix translations:

```
brew link gettext --force
```

5. Install Deluge:

```
pip3 install deluge
```

1.2 How-to guides

A collection of guides covering common issues that might be encountered using Deluge.

1.2.1 How to set Deluge as default torrent application

Check registered mime types

gio mime application/x-bittorrent gio mime x-scheme-handler/magnet

Set Deluge as default mime

gio mime x-scheme-handler/magnet deluge.desktop gio mime application/x-bittorrent deluge.desktop

Troubleshooting

update-mime-database ~/.local/share/mime update-desktop-database ~/.local/share/applications

XDG Check

xdg-mime query default x-scheme-handler/magnet

References

https://help.gnome.org/admin/system-admin-guide/stable/mime-types-custom-user.html.en

1.3 Release notes

A summary of the important changes in major releases of Deluge. For more details see the *changelog* or the git commit log.

1.3.1 Changelog

2.0.3 (2019-06-12)

Gtk UI

- Fix errors running on Wayland (#3265).
- Fix Peers Tab tooltip and context menu errors (#3266).

Web UI

- Fix TypeError in Peers Tab setting country flag.
- Fix reverse proxy header TypeError (#3260).
- Fix request.base 'idna' codec error (#3261).
- Fix unable to change password (#3262).

Extractor plugin

- Fix potential error starting plugin.

Documentation

- Fix macOS install typo.
- Fix Windows install instructions.

2.0.2 (2019-06-08)

Packaging

- Add systemd deluged and deluge-web service files to package tarball (#2034)

Core

- Fix Python 2 compatiblity issue with SimpleNamespace.

2.0.1 (2019-06-07)

Packaging

- Fix setup.py build error without git installed.

2.0.0 (2019-06-06)

Codebase

- Ported to Python 3

Core

- Improved Logging
- Removed the AutoAdd feature on the core. It's now handled with the AutoAdd plugin, which is also shipped with Deluge, and it does a better job and now, it even supports multiple users perfectly.
- Authentication/Permission exceptions are now sent to clients and recreated there to allow acting upon them.
- Updated SSL/TLS Protocol parameters for better security.
- Make the distinction between adding to the session new unmanaged torrents and torrents loaded from state. This will break backwards compatability.
- Pass a copy of an event instead of passing the event arguments to the event handlers. This will break backwards compatability.
- Allow changing ownership of torrents.
- File modifications on the auth file are now detected and when they happen, the file is reloaded. Upon finding an old auth file with an old format, an upgrade to the new format is made, file saved, and reloaded.
- Authentication no longer requires a username/password. If one or both of these is missing, an authentication error will be sent to the client which sould then ask the username/password to the user.
- Implemented sequential downloads.
- Provide information about a torrent's pieces states
- Add Option To Specify Outgoing Connection Interface.
- Fix potential for host_id collision when creating hostlist entries.

Gtk UI

- Ported to GTK3 (3rd-party plugins will need updated).
- Allow changing ownership of torrents.
- Host entries in the Connection Manager UI are now editable.
- Implemented sequential downloads UI handling.
- Add optional pieces bar instead of a regular progress bar in torrent status tab.
- Make torrent opening compatible with all unicode paths.
- Fix magnet association button on Windows.

- Add keyboard shortcuts for changing queue position:
  - Up: Ctrl+Alt+Up
  - Down: Ctrl+Alt+Down
  - Top: Ctrl+Alt+Shift+Up
  - Bottom: Ctrl+Alt+Shift+Down

Web UI

- Server (deluge-web) now daemonizes by default, use '-d' or '–do-not-daemonize' to disable.
- Fixed the '–base' option to work for regular use, not just with reverse proxies.

Blocklist Plugin

- Implemented whitelist support to both core and GTK UI.
- Implemented ip filter cleaning before each update. Restarting the deluge daemon is no longer needed.
- If "check_after_days" is 0(zero), the timer is not started anymore. It would keep updating one call after the other. If the value changed, the timer is now stopped and restarted using the new value.

1.3.2 Deluge 2.0 release notes

Welcome to the latest release of Deluge, a long time in the making!

What's new

Some of the highlights since the last major release.

- Migrated to Python 3 with minimal support retained for Python 2.7.
- Shiny new logo.
- Multi-user support.
- Performance updates to handle thousands of torrents with faster loading times.
- A New Console UI which emulates GTK/Web UIs.
- GTK UI migrated to GTK3 with UI improvements and additions.
- Magnet pre-fetching to allow file selection when adding torrent.
- Fully support libtorrent 1.2 release.
- Language switching support.
- Improved documentation hosted on ReadTheDocs.
- AutoAdd plugin replaces built-in functionality.
- WebUI now daemonizes by default so service scripts will require `-d` option.

Packaging

PyPi

As well as the usual source tarball available for download we now have published Deluge on the PyPi software repository.

- https://pypi.org/project/deluge/

Windows and MacOS

Unfortunately there are no packages yet for Windows or MacOS but they are being worked on. For now alternative install methods are available for testing.

Upgrade considerations

Deluge 2.0 is not compatible with Deluge 1.x clients or daemons so these will require upgrading too. Also third-party Python scripts may not be compatible if they directly connect to the Deluge client and will need migrating.

Always make a backup of your config before a major version upgrade to guard against data loss.

Translations may not be as up-to date so please help out, see *translations* page.

Plugins written for Deluge 1.3 will need upgrading for Deluge 2.0, due to the requirement of Python 3 and GTK3 UI. There is a *update plugin* document to help Plugin authors update their plugins.

1.4 Development & community

Deluge is an open-source project, and relies on its community of users to keep getting better.

1.4.1 Contributing code

Basic requirements and standards

- A new ticket is required for bugs or features. Search the ticket system first, to avoid filing a duplicate.
- Ensure code follows the *syntax and conventions*.
- Code must pass tests. See [testing.md] for information on how to run and write unit tests.
- Commit messages are informative.

Pull request process:

- Fork us on github.
- Clone your repository.
- Create a feature branch for your issue.
- Apply your changes:
  - Add them, and then commit them to your branch.

  - Run the tests until they pass.
  - When you feel you are finished, rebase your commits to ensure a simple and informative commit log.
- Create a pull request on github from your forked repository.
  - Verify that the tests run by Travis-ci are passing.

Syntax and conventions

Code formatters

We use two applications to automatically format the code to save development time. They are both run with pre-commit.

Black

- Python

Prettier

- Javascript
- CSS
- YAML
- Markdown

Common

- Line length: `79` chars.
- Indent: `4 spaces`, no tabs.
- All code should use `'single quotes'`.

Python

We follow PEP8 and Python Code Style which is adhered to with Black formatter.

- Code ‘‘must’’ pass flake8 (w/[https://pypi.python.org/pypi/flake8-quotes flake8-quotes]), [https://pypi.python.org/pypi/isort isort] and [http://www.pylint.org/ Pylint] source code checkers.

  flake8 deluge isort -rc -df deluge pylint deluge pylint deluge/plugins/*/deluge/
- Using the [http://pre-commit.com/ pre-commit] app can aid in picking up issues before creating git commits.

Strings and bytes

To prevent bugs or errors in the code byte strings (`str`) must be decoded to strings (unicode strings, `unicode`) on input and then encoded on output.

Notes:

- PyGTK/GTK+ will accept `str` (utf8 encoded) or `unicode` but will only return `str`. See [http://python-gtk-3-tutorial.readthedocs.org/en/latest/unicode.html GTK+ Unicode] docs.
- There is a `bytearray` type which enables in-place modification of a string. See Python Bytearrays
- Python 3 renames `unicode` to `str` type and byte strings become `bytes` type.

Javascript

- Classes should follow the Ext coding style.
- Class names should be in !CamelCase
- Instances of classes should use camelCase.

Path separators

- All relative path separators used within code should be converted to posix format `/`, so should not contain `\` or `\\`. This is to prevent confusion when dealing with cross-platform clients and servers.

Docstrings

All new docstrings must use Napoleon Google Style with old docstrings eventually converted over.

Google Style example:

```
def func(arg):
   """Function purpose.

   Args:
       arg (type): Description.

   Returns:
      type: Description. If the line is too, long indent next
         line with three spaces.
   """
   return
```

See complete list of [http://www.sphinx-doc.org/en/master/usage/extensions/napoleon.html#docstring-sections supported headers].

Verify that the documentation parses correctly with:

```
python setup.py build_docs
```

1.4.2 Running tests

Deluge testing is implemented using Trial which is Twisted's testing framework and an extension of Python's unittest.

See Twisted website for documentation on Twisted Trial and Writing tests using Trial.

Testing

The tests are located in the source folder under `deluge/tests`. The tests are run from the project root directory. View the unit test coverage at: deluge-torrent.github.io

Trial

Here are some examples that show running all the test through to selecting an individual test.

```
trial deluge
trial deluge.tests.test_client
trial deluge.tests.test_client.ClientTestCase
trial deluge.tests.test_client.ClientTestCase.test_connect_localclient
```

Pytest

```
pytest deluge/tests
pytest deluge/tests/test_client.py
pytest deluge/tests/test_client.py -k test_connect_localclient
```

Plugin

Running the tests for a specific plugin (requires pytest):

```
pytest deluge/plugins/<name-of-plugin>
```

Tox

All the tests for Deluge can be run using tox

See available targets:

```
tox -l
py27
py3
lint
docs
```

Run specific test:

```
tox -e py3
```

Verify code with pre-commit:

```
tox -e lint
```

Travis-ci

Deluge develop branch is tested automatically by Travis. When creating a pull request (PR) on github, Travis will be automatically run the unit tests with the code in the PR.

1.4.3 Documentation contributions

Build

We use Sphinx to create the documentation from source files and docstrings in code.

```
pip install -r docs/requirements.txt
python setup.py build_docs
```

The resulting html files are in `docs/build/html`.

man pages

Located in `docs/man`

1.4.4 Translation contributions

Translators

For translators we have a Launchpad translations account where you can translate the `.po` files.

Marking text for translation

To mark text for translation in Python and ExtJS wrap the string with the function `_()` like this:

```
torrent.set_tracker_status(_("Announce OK"))
```

For GTK the text can also be marked translatable in the `glade/*.ui` files:

```
<property name="label" translatable="yes">Max Upload Speed:</property>
```

For more details see: [http://docs.python.org/library/gettext.html Python Gettext]

Translation process

These are the overall stages in gettext translation:

`Portable Object Template -> Portable Object -> Machine Object`

- The `deluge.pot` is created using `generate_pot.py`.
- Upload `deluge/i18n/deluge.pot` to Launchpad translations.
- Give the translators time to translate the text.

- Download the updated `.po` files from translation site.
- Extract to `deluge/i18n/` and strip the `deluge-` prefix:

```
rename -f 's/^deluge-//' deluge-*.po
```

- The binary `MO` files for each language are generated by `setup.py` using the `msgfmt.py` script.

To enable WebUI to use translations update `gettext.js` by running `gen_gettext.py` script.

Useful applications

- podiff - Compare textual information in two PO files
- gtranslator - GUI po file editor
- Poedit - GUI po file editor

Testing translation

Testing that translations are working correctly can be performed by running Deluge as follows.

Create an `MO` for a single language in the correct sub-directory:

```
mkdir -p deluge/i18n/fr/LC_MESSAGES
python msgfmt.py -o deluge/i18n/fr/LC_MESSAGES/deluge.mo deluge/i18n/fr.po
```

Run Deluge using an alternative language:

```
LANGUAGE=fr deluge
LANGUAGE=ru_RU.UTF-8 deluge
```

Note: If you do not have a particular language installed on your system it will only translate based on the `MO` files for Deluge so some GTK text/button strings will remain in English.

1.5 Development guide

This is a guide to help with developing Deluge.

1.5.1 Developer tutorials

A list of articles to help developers get started with Deluge.

Setup tutorial for Deluge development

The aim of this tutorial is to download the source code and setup an environment to enable development work on Deluge.

Pre-requisites

To build and run the Deluge applications they depends on tools and libraries as listed in DEPENDS.md.

Almost all of the Python packages dependencies will be installed using pip but there are some packages or libraries that are required to be installed to the system.

Ubuntu

Build tools

```
sudo apt install git intltool closure-compiler
pip install --user tox tox-venv
```

Runtime libraries and tools

```
sudo apt install python3-libtorrent python3-geoip python3-dbus  python3-gi \
python3-gi-cairo gir1.2-gtk-3.0 gir1.2-appindicator3 python3-pygame libnotify4 \
librsvg2-common xdg-utils
```

Setup development environment

Clone Deluge git repository

Download the latest git code to local folder.

```
git clone git://deluge-torrent.org/deluge.git
cd deluge
```

Create Python virtual environment

Creation of a [Python virtual environment] keeps the development isolated and easier to maintain and tox has an option to make this process easier:

```
tox -e denv3
```

Activate virtual environment:

```
source .venv/bin/activate
```

Deluge will be installed by tox in *develop* mode which creates links back to source code so that changes will be reflected immediately without repeated installation. Check it is installed with:

```
(.venv) $ deluge --version
deluge-gtk 2.0.0b2.dev149
libtorrent: 1.1.9.0
Python: 2.7.12
OS: Linux Ubuntu 16.04 xenial
```

Setup pre-commit hook

Using pre-commit ensures submitted code is checked for quality when creating git commits.

```
(.venv) $ pre-commit install
```

You are now ready to start playing with the source code.

Reference

- Contributing
- [Key requirements concepts]
- How to install plugins in develop mode?
- How to setup and test translations?
- How to run tests?
- How to create a plugin? –>

1.5.2 How-to guides

A collection of guides for specific issues or to cover more detail than the tutorials.

Web JSON-RPC

How to connect to JSON-RPC with curl

Before continuing make sure deluge-web or webui plugin is running.

Create a curl config

To save a lot of typing and to keep the curl command short we shall create a `curl.cfg` files and put the following contents in it:

```
request = "POST"
compressed
cookie = "cookie_deluge.txt"
cookie-jar = "cookie_deluge.txt"
header = "Content-Type: application/json"
header = "Accept: application/json"
url = "http://localhost:8112/json"
write-out = "\n"
```

To pretty-print the JSON result see: https://stackoverflow.com/q/352098/175584

Login to WebUI

Login to the WebUI and get session cookie:

```
curl -d '{"method": "auth.login", "params": ["deluge"], "id": 1}' -K curl.cfg
```

Result is `true` to signify that login was successful:

```
{
    "error": null,
    "id": 1,
    "result": true
}
```

Check the contents of the cookie file to verify session ID created.

```
cat cookie_deluge.txt
# Netscape HTTP Cookie File
# http://curl.haxx.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

localhost   FALSE   /json   FALSE   1540061203  _session_id <session_id>
```

Check connected to deluged

Use the `web.connected` method to get a boolean response if the webui is connected to a deluged host:

```
curl -d '{"method": "web.connected", "params": [], "id": 1}' -K curl.cfg
```

Result is `false` because WebUI is not yet connected to the daemon:

```
{
    "error": null,
    "id": 1,
    "result": false
}
```

Get list of deluged hosts

Use the `web.get_hosts` method:

```
curl -d '{"method": "web.get_hosts", "params": [], "id": 1}' -K curl.cfg
```

The result contains the `<hostID>` for using in request `params` field.

```
{
    "error": null,
    "id": 1,
    "result": [
        [
            "<hostID>",
            "127.0.0.1",
            58846,
            "localclient"
        ]
    ]
}
```

Get the deluged host status

```
curl -d '{"method": "web.get_host_status", \
"params": ["<hostID>"], "id": 1}' -K curl.cfg
```

The result shows the version and status; *online*, *offline* or *connected*.

```
{
    "error": null,
    "id": 1,
    "result": [
        "<hostID>",
        "Online",
        "2.0.0"
    ]
}
```

Connect to deluged host

To connect to deluged with `<hostID>`:

```
curl -d '{"method": "web.connect", \
"params": ["<hostID>"], "id": 1}' -K curl.cfg
```

The result contains the full list of avaiable host methods:

```
{
    "error": null,
    "id": 1,
    "result": [
        "core.add_torrent_url",
        ...
        "core.upload_plugin"
    ]
}
```

Disconnect from host

```
curl -d '{"method": "web.disconnect", "params": [], "id": 1}' -K curl.cfg
```

A successful result:

```
{
    "error": null,
    "id": 1,
    "result": "Connection was closed cleanly."
}
```

Add a torrent

```
curl -d '{"method": "web.add_torrents", "params": \
[[{"path":"/tmp/ubuntu-12.04.1-desktop-amd64.iso.torrent", \
"options":null}]], "id": 1}' -K curl.cfg
```

Add a magnet URI

```
curl-d '{"method": "core.add_torrent_magnet", \
"params": ["<magnet_uri>", {}], "id": 1}' -K curl.cfg
```

Get list of files for a torrent

```
curl -d '{"method": "web.get_torrent_files", \
"params": ["<torrentid>"], "id": 1}' -K curl.cfg
```

Set a core config option

```
curl -d '{"method": "core.set_config", \
"params":[{"max_upload_slots_global":"200"}], "id": 1}' -K curl.cfg

{"error": null, "result": null, "id": 1}
```

Useful curl config options

For full list of options see man page `man curl` or help `curl --help`:

```
--cookie (-b) # Load cookie file with session id
--cookie-jar (-c) # Save cookie file with session id
--compressed # responses are gzipped
--include (-i) # Include the HTTP header in output (optional)
--header (-H) # HTTP header
--request (-X) # custom request method
--data (-d) # data to send in POST request '{"method": "", "params": [], "id": ""}'
--insecure (-k) # use with self-signed certs https
```

Plugins

How to update a Deluge 1.3 plugin for 2.0

With the new code in Deluge 2.0 there are changes that require authors of existing plugins to update their plugins to work on Deluge 2.0.

The main changes are with Python 3 support and the new GTK3 user interface with the dropping of GTK2. However it is still possible for a 1.3 plugin to be made compatible with 2.0 and this guide aims to helps with that process.

Note that the Deluge 2.0 plugins now use namespace packaging which is not compatible with Deluge 1.3.

Python

Python version matching

Ensure your code is both Python 2.7 and Python >=3.5 compatible.

In `1.3-stable` the plugins that were built with a specfific version of Python could on be loaded if the system Python also matched.

This has change in Deluge 2.0 and it will load any Python version of plugin eggs so compatibility is essential for end-users not to encounter issues.

Six

Use six to assist with compatibility.

Unicode literals

Add this to files to ensure strings and bytes separatation so there are no surprises when running on Python 3.

GTK 3 addition

In order to support both Deluge 1.3 and 2.0 all existing plugin GTK UI files must be copied and then converted to contain only GTK3 code with the old files still using PyGTK e.g.:

```
cp gtkui.py gtk3ui.py
```

Convert from libglade to GtkBuilder

With PyGTK there were two library options for creating the user interface from XML files by default Deluge plugins used libglade but that has been deprecated and removed in GTK3. So the libglade `.glade` files will need converted to GtkBuilder `.ui` files and the Python code updated.

https://developer.gnome.org/gtk2/stable/gtk-migrating-GtkBuilder.html

gtk-builder-convert

Install the `gtk-builder-convert` converter on Ubuntu with:

```
sudo apt install libgtk2.0-dev
```

To convert your GTK run it like so:

```
gtk-builder-convert data/config.glade data/config.ui
```

Glade UI designer for GTK2

The above conversion can be done in Glade UI designer (version <=3.8), ensuring that the minimum Gtk version is set to 2.24 and any deprecated widgets are fixed. The updated file should be saved with file extension `.ui`.

Python code changes

The code needs to replace `gtk.glade` references with `gtk.Builder` and the first step is updating how the files are loaded:

```
- self.glade = gtk.glade.XML(get_resource("config.glade"))
+ self.builder = gtk.Builder()
+ self.builder.add_from_file(get_resource("config.ui"))
```

The next stage is to replace every occurange of these `glade` methods with the `builder` equivalents:

```
glade.signal_autoconnect -> builder.connect_signals
glade.get_widget -> builder.get_object
```

Migrate XML files to GTK3

If you open and save the file it will update with the new requirement header:

```
<!-- Generated with glade 3.18.3 -->
<interface>
  <requires lib="gtk+" version="3.0"/>
```

You can fix deprecated widgets but keep the minimum GTK version to <= 3.10 for desktop compatiblity.

An example of migrating a Deluge plugin to GtkBuilder: AutoAdd GtkBuilder

Gtk import rename

Move from PyGTK to GTK3 using Python bindings.

https://pygobject.readthedocs.io/en/latest/guide/porting.html

```
wget https://gitlab.gnome.org/GNOME/pygobject/raw/master/tools/pygi-convert.sh
cp gtkui.py gtk3ui.py
sh pygi-convert.sh gtk3ui.py
```

```
-import gtk
+from gi.repository import Gtk
```

```
- self.builder = gtk.Builder()
+ self.builder = Gtk.Builder()
```

Deluge GTK3

Imports will need updated from `deluge.ui.gtkui` to `deluge.ui.gtk3`.

PluginBase

```
-from deluge.plugins.pluginbase import GtkPluginBase
+from deluge.plugins.pluginbase import Gtk3PluginBase
-class GtkUI(GtkPluginBase):
+class Gtk3UI(Gtk3PluginBase):
```

1.5.3 Packaging documentation

Release Checklist

Pre-Release

- Update *translation* `po` files from Launchpad account.
- Changelog is updated with relevant commits and release date is added.
- Docs release notes are updated.
- Tag release in git and push upstream e.g.

```
git tag -a deluge-2.0.0 -m "Deluge 2.0.0 Release"
```

Release

- Create source and wheel distributions:

```
python setup.py sdist bdist_wheel
```

- Upload to PyPi (currently only accepts `tar.gz`):

```
twine upload dist/deluge-2.0.0.tar.gz dist/deluge-2.0.0-py3-none-any.whl
```

- Calculate `sha256sum` for each file e.g.

```
cd dist; sha256sum deluge-2.0.0.tar.xz > deluge-2.0.0.tar.xz.sha256
```

- Upload source tarballs and packages to `download.deluge-torrent.org`.
  - Ensure file permissions are global readable: `0644`
  - Sub-directories correspond to *major.minor* version e.g. all `2.0.x` patch releases are stored in `source/2.0`.
  - Change release version in `version` files.
  - Run `trigger-deluge` to sync OSUOSL ftp site.
- Create packages (Ubuntu, Windows, OSX).
  - Ubuntu: https://code.launchpad.net/~deluge-team/+recipe/stable-releases

Post-Release

- Update with version, hashes and release notes:
  - Publish docs on ReadTheDocs.

  - Forum announcement.
  - IRC welcome message.
  - Wikipedia
- Close Trac milestone and add new milestone version for future tickets.
- Ensure all stable branch commits are also applied to development branch.

Launchpad recipe

The launchpad build recipes are for build from source automatically to provide ubuntu packages. They are used to create daily builds of a git repo branch.

Note these don't have the same control as a creating a publishing to PPA.

Main reference: https://help.launchpad.net/Packaging/SourceBuilds/Recipes

Deluge launchpad build recipes

Recipe configuration: https://code.launchpad.net/~deluge-team/+recipes

An example for building the develop branch:

```
# git-build-recipe format 0.4 deb-version 2.0.0.dev{revno}+{git-commit}+{time}
lp:deluge develop
nest-part packaging lp:~calumlind/+git/lp_deluge_deb debian debian develop
```

There are two parts, first to get the source code branch and then the `debian` files for building the package.

Testing and building locally

Create a `deluge.recipe` file with the contents from launchpad and create the build files with `git-build-recipe`:

```
git-build-recipe --allow-fallback-to-native deluge.recipe lp_build
```

Setup pbuilder and build the deluge package:

```
sudo pbuilder build lp_build/deluge*.dsc
```

Alternatively to build using local files with pdebuild:

```
cd lp_build/deluge/deluge
pdebuild
```

This will allow modifying the `debian` files to test changes to `rules` or `control`.

Packaging for Windows

Currently there is no working package for Deluge 2.0. The previous Python freezing application `bbfreeze` is not compatible with Python 3 and the project is no longer maintained.

There are two alternatives `cxfreeze` and `pyinstaller` but neither is trivial with the GTKUI application.

See #3201

1.6 Reference

Technical reference material.

1.6.1 Deluge Web UI

The Deluge web interface is a full featured interface built using the ExtJS framework, running on top of a Twisted webserver.

SSL Configuration

By default the web interface will use the same private key and certificate as the Deluge daemon. You can use a different certificate/key and specify it in the Web UI config, see below for details.

Create SSL Certificate Examples

Sample guide: How to Create a SSL Certificate

Linux

```
openssl req -new -x509 -nodes -out deluge.cert.pem -keyout deluge.key.pem
```

Windows

```
C:\OpenSSL\bin\openssl.exe req -config C:\OpenSSL\bin\openssl.cnf -x509 -days 365 -
↪newkey rsa:1024 -keyout hostkey.pem -nodes -out hostcert.pem
```

Enable Web UI SSL

There are two ways to enable SSL encryption in the webserver:

- Specify in your config (accessible via the Preferences window).
- Use `--ssl` when running the webserver, overriding the configuration value to enable SSL.

Enable Development mode

Add `?dev=true` to the webui url to enable development mode, uses the source js files (if available) rather than compressed versions:

```
http://127.0.0.1:8112/?dev=true
```

1.6.2 Deluge RPC

Message Formats

DelugeRPC is a protocol used for daemon/client communication. There are four types of messages involved in the protocol: RPC Request, RPC Response, RPC Error and Event. All messages are zlib compressed rencoded strings and their data formats are detailed below.

RPC Request

This message is created and sent by the client to the server requesting that a remote method be called. Multiple requests can be bundled in a list.

[[request_id, method, [args], {kwargs}], . . .]

request_id (int) An integer determined by the client that is used in replies from the server. This is used to ensure the client knows which request the data is in response to. Another alternative would be to respond in the same order the requests come in, but this could cause lag if an earlier request takes longer to process.

method (str) The name of the remote method to call. This name can be in dotted format to call other objects or plugins methods.

args (list) The arguments to call the method with.

kwargs (dict) The keyword arguments to call the method with.

RPC Response

This message is created and sent in response to a RPC Request from a client. It will hold the return value of the requested method call. In the case of an error, a RPC Error message will be sent instead.

[message_type, request_id, [return_value]]

message_type (int) This will be a RPC_RESPONSE type id. This is used on the client side to determine what kind of message is being received from the daemon.

request_id (int) The request_id is the same as the one sent by the client in the initial request. It used on the client side to determine what message this is in response to.

return_value (list) The return value of the method call.

RPC Error

This message is created in response to an error generated while processing a RPC Request and will serve as a replacement for a RPC Response message.

[message_type, request_id, exception_type, exception_msg, traceback]

message_type (int) This will be a RPC_ERROR type id.

request_id (int) The request_id is the same as the one sent by the client in the initial request.

exception_type (str) The type of exception raised.

exception_msg (str) The message as to why the exception was raised.

traceback (str) The traceback of the generated exception.

Event

This message is created by the daemon and sent to the clients without being in response to a RPC Request. Events are generally sent for changes in the daemon's state that the clients need to be made aware of.

[message_type, event_name, data]

message_type (int) This will be a RPC_EVENT type id.

event_name (str) This is the name of the event being emitted by the daemon.

data (list) Additional data to be sent with the event. This is dependent upon the event being emitted.

1.6.3 Deluge RPC API

- *Deluge RPC*

class deluge.core.core.**Core**(*listen_interface=None*, *outgoing_interface=None*, *read_only_config_keys=None*)

add_torrent_file(*filename*, *filedump*, *options*)
Adds a torrent file to the session.

Parameters

- **filename** (*str*) – The filename of the torrent.
- **filedump** (*str*) – A base64 encoded string of the torrent file contents.
- **options** (*dict*) – The options to apply to the torrent upon adding.

Returns *str* – The torrent_id or None.

RPC exported method (*Auth level: 5*)

add_torrent_file_async(*filename*, *filedump*, *options*, *save_state=True*)
Adds a torrent file to the session asynchonously.

Parameters

- **filename** (*str*) – The filename of the torrent.
- **filedump** (*str*) – A base64 encoded string of torrent file contents.
- **options** (*dict*) – The options to apply to the torrent upon adding.
- **save_state** (*bool*) – If the state should be saved after adding the file.

Returns *Deferred* – The torrent ID or None.

RPC exported method (*Auth level: 5*)

add_torrent_files(*torrent_files*)
Adds multiple torrent files to the session asynchonously.

Parameters **torrent_files** (*list of tuples*) – Torrent files as tuple of (filename, filedump, options).

Returns Deferred

RPC exported method (*Auth level: 5*)

add_torrent_magnet(*uri*, *options*)
Adds a torrent from a magnet link.

Parameters

- **uri** (*string*) – the magnet link
- **options** (*dict*) – the options to apply to the torrent on add

Returns the torrent_id

Return type string

RPC exported method (*Auth level: 5*)

add_torrent_url(*url*, *options*, *headers=None*)
Adds a torrent from a url. Deluge will attempt to fetch the torrent from url prior to adding it to the session.

Parameters

- **url** (*string*) – the url pointing to the torrent file
- **options** (*dict*) – the options to apply to the torrent on add
- **headers** (*dict*) – any optional headers to send

Returns a Deferred which returns the torrent_id as a str or None

RPC exported method (*Auth level: 5*)

connect_peer(*torrent_id*, *ip*, *port*)
RPC exported method (*Auth level: 5*)

create_account(*username*, *password*, *authlevel*)
RPC exported method (*Auth level: 10*)

create_torrent(*path*, *tracker*, *piece_length*, *comment*, *target*, *webseeds*, *private*, *created_by*, *trackers*, *add_to_session*)
RPC exported method (*Auth level: 5*)

disable_plugin(*plugin*)
RPC exported method (*Auth level: 5*)

enable_plugin(*plugin*)
RPC exported method (*Auth level: 5*)

force_reannounce(*torrent_ids*)
RPC exported method (*Auth level: 5*)

force_recheck(*torrent_ids*)
Forces a data recheck on torrent_ids

RPC exported method (*Auth level: 5*)

get_auth_levels_mappings()
RPC exported method (*Auth level: 0*)

get_available_plugins()
Returns a list of plugins available in the core

RPC exported method (*Auth level: 5*)

get_completion_paths(*args*)
Returns the available path completions for the input value.

RPC exported method (*Auth level: 5*)

get_config()
Get all the preferences as a dictionary

RPC exported method (*Auth level: 5*)

get_config_value(*key*)
Get the config value for key

RPC exported method (*Auth level: 5*)

get_config_values(*keys*)
Get the config values for the entered keys

RPC exported method (*Auth level: 5*)

get_enabled_plugins()
Returns a list of enabled plugins in the core

RPC exported method (*Auth level: 5*)

get_external_ip()
Returns the external ip address recieved from libtorrent.

RPC exported method (*Auth level: 5*)

get_filter_tree(*show_zero_hits=True*, *hide_cat=None*)
returns {field: [(value,count)] } for use in sidebar(s)

RPC exported method (*Auth level: 5*)

get_free_space(*path=None*)
Returns the number of free bytes at path

Parameters **path** (*string*) – the path to check free space at, if None, use the default download location

Returns the number of free bytes at path

Return type int

Raises **InvalidPathError** – if the path is invalid

RPC exported method (*Auth level: 5*)

get_known_accounts()
RPC exported method (*Auth level: 10*)

get_libtorrent_version()
Returns the libtorrent version.

Returns the version

Return type string

RPC exported method (*Auth level: 5*)

get_listen_port()
Returns the active listen port

RPC exported method (*Auth level: 5*)

get_path_size(*path*)

Returns the size of the file or folder 'path' and -1 if the path is unaccessible (non-existent or insufficient privs)

RPC exported method (*Auth level: 5*)

get_proxy()
Returns the proxy settings

Returns *dict* – Contains proxy settings.

Notes

Proxy type names: 0: None, 1: Socks4, 2: Socks5, 3: Socks5 w Auth, 4: HTTP, 5: HTTP w Auth, 6: I2P

RPC exported method (*Auth level: 5*)

get_session_state()
Returns a list of torrent_ids in the session.

RPC exported method (*Auth level: 5*)

get_session_status(*keys*)
Gets the session status values for 'keys', these keys are taking from libtorrent's session status.

See: http://www.rasterbar.com/products/libtorrent/manual.html#status

Parameters **keys** (*list*) – the keys for which we want values

Returns a dictionary of {key: value, ...}

Return type dict

RPC exported method (*Auth level: 5*)

get_torrent_status(*torrent_id*, *keys*, *diff=False*)
RPC exported method (*Auth level: 5*)

get_torrents_status(*filter_dict*, *keys*, *diff=False*)
returns all torrents , optionally filtered by filter_dict.

RPC exported method (*Auth level: 5*)

glob(*path*)
RPC exported method (*Auth level: 5*)

is_session_paused()
Returns the activity of the session

RPC exported method (*Auth level: 5*)

move_storage(*torrent_ids*, *dest*)
RPC exported method (*Auth level: 5*)

pause_session()
Pause the entire session

RPC exported method (*Auth level: 5*)

pause_torrent(*torrent_id*)
Pauses a torrent

RPC exported method (*Auth level: 5*)

pause_torrents(*torrent_ids=None*)
Pauses a list of torrents

RPC exported method (*Auth level: 5*)

prefetch_magnet_metadata(*magnet*, *timeout=30*)
Download magnet metadata without adding to Deluge session.

Used by UIs to get magnet files for selection before adding to session.

Parameters

- **magnet** (*str*) – The magnet uri.

- **timeout** (*int*) – Number of seconds to wait before cancelling request.

Returns *Deferred* – A tuple of (torrent_id (str), metadata (dict)) for the magnet.

RPC exported method (*Auth level: 5*)

queue_bottom(*torrent_ids*)
RPC exported method (*Auth level: 5*)

queue_down(*torrent_ids*)
RPC exported method (*Auth level: 5*)

queue_top(*torrent_ids*)
RPC exported method (*Auth level: 5*)

queue_up(*torrent_ids*)
RPC exported method (*Auth level: 5*)

remove_account(*username*)
RPC exported method (*Auth level: 10*)

remove_torrent(*torrent_id*, *remove_data*)
Removes a single torrent from the session.

Parameters

- **torrent_id** (*str*) – The torrent ID to remove.
- **remove_data** (*bool*) – If True, also remove the downloaded data.

Returns *bool* – True if removed successfully.

Raises **InvalidTorrentError** – If the torrent ID does not exist in the session.

RPC exported method (*Auth level: 5*)

remove_torrents(*torrent_ids*, *remove_data*)
Remove multiple torrents from the session.

Parameters

- **torrent_ids** (*list*) – The torrent IDs to remove.
- **remove_data** (*bool*) – If True, also remove the downloaded data.

Returns

list –

An empty list if no errors occurred otherwise the list contains tuples of strings, a torrent ID and an error message. For example:

[('<torrent_id>', 'Error removing torrent')]

RPC exported method (*Auth level: 5*)

rename_files(*torrent_id*, *filenames*)
Rename files in torrent_id. Since this is an asynchronous operation by libtorrent, watch for the TorrentFileRenamedEvent to know when the files have been renamed.

Parameters

- **torrent_id** (*string*) – the torrent_id to rename files
- **filenames** (*((index, filename), ..)*) – a list of index, filename pairs

Raises **InvalidTorrentError** – if torrent_id is invalid

RPC exported method (*Auth level: 5*)

rename_folder(*torrent_id*, *folder*, *new_folder*)
Renames the 'folder' to 'new_folder' in 'torrent_id'. Watch for the TorrentFolderRenamedEvent which is emitted when the folder has been renamed successfully.

Parameters

- **torrent_id** (*string*) – the torrent to rename folder in
- **folder** (*string*) – the folder to rename
- **new_folder** (*string*) – the new folder name

Raises **InvalidTorrentError** – if the torrent_id is invalid

RPC exported method (*Auth level: 5*)

rescan_plugins()
Rescans the plugin folders for new plugins

RPC exported method (*Auth level: 5*)

resume_session()
Resume the entire session

RPC exported method (*Auth level: 5*)

resume_torrent(*torrent_id*)
Resumes a torrent

RPC exported method (*Auth level: 5*)

resume_torrents(*torrent_ids=None*)
Resumes a list of torrents

RPC exported method (*Auth level: 5*)

set_config(*config*)
Set the config with values from dictionary

RPC exported method (*Auth level: 5*)

set_torrent_auto_managed(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'auto_managed'

RPC exported method (*Auth level: 5*)

set_torrent_file_priorities(*torrent_id*, *priorities*)
Deprecated: Use set_torrent_options with 'file_priorities'

RPC exported method (*Auth level: 5*)

set_torrent_max_connections(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'max_connections'

RPC exported method (*Auth level: 5*)

set_torrent_max_download_speed(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'max_download_speed'

RPC exported method (*Auth level: 5*)

set_torrent_max_upload_slots(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'max_upload_slots'

RPC exported method (*Auth level: 5*)

set_torrent_max_upload_speed(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'max_upload_speed'

RPC exported method (*Auth level: 5*)

set_torrent_move_completed(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'move_completed'

RPC exported method (*Auth level: 5*)

set_torrent_move_completed_path(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'move_completed_path'

RPC exported method (*Auth level: 5*)

set_torrent_options(*torrent_ids*, *options*)
Sets the torrent options for torrent_ids

Parameters

- **torrent_ids** (*list*) – A list of torrent_ids to set the options for.
- **options** (*dict*) – A dict of torrent options to set. See torrent.TorrentOptions class for valid keys.

RPC exported method (*Auth level: 5*)

set_torrent_prioritize_first_last(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'prioritize_first_last'

RPC exported method (*Auth level: 5*)

set_torrent_remove_at_ratio(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'remove_at_ratio'

RPC exported method (*Auth level: 5*)

set_torrent_stop_at_ratio(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'stop_at_ratio'

RPC exported method (*Auth level: 5*)

set_torrent_stop_ratio(*torrent_id*, *value*)
Deprecated: Use set_torrent_options with 'stop_ratio'

RPC exported method (*Auth level: 5*)

set_torrent_trackers(*torrent_id*, *trackers*)
Sets a torrents tracker list. trackers will be [{"url", "tier"}]

RPC exported method (*Auth level: 5*)

test_listen_port()
Checks if the active port is open

Returns True if the port is open, False if not

Return type bool

RPC exported method (*Auth level: 5*)

update_account(*username*, *password*, *authlevel*)
RPC exported method (*Auth level: 10*)

upload_plugin(*filename*, *filedump*)

This method is used to upload new plugins to the daemon. It is used when connecting to the daemon remotely and installing a new plugin on the client side. 'plugin_data' is a xmlrpc.Binary object of the file data, ie, plugin_file.read()

RPC exported method (*Auth level: 5*)

class deluge.core.daemon.**Daemon**(*listen_interface=None*, *outgoing_interface=None*, *interface=None*, *port=None*, *standalone=False*, *read_only_config_keys=None*)

The Deluge Daemon class

__init__(*listen_interface=None*, *outgoing_interface=None*, *interface=None*, *port=None*, *standalone=False*, *read_only_config_keys=None*)

Parameters

- **listen_interface** (*str, optional*) – The IP address to listen to BitTorrent connections on.
- **outgoing_interface** (*str, optional*) – The network interface name or IP address to open outgoing BitTorrent connections on.
- **interface** (*str, optional*) – The IP address the daemon will listen for UI connections on.
- **port** (*int, optional*) – The port the daemon will listen for UI connections on.
- **standalone** (*bool, optional*) – If True the client is in Standalone mode otherwise, if False, start the daemon as separate process.
- **read_only_config_keys** (*list of str, optional*) – A list of config keys that will not be altered by core.set_config() RPC method.

authorized_call(*rpc*)

Determines if session auth_level is authorized to call RPC.

Parameters **rpc** (*str*) – A RPC, e.g. core.get_torrents_status

Returns *bool* – True if authorized to call RPC, otherwise False.

RPC exported method (*Auth level: 1*)

get_method_list()

Returns a list of the exported methods.

RPC exported method (*Auth level: 5*)

get_version()

Returns the daemon version

RPC exported method (*Auth level: 5*)

shutdown(**args*, ***kwargs*)

RPC exported method (*Auth level: 5*)

1.6.4 Deluge Web JSON-RPC API

- Spec: JSON-RPC v1
- URL: `/json`
- *Deluge RPC API*

class deluge.ui.web.json_api.**WebApi**

The component that implements all the methods required for managing the web interface. The complete web json interface also exposes all the methods available from the core RPC.

add_host(*host*, *port*, *username=''*, *password=''*)

Adds a host to the list.

Parameters

- **host** (*str*) – The IP or hostname of the deluge daemon.
- **port** (*int*) – The port of the deluge daemon.
- **username** (*str*) – The username to login to the daemon with.
- **password** (*str*) – The password to login to the daemon with.

Returns

tuple –

A tuple of (bool, str). If True will contain the host_id, otherwise if False will contain the error message.

add_torrents(*torrents*)

Add torrents by file

Parameters **torrents** (*list*) – A list of dictionaries containing the torrent path and torrent options to add with.

```
json_api.web.add_torrents([{
    "path": "/tmp/deluge-web/some-torrent-file.torrent",
    "options": {"download_location": "/home/deluge/"}
}])
```

connect(*host_id*)

Connect the web client to a daemon.

Parameters **host_id** (*str*) – The id of the daemon in the host list.

Returns *Deferred* – List of methods the daemon supports.

connected()

The current connection state.

Returns True if the client is connected

Return type booleon

deregister_event_listener(*event*)

Remove an event listener from the event queue.

Parameters **event** (*string*) – The event name

disconnect()

Disconnect the web interface from the connected daemon.

download_torrent_from_url(*url*, *cookie=None*)

Download a torrent file from a url to a temporary directory.

Parameters **url** (*string*) – the url of the torrent

Returns the temporary file name of the torrent file

Return type string

edit_host(*host_id*, *host*, *port*, *username=''*, *password=''*)
Edit host details in the hostlist.

Parameters

- **host_id** (*str*) – The host identifying hash.
- **host** (*str*) – The IP or hostname of the deluge daemon.
- **port** (*int*) – The port of the deluge daemon.
- **username** (*str*) – The username to login to the daemon with.
- **password** (*str*) – The password to login to the daemon with.

Returns *bool* – True if succesful, False otherwise.

get_config()
Get the configuration dictionary for the web interface.

Return type dictionary

Returns the configuration

get_events()
Retrieve the pending events for the session.

get_host_status(*host_id*)
Returns the current status for the specified host.

Parameters **host_id** (*string*) – the hash id of the host

get_hosts()
Return the hosts in the hostlist.

get_magnet_info(*uri*)
Parse a magnet URI for hash and name.

get_plugin_info(*name*)
Get the details for a plugin.

get_plugin_resources(*name*)
Get the resource data files for a plugin.

get_plugins()
All available and enabled plugins within WebUI.

Note: This does not represent all plugins from deluge.client.core.

Returns *dict* – A dict containing 'available_plugins' and 'enabled_plugins' lists.

get_torrent_files(*torrent_id*)
Gets the files for a torrent in tree format

Parameters **torrent_id** (*string*) – the id of the torrent to retrieve.

Returns The torrents files in a tree

Return type dictionary

get_torrent_info(*filename*)
Return information about a torrent on the filesystem.

Parameters **filename** (*string*) – the path to the torrent

Returns information about the torrent:

```
{
    "name": the torrent name,
    "files_tree": the files the torrent contains,
    "info_hash" the torrents info_hash
}
```

Return type dictionary

get_torrent_status(*torrent_id*, *keys*)
Get the status for a torrent, filtered by status keys.

register_event_listener(*event*)
Add a listener to the event queue.

Parameters **event** (*string*) – The event name

remove_host(*host_id*)
Removes a host from the hostlist.

Parameters **host_id** (*str*) – The host identifying hash.

Returns *bool* – True if succesful, False otherwise.

set_config(*config*)
Sets the configuration dictionary for the web interface.

Parameters **config** (*dictionary*) – The configuration options to update

start_daemon(*port*)
Starts a local daemon.

stop_daemon(*host_id*)
Stops a running daemon.

Parameters **host_id** (*string*) – the hash id of the host

update_ui(*keys*, *filter_dict*)
Gather the information required for updating the web interface.

Parameters

- **keys** (*list*) – the information about the torrents to gather
- **filter_dict** (*dictionary*) – the filters to apply when selecting torrents.

Returns The torrent and ui information.

Return type dictionary

upload_plugin(*filename*, *path*)
Upload a plugin to config.

class deluge.ui.web.json_api.**WebUtils**
Utility functions for the webui that do not fit in the WebApi.

get_languages()
Get the available translated languages

Returns *list* – of tuples [(lang-id, language-name), ...]

- genindex
- modindex