Freeseer Documentation

Release 3.0.0

Free and Open Source Software Learning Centre

October 18, 2014

Contents

Welcome to the official documentation for Freeseer, screencasting software for recording and streaming presentations.

CHAPTER 1

About Freeseer

Software with a Purpose.

Development of Freeseer is led by the Free and Open Source Software Learning Centre (FOSSLC), a non-profit whose vision is to improve lives with open source. FOSSLC also specializes in technology and know-how to record conferences with excellent quality.
From large conferences with hundreds of talks to presentations in classrooms, anyone can use Freeseer to capture talks.

Freeseer (pronounced *free-see-ar*) is a free, open source, cross-platform application that captures or streams your desktop. It's designed for capturing presentations, and has been succesfully used at many open source conferences to record hundreds of talks (which can be seen at fosslc.org). Though designed for capturing presentations, it can also be used to capture demos, training materials, lectures, and other videos.

Freeseer is written in Python, uses Qt4 for its GUI, and Gstreamer for video/audio processing. Freeseer is based on open standards and supports royalty free audio and video codecs.

Freeseer's source code is licensed under the GNU General Public License and is available on GitHub.

1.1 Who We Are

We're an open source community of developers, writers, designers, bloggers, students, and open source enthusiasts whose goal is to make meaningful software with world-class documentation (inspired by ThinkUp and GitHub).

You're reading the very beginning of that effort. This documentation is an incomplete, work-in-progress. Please join us and help *fill in the gaps*.

1.2 Why We Exist

FOSSLC has been recording conferences since 2008, but they weren't quite happy with their recording solutions. There were issues such as costs, ownership, portability, and simplicity. There had to be a better way.

Freeseer began in 2009 as an in-house solution for FOSSLC to record conference talks. Many people have contributed since then and development is still on-going.

See also:

Freeseer's history

1.3 Expectations Around Sharing Freeseer

The Freeseer project includes code, documentation, and more, written by many different people. All Freeseer contributors retain copyright on their contributions, but agree to release it under the same license as Freeseer. If you are unable or unwilling to contribute a patch under the GPL version 3 or later, do not submit a patch.

Freeseer is copyrighted by FOSSLC and various contributors (listed above).

1.4 Who Freeseer Is For

Freeseer can be used by everyone but is aimed at organizations and personalities who are actively involved in conferences or events and have to record many presentations and talks in a short period of time.

Freeseer will be most useful for:

Presenters who want to record their own talks using a simple application that has virtually no learning curve, and covers all their basic needs.

Conference Staff who want easier ways to manage the recording of talks, and record talks with top audio and video quality.

Instructors like professors, bloggers, or consultants who want an easy way to record their lectures, tutorials, or training material to later share with others.

1.5 What Freeseer Can Do

Introduction to Freeseer v2.5 (outdated video, but still demonstrates the essence of Freeseer)

At its heart, Freeseer is a screencasting tool. Freeseer helps you record your desktop and audio, whether you're recording hundreds of talks at a conference or making a how-to video at home.

Using Freeseer, you can:

Record Talks: The recording interface is designed to be simple so you can focus on recording. You can also pause and resume recordings. Did we mention that the audio and video quality are great?

Manage Talks: Freeseer allows you to add a list of talks before you record them. Then when you're ready to record, you simply select the talk from the list and start recording. This allows all talks to be added from beforehand, so when it's time to present you can focus on recording and not worry about distractions (such as filling in the talk info). You can see why Freeseer is great for recording many talks right after another.

The Talk Editor tool allows you to manually add talks, load them via RSS (a lot of conferences have an event schedule online), or load them from a CSV file. You can also edit or remove talks. Talks are neatly displayed in a table and are sortable by title, speaker, event, room, and more.

Configure Freeseer: Configure your audio and video input, output, save location, and more. With Freeseer's plugin system, developers can easily write their own plugin to add a new feature. Plugins are configurable via the config tool.

1.6 Recording Events

1.6.1 Recording Events

Freeseer's primary use case is recording large events such as conferences. You can run Freeseer locally and have the presenter record their desktop, but a much more useful configuration is when Freeseer is used from a dedicated laptop to record VGA output from the presenter's laptop.

Equipment Needed

- A working installation of Freeseer on a dedicated computer for recording
- VGA Capture Device
  - We recommend an Epiphan frame grabber
  - We use the VGA2USB device (the red one) as it's the cheapest and does a decent job
- Wireless Microphones
  - We use Sennheiser EW100 G2
- USB extension cable (optional)

Setup

Preparation (Avoiding Common Errors)

- Presentations that contain lots of animations or quick movements (e.g. live demos) are best recorded with a higher-end frame grabber since they output higher FPS.
- Use the Talk Editor tool to enter all the talks and their info beforehand.
- Do a test recording with your completed setup to make sure everything works.
- Verify that the presenters don't have their mics muted.
- Have the presenter disable their screen saver and power saving mode. [1]
  - Linux users can use the caffeine app:

```
sudo add-apt-repository ppa:caffeine-developers/ppa
sudo apt-get update
sudo apt-get install caffeine
```

Troubleshooting

- If you only see the red indicator light on the VGA2USB device (the green one isn't showing), and everything has worked previously, then it's possible that the USB cable connecting the device is damaged.
- When a new presenter's laptop is plugged-in and you get a noisy signal the first time you hit record, then try stopping and restarting the recording.

[1] If the presenter's laptop goes into power saving mode, then the vga2usb device loses the signal and you will have to pause and restart the recording.

Hosting the Videos

The Free and Open Source Software Learning Centre (FOSSLC) can place your videos on fosslc.org for free if the talks are open source related. FOSSLC will promote them on the site and host them on YouTube. The site is somewhat popular and gets quite a few hits from various communities. If you decide to use the services of FOSSLC, please email fosslc@gmail.com. Of course you can always use your own hosting solution.

CHAPTER 2

Quick-Start Guide

Freeseer is a free and open source screencasting application, primarily developed for capturing and streaming computer-aided presentations at conferences.

It's been successfully used to capture presentations, demos, training material, and other videos. It's capable of handling large conferences with many talks in various rooms.

With Freeseer, you can record video from external sources such as FireWire and USB (e.g. webcam or another computer's screen via VGA output [1]).

Freeseer is written in Python, uses Qt4 for its GUI, and Gstreamer for video/audio processing. And it's based on open standards so it supports royalty free audio and video codecs.

Read our history to find out why Freeseer was created.

2.1 Installing Freeseer from a package

Warning: You should only install Freeseer from a package if you plan to use it as an end-user. If you want to contribute to the project, **do not** *install Freeseer from a package* (if you already have, you'll need to uninstall it). Instead, follow the instructions for *installing Freeseer for development*.

2.1.1 Arch Linux

Freeseer is available in AUR: https://aur.archlinux.org/packages/freeseer-git/

Or install with yaourt:

```
yaourt -S freeseer-git
```

2.1.2 Gentoo Linux

Freeseer is available in PaddyMac's portage overlay: https://github.com/PaddyMac/overlay

After adding this overlay:

```
emerge -av freeseer
```

[1] *Requires a VGA capture device*, also known as a frame grabber.

2.1.3 OpenSUSE

Freeseer is available in the OpenSUSE repository:

```
zypper install freeseer
```

2.1.4 Python Package Index

Freeseer can also be installed with pip:

```
pip install freeseer
```

2.2 Installing Freeseer for Development

If you plan on contributing to Freeseer's development, you'll have to run Freeseer from source.

1. Uninstall any previously installed instances of Freeseer
2. Obtain the source code by *forking and cloning the project*
3. Install the required dependencies
    - Follow the below instructions for your operating system and *use pip to install additional dependencies*

Now you're ready to run Freeseer from the command line. There are two ways to do this.

1. Run the Freeseer module as a script:

    ```
    cd freeseer/src/
    python -m freeseer
    ```

    You'll have to repeat these steps whenever you want to run Freeseer.

2. Install Freeseer in editable mode:

    ```
    cd freeseer/src/
    pip install -e .
    ```

    After this one-time install, you can now run `freeseer` from anywhere.

2.2.1 Dependencies

- Git
- Python 2.7+
- sqlite3
- gstreamer0.10-python (pygst)
- PyQT development tools

Debian and Ubuntu Linux

```
sudo apt-get update
sudo apt-get install -y build-essential git wget python2.7-dev python-gst0.10 python-gst0.10-dev \
  qt4-qmake python-qt4 python-qt4-dev python-qt4-sql pyqt4-dev-tools libqt4-dev libqt4-sql libqt4-sql
  gstreamer0.10-plugins-good gstreamer0.10-plugins-base gstreamer0.10-pulseaudio gstreamer0.10-alsa
```

Fedora Linux

```
sudo yum install git PyQt4-devel gstreamer-python sphinx python-sphinx
```

Warning: This list may be incomplete. Please `let us know` if you notice any missing packages.

Windows

Note: x86 version recommended whenever there is a choice.

- python 2.7.* x86
- setuptools-0.6c11.win32-py2.7
- GStreamer-WinBuilds-GPL-x86-Beta04-0.10.7
- **GStreamer-WinBuilds-SDK-GPL-x86-Beta04-0.10.7**
  - If you encounter the error "`ImportError: DLL load failed`" when attempting to run freeseer, copy the contents of `<GStreamer_dir>\<version>\sdk\bindings\python\v2.7\lib` to `<GStreamer_dir>\<version>\lib`, and delete `<GStreamer_dir>\<version>\lib\gstreamer-0.1`
- PyQt-Py2.7-x86-gpl-4.8.5-1
- **PyGTK py2.7 all-in-one**
  - Windows 32-bit packages are recommended because pygtk-all-in-one package does not have a 64-bit installer.
  - Add the following paths to your PATH variable : `'C:\Python27;C:\Python27\Lib\site-packages\PyQt4`

PyPI Packages

You'll need to get some packages from the Python Package Index (PyPI).

You can install pip by securely downloading get-pip.py and executing it with administrator access:

```
wget -q https://raw.github.com/pypa/pip/master/contrib/get-pip.py -O- | sudo python
```

If you already have pip, first upgrade it to the latest version:

```
pip install --upgrade pip
```

Install the remaining packages. You may need administrator access.

On Linux:

```
pip install -Ur dev_requirements.txt
```

On Windows:

```
pip install -Ur windows_requirements.txt
```

2.3 Running Freeseer

Once you've installed Freeseer, you can run the various tools:

```
freeseer         # Recording UI (default when no arguments supplied)
freeseer record  # Recording UI
freeseer talk    # Talk Editor UI
freeseer config  # Configuration UI
```

You can view usage with the `-h` or `--help` option:

```
freeseer -h          # General usage
freeseer record -h   # Recording usage
freeseer talk -h     # Talk Editor usage
freeseer config -h   # Config usage
```

Note: If you're going to hack on Freeseer, you'll need to run it from source. Go into the `src/` directory and run it like:

```
python -m freeseer
python -m freeseer record
python -m freeseer talk
python -m freeseer config
```

2.4 Issue tracker

Found an issue? Open an issue on GitHub!

https://github.com/Freeseer/freeseer/issues

2.5 IRC channel

Drop by our #freeseer channel on irc.freenode.net to chat with us.

2.6 Mailing list

We have a mailing list that's also a discussion group.

http://groups.google.com/group/freeseer

Once you've joined the group, you can email subscribers at freeseer@googlegroups.com.

2.7 Authors

- Andrew Ross

- Thanh Ha

And many student contributors from Google Summer of Code, Fedora Summer Coding, and Undergraduate Capstone Open Source Projects.

2.8 Copyright and license

CHAPTER 3

User Guide

Freeseer is an extensible platform with core features, and features added by plugins. This user guide aims to cover all Freeseer's features offered by the core applications and the default plugins which are distributed with it.

Our goal is for Freeseer to become so straightforward to use that this user guide should be seldom referenced, but we're not there yet.

If you ever have to refer to this guide, then please `let us know` how we can improve Freeseer!

3.1 Record

3.1.1 Record Interface

The Freeseer recording interface provides the main window for recording a video. This interface is designed with features that enable you to record talks at a conference very easily.

Select a Talk

Typically the first task is to **find the talk you want to record** out of all the possible talks that are stored in the Freeseer database. You can **filter the list of talks** by selecting an **Event**, **Room**, and then a **Date** from the dropdown lists.

Note: When you record a talk, the output file will contain some of the associated talk data as metadata.

Tip: You can enter new talks or modify existing talks using the *Talk Editor* tool.

Conferences typically release information about their talks prior to the event. It's good practice to enter such data (e.g. title, speaker, event, etc.) for every talk you plan to record, prior to attending the conference rather than last minute.

Audio Feedback

At the bottom right of the interface there is a checkbox with a headphones icon. Checking this will enable audio feedback, which will play any sounds being recorded back to your speakers. This is useful if you need to check the audio levels. We recommend using headphones when using this feature.

Important:

- You cannot enable or disable audio feedback while recording; the checkbox will be locked

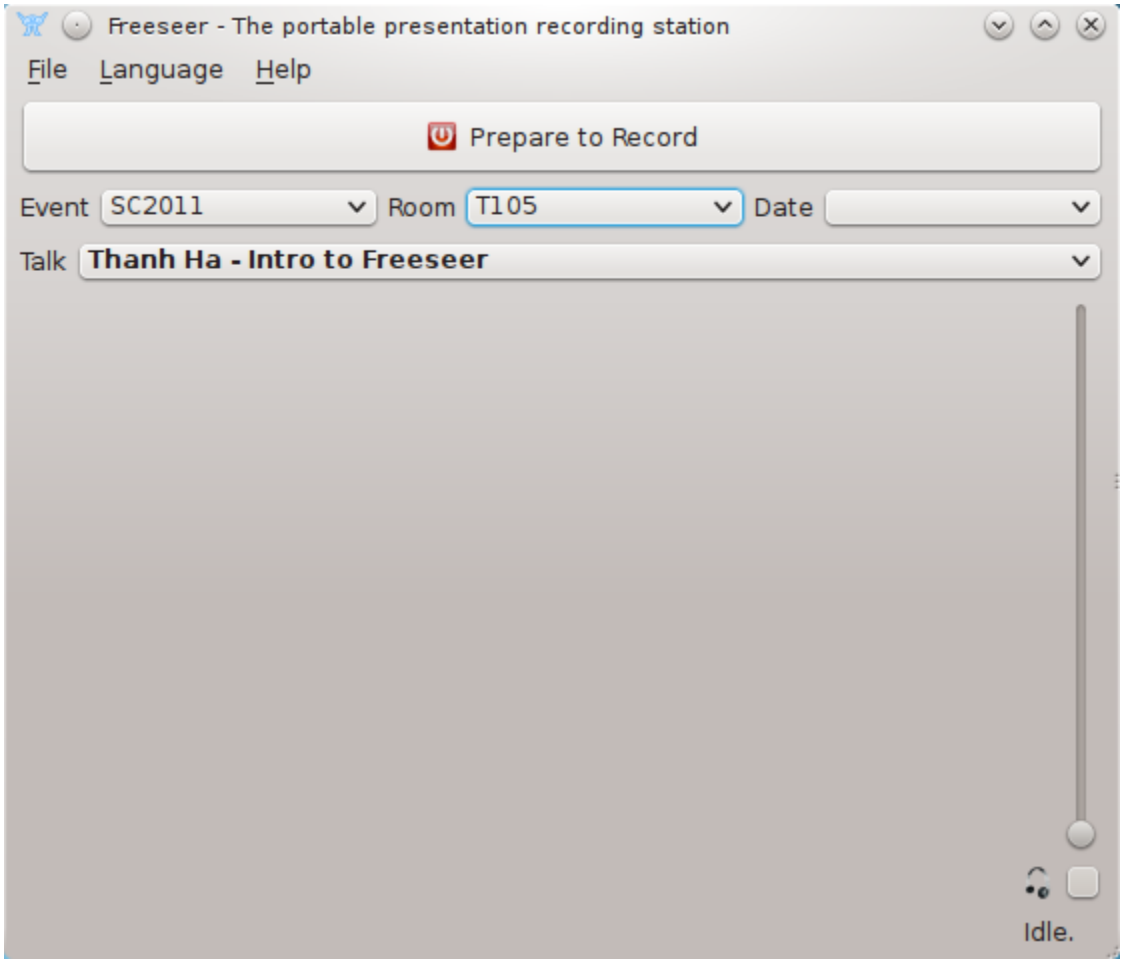


Figure 3.1: *Freeseer's Recording interface with a sample talk loaded*

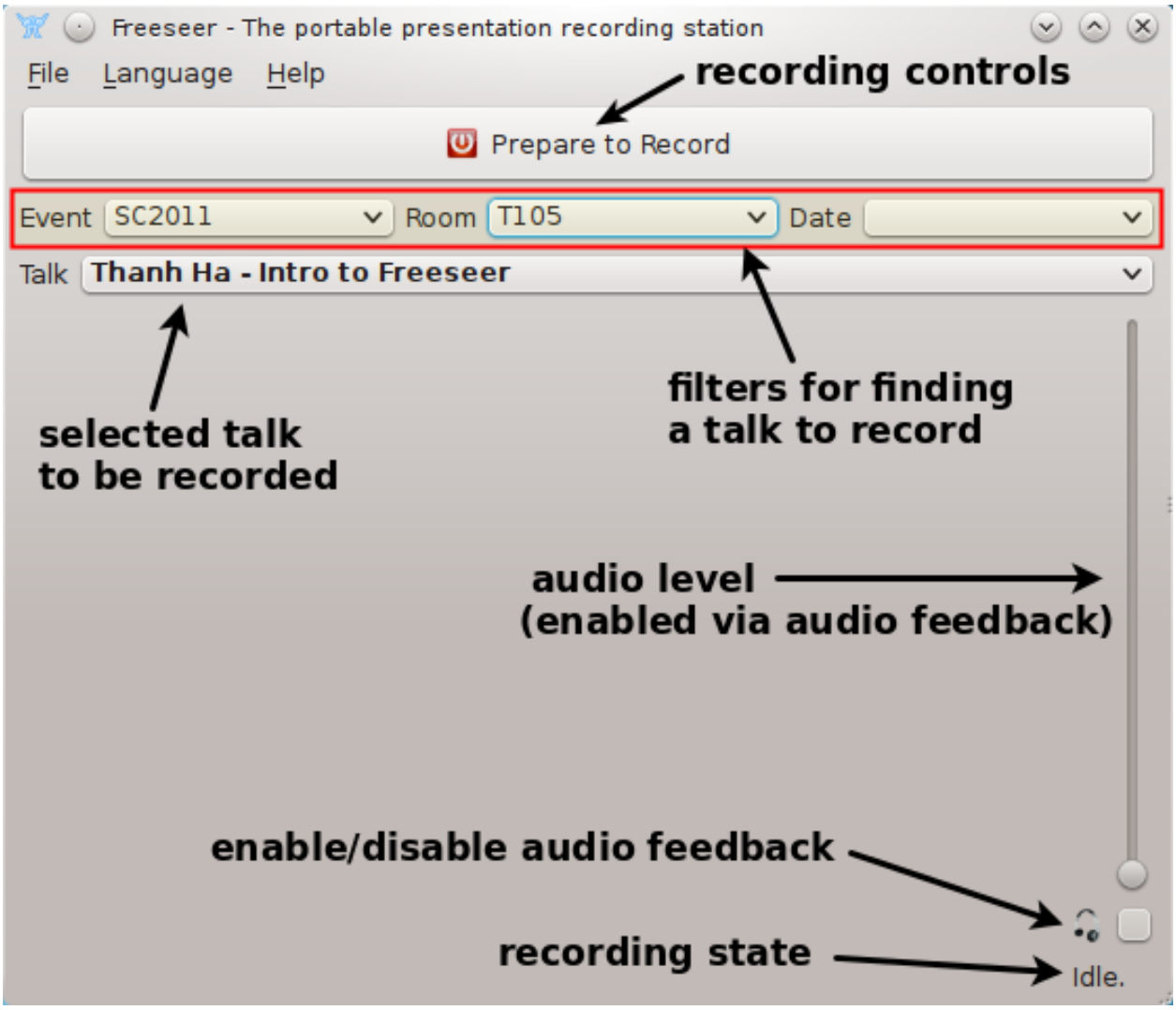


Figure 3.2: *Freeseer's Recording interface with notes*

- Audio feedback is only audible while recording

Status Area

At the bottom of the interface where it says “Idle”, is the status area. It shows information such as what state the program is in and how much free space is remaining once you start recording.

Recording Controls

Pressing “Prepare to Record” will initialize the Freeseer recording backend. Freeseer has to be “prepared” to record because of a technical limitation.

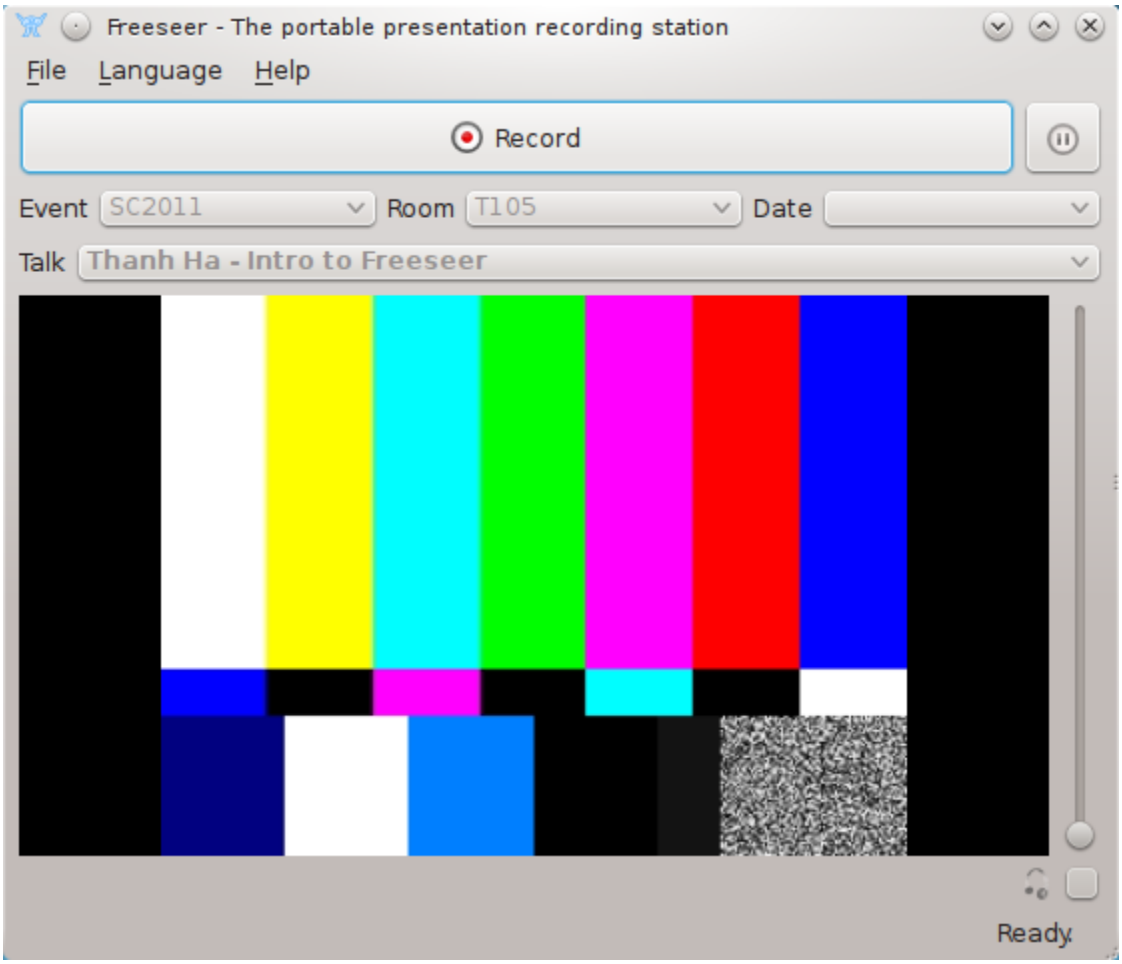


Figure 3.3: *Freeseer is ready to record*

Once “Prepare to Record” is clicked, new buttons “Record” and “Pause” will replace the “Prepare to Record” button.

Click “Record” to begin recording.

During a recording you can click “Pause” to pause the recording and resume it at a later time. This is useful if presentation contains a break period.

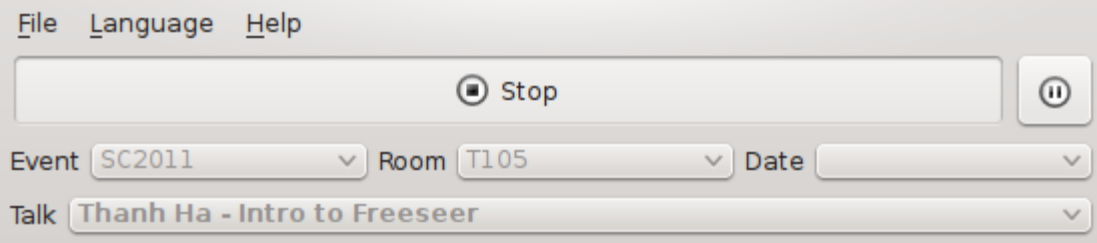


Figure 3.4: *Freeseer shows a Stop button during recording*

When a recording is in progress the "Record" button will change into a "Stop" button. Clicking this will stop recording and Freeseer's interface will return to its initial state.

3.1.2 Report Tool

Issues can occasionally occur during a recording, such as the audio or video not working, or perhaps the presenter does not wish to be recorded. After a conference, it's difficult and time consuming for one person to know which recordings have issues, especially if 100s of talks were recorded at the event.

To solve this issue, Freeseer comes with a basic reporting tool that allows the person recording to report issues with their recordings inside the application. This will allow whomever is uploading the recordings to quickly scan through and find those which have issues.

Using the Report Tool

To access the report tool:

1. Open **freeseer-record**
2. Click **Help > Report**

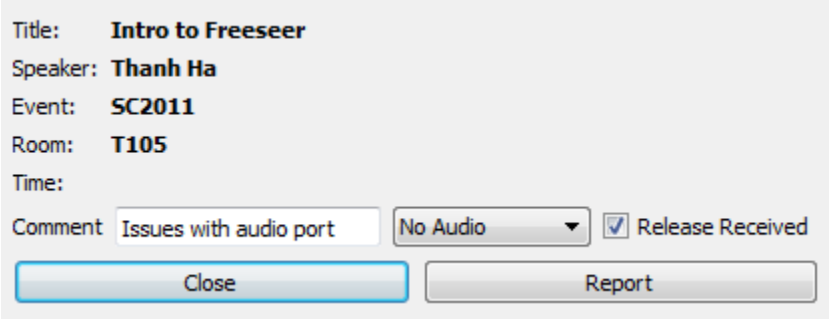


Figure 3.5: A basic form will open for the currently selected talk.

The form has 3 fields which need to be assessed by the reporter.

1. A textfield for a short comment describing the issue
2. A dropdown list containing options for the type of issue
   - Current options are "No Issues", "No Audio", "No Video", and "No Audio/Video"
3. A checkbox indicating if a Release Form was received [1]

[1] The "Release Received" checkbox was added since many events require presenters to sign a legal release form indicating they are giving permission to record their talk.

After an event is complete the organizer can use the *Report Editor* to view the submitted reports.

3.1.3 Record Over a Network

Via Command Line Interface (CLI)

Todo

Document using SSH and Freeseer's CLI to record over a network.

Via Graphical User Interface (GUI)

The client tool is used for controlling Freeseer over a network. The client connects to a running Freeseer *Server*. The server can be used to start, pause, and stop recording on multiple remote instances of Freeseer.

To set up your client:

1. Open **freeseer-record**
2. Click **File > Connect to server**

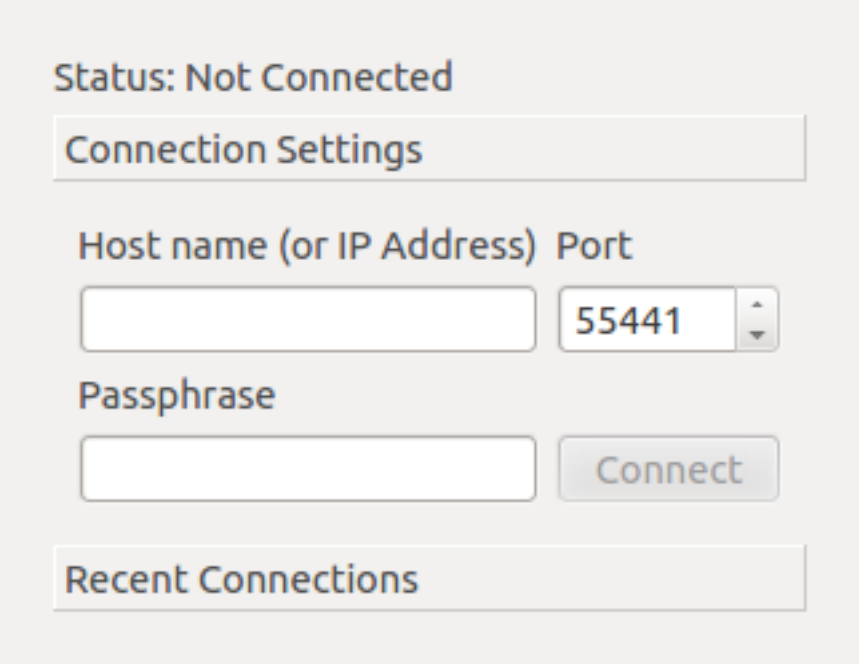


Figure 3.6: *The client window*

Simply enter the host name or IP address of the server tool, enter the passphrase, then hit *Connect*.

If you've connected to a server in the past, you can also use the *Recent Connections* tab to select a server.

3.2 Talk Editor

Talk management is what sets Freeseer apart from other screencast software and why fosslc.org is able to record numerous events and talks very quickly.

Using the Talk Editor you can pre-populate a database containing all the information necessary for an event. Which rooms, time, Speaker, Title, Description, etc... can be prepopulated in the database. This database is then used later by the *Record* to filter metadata to a room and also datafill the produced video files with this metadata.

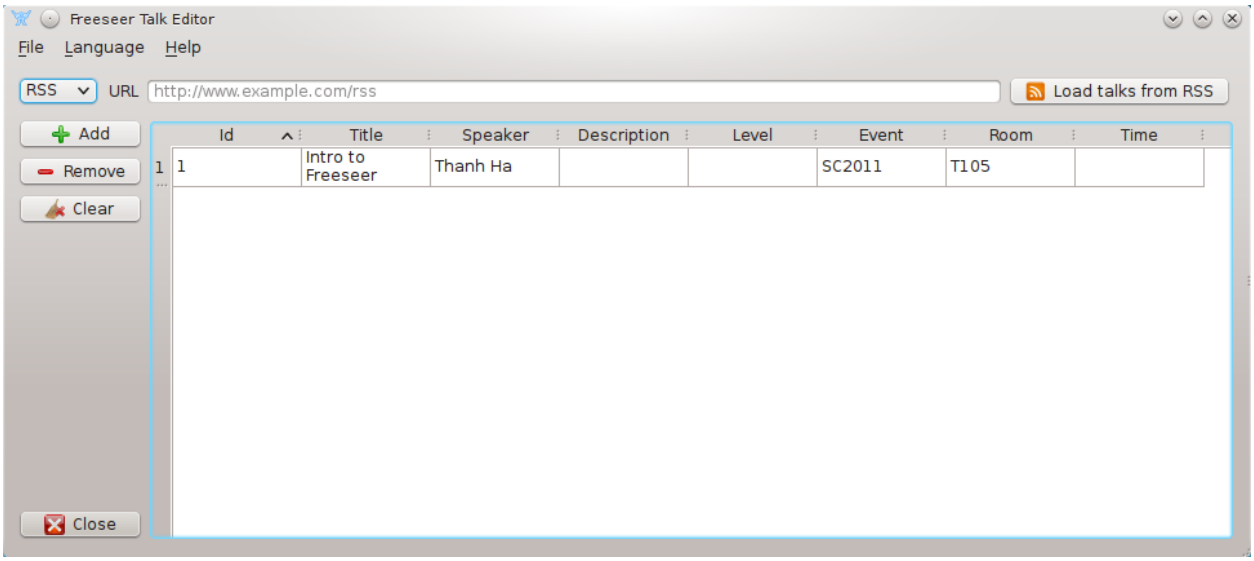


Figure 3.7: *Talk Editor*

Importing talk data from a RSS or CSV file is possible by entering the URL to the files in the URL bar and clicking Load.

You can "Add" and "Remove" a talk by clicking the appropriate buttons. Clicking "Clear" will clear the database.

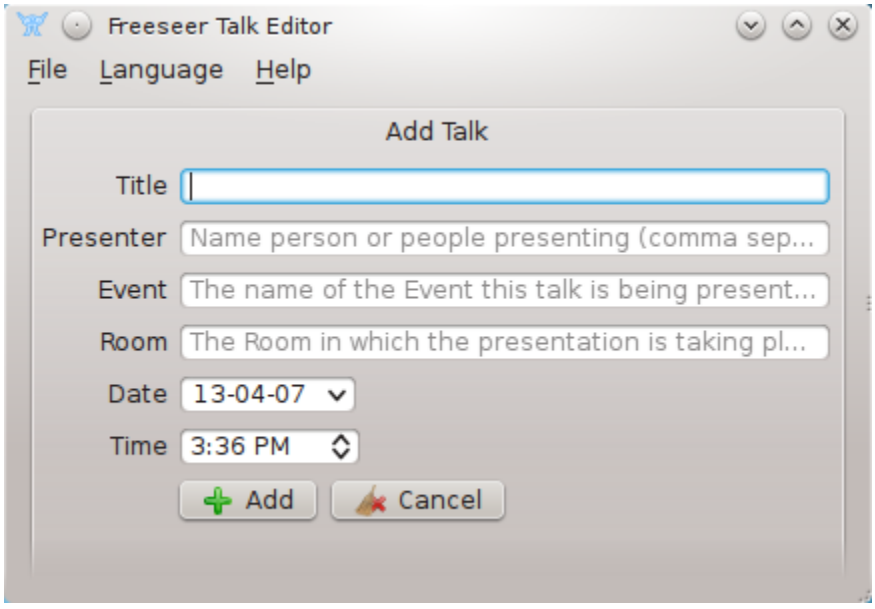


Figure 3.8: *Add Talk*

To add a talk at minimum you should at least enter a Title. A speaker is recommended. All other Fields are optional.

3.3 Configuration (Settings)

Todo

This page is still in progress.

3.3.1 General

3.3.2 Command line

There's currently two main configuration options:

`freeseer config reset` Wipes various contents of the `~/.freeseer` directory. A fresh configuration is useful when encountering weird issues.

`freeseer config youtube` For configuring Freeseer's YouTube Uploader. The `--client-secrets` and `--token` options are used in the authentication process.

The `-h/--help` option can be used for more info.

```
$ freeseer config --help
usage: freeseer config [-h] {reset,youtube} ...

positional arguments:
  {reset,youtube}
    reset          Reset Freeseer configuration and database
    youtube        Obtain OAuth2 token for Youtube access

optional arguments:
  -h, --help       show this help message and exit

$ freeseer config reset --help
usage: freeseer config reset [-h] [-p PROFILE] {all,configuration,database}

positional arguments:
  {all,configuration,database}
                        Resets Freeseer (default: all)

                               Options:
                                   all           - Resets Freeseer (removes the Freeseer configurati
                                   configuration - Resets Freeseer configuration (removes freeseer.c
                                   database      - Resets Freeseer database (removes presentations.c


optional arguments:
  -h, --help            show this help message and exit
  -p PROFILE, --profile PROFILE
                        Profile to reset (Default: default)

$ freeseer config youtube --help
usage: freeseer config youtube [-h] [--auth_host_name AUTH_HOST_NAME]
                               [--noauth_local_webserver]
                               [--auth_host_port [AUTH_HOST_PORT [AUTH_HOST_PORT ...]]]
                               [--logging_level {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                               [-c CLIENT_SECRETS] [-t TOKEN]
```

```
optional arguments:
  -h, --help            show this help message and exit
  --auth_host_name AUTH_HOST_NAME
                        Hostname when running a local web server.
  --noauth_local_webserver
                        Do not run a local web server.
  --auth_host_port [AUTH_HOST_PORT [AUTH_HOST_PORT ...]]
                        Port web server should listen on.
  --logging_level {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Set the logging level of detail.
  -c CLIENT_SECRETS, --client-secrets CLIENT_SECRETS
                        Path to client secrets file
  -t TOKEN, --token TOKEN
                        Location to save token file
```

3.3.3 Plugins

See *Plugins*.

3.4 Plugins

3.4.1 Audio Input Plugins

3.4.2 Audio Mixer Plugins

3.4.3 Video Input Plugins

3.4.4 Video Mixer Plugins

3.4.5 Output Plugins

3.5 Report Editor

After an event, the organizer can use the Report Editor in Freeseer to quickly browse issue reports from recordings to find problematic files.

3.5.1 Using the Report Editor

Selecting a report will show all the given details belonging to the talk, on the right pane of the window.

There are a few operations which can be performed from this interface:

- **Modify** the data of a cell (by double clicking it)
- **Remove** the currently selected row from the database
- **Clear** the entire report database (a confirmation dialog box will appear)
- **Close** the Report Editor

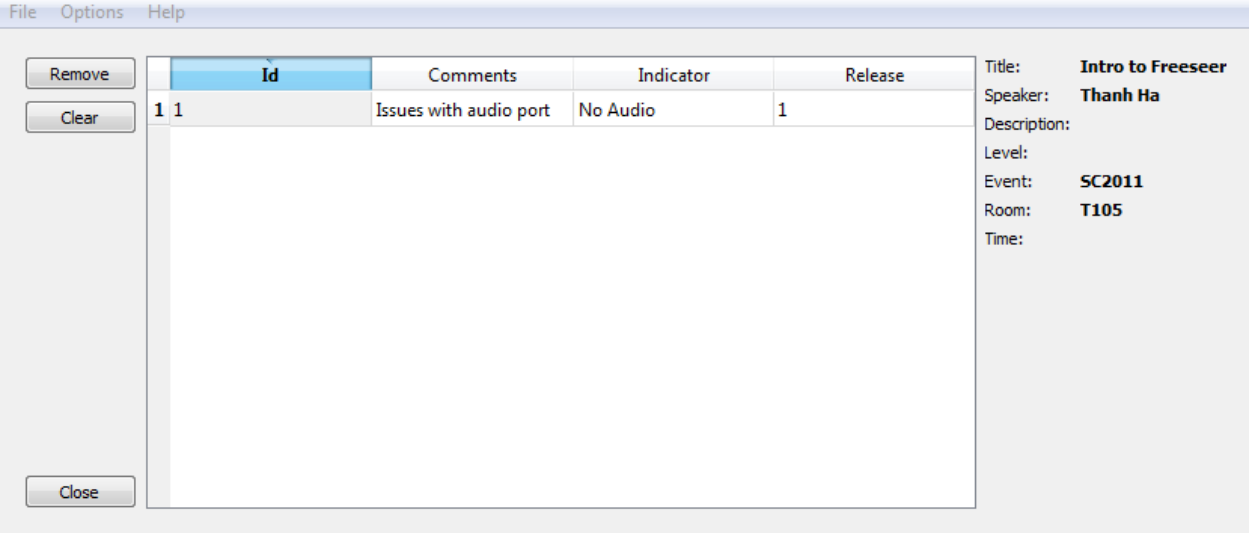


Figure 3.9: The editor provides a simple interface which displays all submitted reports found in the database.

3.6 Server

The *Freeseer Server* tool communicates with the *Freeseer Record* tool over a network. The server can be used to **start**, **stop**, **pause**, and **resume recording**.

3.6.1 Usage

1. Open **freeseer-server**

Before clicking "Start Server" button, you need to select the IP and port you wish to listen for connections on. 0.0.0.0 will listen on all available IP Addresses. Additionally selecting a passphrase for the server this password is needed by Freeseer clients and must match what is configured on the server in order to successfully connect to the server. This passphrase is not secure and is simply a simple check to ensure the client is connecting to the right server.

Once the server is running the server connection details will appear in the text box at the top. This is a convenience feature which you can use to copy and paste the settings in the client dialog.

Pressing "Stop Server" will stop the server.

Clicking the "Control Clients" tab will switch to a view that will allow you to see all clients currently connected to the server.

From here we can tell the client to start, pause, or stop recording. Client's status is shown next to the IP address of the client. Right now the client is idle so we can start recording. When the "Start Recording" button is pressed the client's status will change. According to the client's status, the buttons will updated with the appropriate labels.

Clients can also be disconnected if "Disconnect" button is triggered. The Client can also be disconnect from client side.

When client is recording we could pause or stop recording. Triggering the appropriate button will send the action to the client.

When recording is paused it can be resumed or it can be stopped.

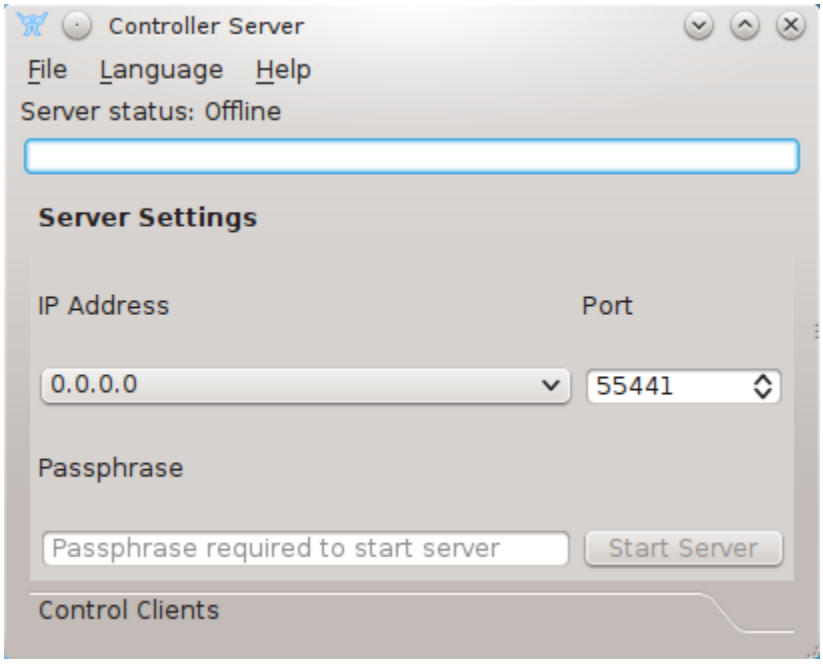


Figure 3.10: *Main server interface*

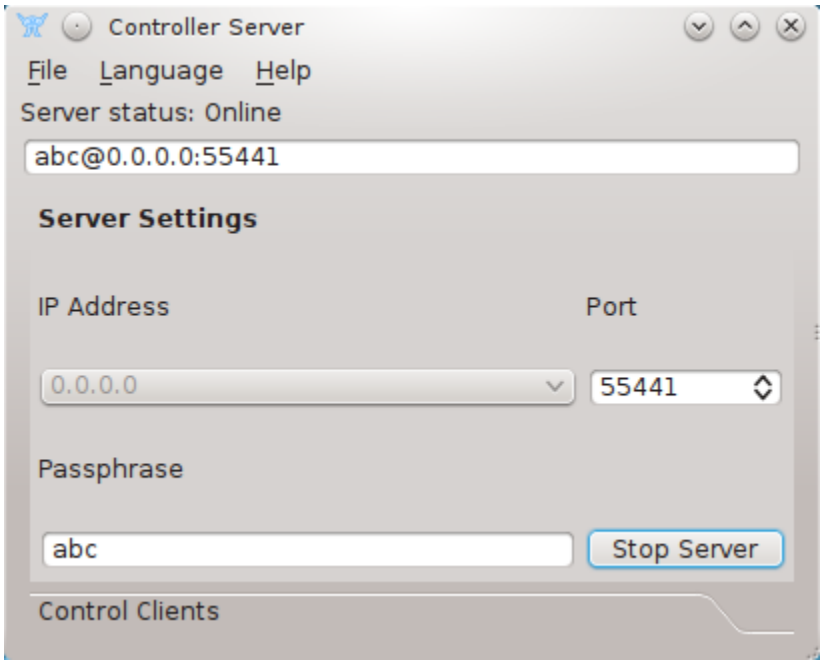


Figure 3.11: *Server interface while running*

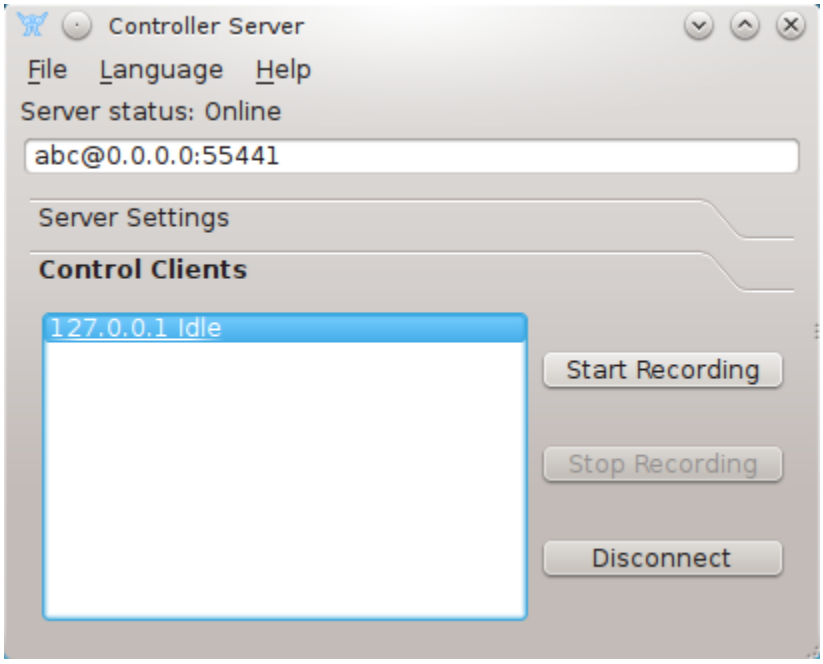


Figure 3.12: *A client is connected*

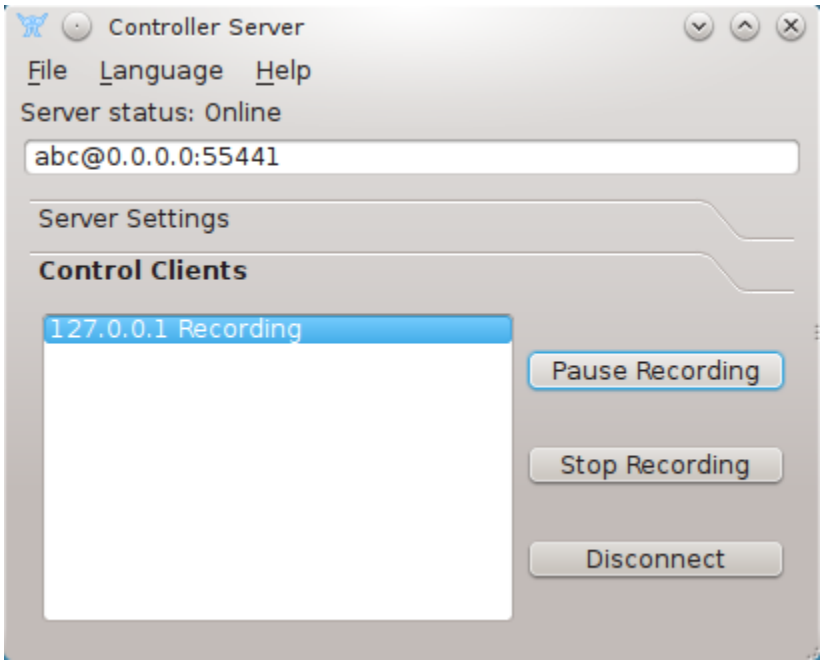


Figure 3.13: *Client in recording state*

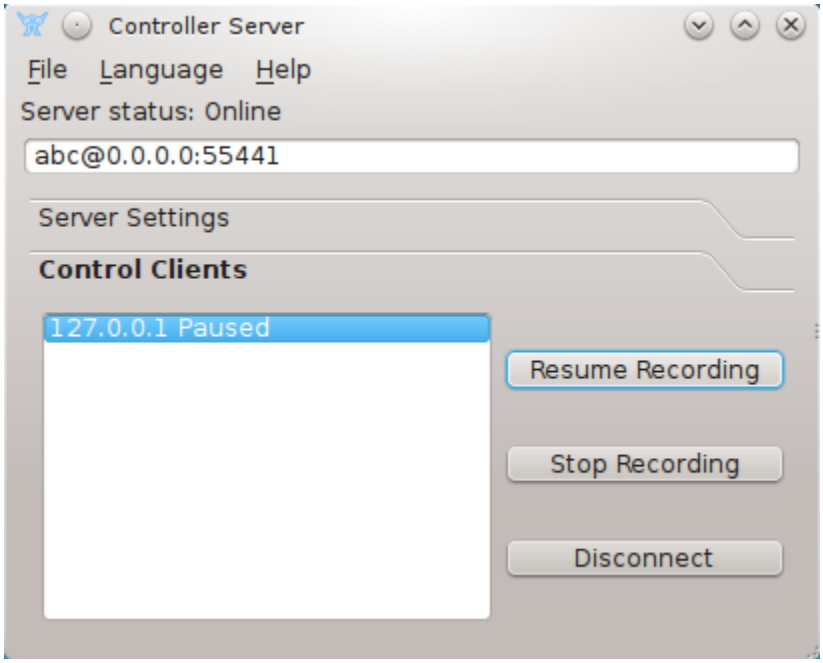


Figure 3.14: *Client in paused state*

3.7 YouTube Uploader

This is a CLI tool that can upload OGG and MPG videos to YouTube. It uses youtube-upload, an open source tool for uploading videos from the command line. It can upload a single video or an entire directory (including subdirectories). If the video is in the Freeseer OGG format, the metadata for the title and descripion will be used to populate the YouTube title and description fields. Otherwise, the title will default to the filename and the description will be blank. The category is set to *Education* by default.

3.7.1 Dependencies

- Google Data APIs Python Client Library (v1.2.4)
- Mutagen (v1.20)
- Python Progress Bar (optional, v2.3)

3.7.2 How to Use

The youtube-uploader tool will prompt you for your email address associated with YouTube, your password, and the path to the file or directory you want to upload. Specifying a directory will upload all videos in that directory and its subdirectories.

```
./src/youtube-uploader
Email (user@example.com): <user enters their youtube account email>
Password: <user enters password>
Video or Directory: path/to/videos <user enters file or directory to be uploaded>
```

If the login credentials are valid, the video(s) will be uploaded sequentially and their URLs will be displayed.

Note: The video directory configured with Freeseer is used to find videos.

CHAPTER 4

Contributor Guide

We value developers, but a project doesn't just run on code contributions. Writers, designers, power users, and others all play a role in our success.

How you can contribute depends on your skillset and interests. If you need help deciding, `tell us` what your skills are and what you're most excited about in Freeseer, and how you'd like to help. The community can point you in the right direction.

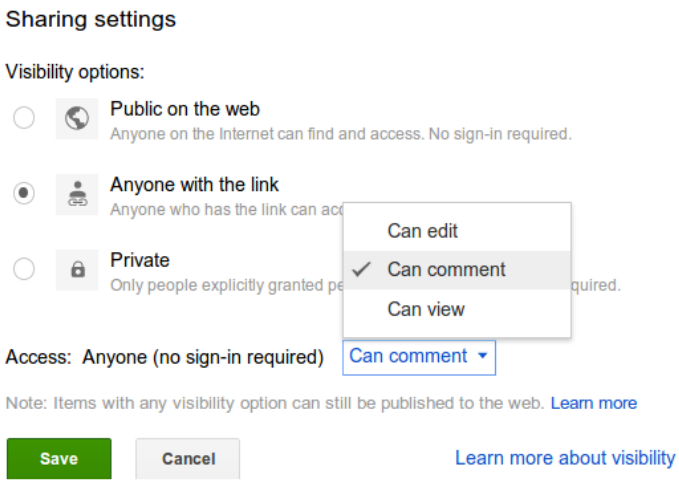
4.1 I am a...

4.1.1 Student

We love working with students and having them solve fun and challenging problems. For many, it's their first time contributing to an open source project. We've had students from Google Summer of Code (GSoC), Undergraduate Capstone Open Source Projects (UCOSP), and Fedora Summer Coding.

Getting Started

Step 7: Alter sharing settings for your proposal



1. Create a GitHub account
2. Introduce yourself on our *mailing list*
   - Include your GitHub username so we can add you to the Freeseer organization
3. Get familiar with IRC and start hanging out in *#freeseer on Freenode*
4. *Fork & clone Freeseer*
5. Get Freeseer up and running, play around with it, look at the source code
6. Read over the assessment document
7. Copy the project proposal document to your Google Drive
   - `File → Make a copy`
   - Change the sharing settings so **anyone with the link can view and comment**, then share the link with us
   - Replace the highlighted sections with your own content
8. Bookmark the style guides that we use: PEP 8 and Google's Python Style Guide
   - Reference them when writing code
9. Sign up for a blogging platform (or use your own website)
   - Share the URL on the mailing list
10. Start thinking of ways how you can improve Freeseer!

Tip: As a student, you can apply for a **free micro account** at github.com/edu and get 5 private repositories!

What Happens at the Sprint

Depending on which program you're participating in, you may have to attend a sprint with your new teammates. Use the sprint to ask questions, help your teammates, brainstorm what to work on, get started on your project proposal, open a pull request, make new friends, and generally have a good time.

We'll also decide on a time for our weekly G+ hangouts, which are done scrum style. Each team member will briefly update the others with what they've accomplished since the last meeting, what they will be working on next, and any stumbling blocks they ran into. Any detailed discussion should happen outside the meeting.

Deciding What To Work On

We let students choose their projects. We'll suggest a couple of project ideas. You can also look through our open issues or come up with your own idea and open an issue for it.

You should have a copy of the project proposal document on your Google Drive. Feel free to add or remove sections that you think are relevant or irrelevant to your project. It also includes a timeline to help schedule your work. There are no set work hours – meet your deadlines and you'll be fine. Decide with your professor and/or project mentor when your exact end date will be.

Done is better than perfect when it comes to your proposal. Don't spend too much (or too little) time on it. Iterate on it quickly and often, you don't need a highly polished document on your first try.

When you finish your first draft of the proposal, notify everyone on the mailing list and include a link so they can view it on Google Drive. We'll give you feedback so you can make any changes if necessary.

Note: You can *add or update translations* in addition to your main project.

Tip: Start working on your project while your proposal is still in progress.

Communicate

One of the biggest indicators of success is staying in touch with us. If we don't know what's happening on your side, we can't help you. If we don't hear from you, we usually won't go looking after you.

Make an effort to hang out in the IRC channel daily (lurking is fine). Participate in the mandatory weekly status update meetings. Be responsive on GitHub and the mailing list.

You'll also be required to write weekly blog posts about your progress. These will cover the updates you'll discuss in the weekly meetings, but you can go into more detail on your blog. You may write your blog posts at fosslc.org if you don't want to use a personal blog. Remember to tell us on the mailing list where we can read your latest blog post.

Your updates in the weekly meetings and blog posts shouldn't come as a big surprise to us. If that's the case, you should probably be communicating more often.

See also:

How to succeed or fail at Google Summer of Code

Expectations

- Be available for others to contact you
- Keep up to the date with the mailing list
- Communicate often, at the very least lurk on IRC
- Be a team player, not just a teammate
- Don't be afraid to ask for help
- Treat your project as a scientific experiment; a failed outcome is not a failed project if well documented
- 8-10 hours of work per week, as much as any other course

4.1.2 Developer

Freeseer hopes to be one of the most fun open source projects you could spend your time hacking on. Our codebase is still fairly small, so you should be able to get involved quickly.

Freeseer is written in Python and uses the Qt framework for the interface and GStreamer as the multimedia framework.

Note that some of the existing code may not be well documented or may not follow our coding guidelines and styleguides that we use. If you come across this, please notify us by opening an issue, or open a Pull Request if you'd like to fix it.

Style Guide

Every major project has its own style guide: a set of conventions (sometimes arbitrary) about how to write code for that project. It's much easier to understand a large codebase when all the code is in a consistent style.

Freeseer follows PEP 8 (Style Guide for Python Code) and the Google Python Style Guide. The two style guides mostly overlap but the Google one is more detailed and easier to navigate. Freeseer also contains tests that will loosely check the code for PEP 8 compatibility.

Not all existing code follows these guidelines, but all new code is expected to.

Custom Guidelines

There are a few exceptions or additions to the above style guides.

Line Length

We use 120 characters at most, instead of the common 80-column limit.

Python 3 Compatibility

Write code that's compatible with Python 3 whenever possible. This will make our transition easier.

Printing vs Logging

Do not print for debugging purposes, log instead. Make sure to use the appropriate logging level. The logging calls may come in handy in the future, so consider leaving them in.

Use print when working on a CLI tool and the output must be shown to the end user.

Log on a Per-Module Basis

Create an instance of a logger inside your module and name it after the module that contains it by using `__name__`.

Good:

```
import logging
log = logging.getLogger(__name__)
log.info("All your base are belong to us")
```

Bad:

```
import logging
logging.info("For great justice")
```

String Formatting

For logging, use printf style formatting. For everything else, use `str.format()`.

Good:

```
log.info('%s : %d', key, value)
greeting = 'Hello {} {}'.format(first_name, last_name)
print('{0} - {1} - {0}'.format(foo, bar))
```

Bad:

```
log.info(key + ' : ' + value)
greeting = 'Hello %s %s' % (first_name, last_name)
print foo, '-', bar, '-', foo
```

Write Short Methods

Methods and functions should be kept small and focused.

Long methods are sometimes appropriate, so no hard limit is placed on method length. However, if a method exceeds 40 lines or so, think about whether it can be broken up without harming the structure of the program.

Write Descriptive Docstrings

Comments should be descriptive ("Opens the file") rather than imperative ("Open the file"). The comment *describes* the method, function, or class, it does not tell it what to do.

Testing

Configure your Test Environment

Freeseer should not need additional configuration after installing the *development requirements*. This is because Python's `pytest` and PyQt's `QtTest` module are used for Freeseer's test suite. The `pytest` module is a feature rich testing framework that makes writing tests simple. The `QtTest` module is included with the PyQt4 package, which you should have installed as it's a dependency for Freeseer.

If you want to make sure you have the packages, you can start the Python interpreter and import `pytest` and `QtTest`:

```
>>> import pytest
>>> from PyQt4 import QtTest
```

If there are any errors, you won't be able to proceed with testing.

Extending the Test Suite

Structure of Test Directory All of Freeseer's tests exist in `src/freeseer/tests/`. Since Freeseer is well organized into modules, we'd like to mirror this setup in the test folder. This means that if your code is located in `src/freeseer/framework/core.py` then your test code should be found in

`src/freeseer/tests/framework/test_core.py` (more about file naming conventions later). We do this for logical ordering: it tells us that test modules in `src/freeseer/tests/folder_name` are for testing modules in `src/freeseer/folder_name`.

If you are creating a new folder in `src/freeseer/tests/`, ensure that your folder contains a `__init__.py` such that your test module can be imported.

Adding/Editing a test module

An example We show a set of test methods for the database class found in `src/freeseer/framework/database.py`. This class contains a database connector that lets the framework fetch stored data efficiently. The purpose of this unit test is to demonstrate some of the functionality that is provided by `pytest`.

To create a test module make an empty file with the name `test_database.py`. The convention used by Freeseer is `test_module_name.py` where the module counterpart is name `module_name.py`. Thus, your module name should start with **test_** and finish with **.py** at the very least.

Let's add fake functionality to the test module `test_database.py`!

```
import os

from PyQt4 import QtSql
import pytest

from freeseer.framework.config.profile import Profile
from freeseer.framework.plugin import PluginManager
from freeseer.framework.presentation import Presentation

@pytest.fixture
def db(tmpdir):
    """Construct a database connector fixture"""
    profile_path = str(tmpdir)
    profile = Profile(profile_path, 'testing')
    return profile.get_database()

def test_query_result_type_is_query(db):
    assert isinstance(db.get_talks(), QtSql.QSqlQuery)
    assert isinstance(db.get_events(), QtSql.QSqlQuery)
    assert isinstance(db.get_talk_ids(), QtSql.QSqlQuery)
    assert isinstance(db.get_talks_by_event('SC2011'), QtSql.QSqlQuery)
    assert isinstance(db.get_talks_by_room('T105'), QtSql.QSqlQuery)

def test_query_result_type_is_presentation(db):
    assert isinstance(db.get_presentation(1), Presentation)

def test_query_result_type_is_model(db):
    assert isinstance(db.get_presentations_model(), QtSql.QSqlTableModel)
    assert isinstance(db.get_events_model(), QtSql.QSqlQueryModel)
    assert isinstance(db.get_rooms_model('SC2011'), QtSql.QSqlQueryModel)
    assert isinstance(db.get_talks_model('SC2011', 'T105'), QtSql.QSqlQueryModel)

def test_add_talks_from_csv(db):
    """Test that talks are retrieved from the CSV file"""

    dirname = os.path.dirname(__file__)
    fname = os.path.join(dirname, 'sample_talks.csv')
```

```
    presentation = Presentation('Building NetBSD', 'David Maxwell')

    db.add_talks_from_csv(fname)
    assert(db.presentation_exists(presentation))
```

Break down of the unit test:

import pytest This lets us use all of the testing features provided by `pytest` like fixtures and function tests. It should be noted, unit tests written using the old framework will import the `unittest` module instead.

@pytest.fixture pytest provides fixture objects which allows a developer to put frequently created function call results into an object. Fixtures can be used in place of conventional setup functions as in `unittest`. In the example, the fixture contains a QtDBConnector object which all of the test methods can access. `unittest` teardown functions can be written with yield fixtures. Documentation on fixtures is available from pytest.

pytest test_* functions `pytest` will recurse into directories (that are not marked as *norecursedirs*), will look for `test_*.py` or `*_test.py` files, `Test` prefixed test classes, and `test_` prefixed functions. `pytest` will also discover traditional `unittest.TestCase` tests. Further documentation can be found on the pytest site.

It should be noted that testing the return type of function calls in unit tests is not very useful, as in the example `test_query_result_type_*()`.

The assert methods Each `assert` has the power to **FAIL** a `test_*` method. A test could contain several assert methods and will continue to run until an assertion fails. If no assertion fails, then the test will be marked as OK. It is important not to write too many assert statements in a test method. If this occurs than the test is probably trying to cover too many test scenarios and therefore the test should be broken up into smaller parts.

If an assertion fails `pytest` will give very generous failure information. For example, with the use of a fake test file:

```
import pytest

def test_crustacean():
    assert 'lobster' == 'crab'
```

The fake script will fail because the two strings are not equivalent, it will output the following when `$ py.test test_crustacean.py` is run from the command line:

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.6 -- py-1.4.23 -- pytest-2.5.2
plugins: cov
collected 1 items

test_crustacean.py F

=================================== FAILURES ===================================
_____________________________________ test_crustacean _____________________________________

    def test_crustacean():
>       assert 'lobster' == 'crab'
E       assert 'lobster' == 'crab'
E         - lobster
E         + crab

test_crustacean.py:6: AssertionError
========================== 1 failed in 0.01 seconds ============================
```

Running the Test Suite

Introduction We've written our test case(s) and now we want to see the results. First, let's go over the expected results:

```
@pytest.fixture
def db(tmpdir):
    """Construct a database connector fixture"""
    profile_path = str(tmpdir)
    profile = Profile(profile_path, 'testing')
    return profile.get_database()
```

The `pytest` fixture creates an instance of the QtDBConnector object and allows each method matching the `test_*` pattern to have access to it. The fixture is created each time a `test_*` function receives it as an argument. The argument `tmpdir` is a `pytest` built in test function argument that provides a unique temporary directory to each test function that calls it. The example recreates the `QtDBConnector` each time a test function uses the `db` fixture. If you need to finer control over how the fixture is created then refer to the pytest documentation.

```
def test_query_result_type_is_query(db):
    assert isinstance(db.get_talks(), QtSql.QSqlQuery)
    assert isinstance(db.get_events(), QtSql.QSqlQuery)
    assert isinstance(db.get_talk_ids(), QtSql.QSqlQuery)
    assert isinstance(db.get_talks_by_event('SC2011'), QtSql.QSqlQuery)
    assert isinstance(db.get_talks_by_room('T105'), QtSql.QSqlQuery)

def test_query_result_type_is_presentation(db):
    assert isinstance(db.get_presentation(1), Presentation)

def test_query_result_type_is_model(db):
    assert isinstance(db.get_presentations_model(), QtSql.QSqlTableModel)
    assert isinstance(db.get_events_model(), QtSql.QSqlQueryModel)
    assert isinstance(db.get_rooms_model('SC2011'), QtSql.QSqlQueryModel)
    assert isinstance(db.get_talks_model('SC2011', 'T105'), QtSql.QSqlQueryModel)
```

In the `test_query_result_type_is_*()` test functions, we are checking that the database queries return expected types.

```
def test_add_talks_from_csv(db):
    """Test that talks are retrieved from the CSV file"""

    dirname = os.path.dirname(__file__)
    fname = os.path.join(dirname, 'sample_talks.csv')

    presentation = Presentation('Building NetBSD', 'David Maxwell')

    db.add_talks_from_csv(fname)
    assert(db.presentation_exists(presentation))
```

Finally, in `test_add_talks_from_csv()`, we are checking that we can also add talks from comma separated value format files.

Command line options **Note: to avoid package import errors, we need to run the following commands from the src folder.**

Example: Run all tests with pytest To run all of the tests in `src/freeseer/tests/`, issue the following command from the `src/` directory:

```
$ python setup.py test
```

The output will contain information about the test session. If there are any failures during the session then failure messages will be logged and testing will continue. If there is a failure, the developer may read through the output to see what went wrong. Information related to which line the failure occurred is printed in the output's **FAILURES** section, as well as **DEBUG** or **INFO** output that was printed to stderr in the erroneous code. At the bottom of the output from the script statistics on code coverage are displayed.

Gotchas! a.k.a Q&A

Q: I set a variable in one of my unit tests, but my other unit tests cannot see the values I set!

A: There is no guarantee for the order in which unit tests run. It is also not a good practice to have dependencies between unit tests. Each of the unit tests should be stand alone and should not alter the test environment for tests running after said unit test. If you want to test that a unit test produces a given value, then the result of the unit test could be compared to a fixture to assert the condition has been met. The same fixture could then be used in the following unit test that you were using the result of the prior unit test in. This would separate the two unit tests from depending on the order in which they are ran by the test suite.

Q: Can pytest run UnitTest files?

A: Yes, `pytest` can run `unittest.TestCase` based unit tests if they follow the test discovery naming conventions.

What should testers focus on? Ultimately, testers should protect users and the organization from bad design, confusing UX, functional bugs, security and privacy issues, and so forth.

Some things testers should consider:

· Where are the weak points in the software?

· What are the security, privacy, performance and reliability concerns?

· Do all the primary user scenarios work as expected? For all international audiences?

· Does the product interoperate with other products (hardware and software)?

· In the event of a problem, how good are the diagnostics?

Plugin Framework

Freeseer uses plugins so developers can easily extend the capabilities of Freeseer in a modular fashion.

Freeseer's plugin framework is built on Yapsy, a minimal plugin system that only depends on Python's standard library.

Plugin System Setup

Yapsy's `PluginManager` class provides the core logic needed to find, load, and activate plugins. Freeseer has a `PluginManager` class that builds on top of that, and can be found in `src/freeseer/framework/plugin.py`.

Yapsy provides a `PluginFileLocator` class which locates plugins when they are accessible via the filesystem. [1] Plugins are described by a text file called the *plugin info file* which have a ".yapsy-plugin" extension by default. But Freeseer plugins use a customized extension, ".freeseer-plugin".

[1] The plugins are detected through Python, so all directories leading to plugins should have an `__init__.py` file in them.

Todo

introduce the code snippet so it's not shown as a surprise

```
from yapsy.PluginManager import PluginManagerSingleton
from yapsy.ConfigurablePluginManager import ConfigurablePluginManager
...

class PluginManager(QtCore.QObject):
    """Freeseer's Plugin Manager provides plugin support."""

    def __init__(...):
      ...
      PluginManagerSingleton.setBehaviour([ConfigurablePluginManager])
      self.plugmanc = PluginManagerSingleton.get()
      ...
      locator = self.plugmanc.getPluginLocator()
      locator.setPluginInfoExtension("freeseer-plugin")
      ...
```

Freeseer searches three different directory paths for plugins:

1. User's HOME directory (`~/.freeseer/plugins/`)
2. Relative to the src directory (`src/freeseer/plugins/`)
3. If you installed Freeseer, the Python installation packages (`site-packages/freeseer/plugins/`)

Yapsy's `IPlugin` class defines the minimal interface needed for Yapsy plugins. We want Freeseer's plugin classes to have a richer interface than what `IPlugin` provides, so we created the `IBackendPlugin` class in `freeseer/framework/plugin.py` which inherits from `IPlugin` and defines the minimal interface for all Freeseer plugin classes. All Freeseer plugins should descend from the `IBackendPlugin` class.

```
from yapsy.IPlugin import IPlugin
...

class IBackendPlugin(IPlugin):
    """Defines the interface for all Freeseer plugins."""
    CATEGORY = "Undefined"
    ...

class IAudioInput(IBackendPlugin):
    """A Freeseer plugin for Audio Input."""
    CATEGORY = "AudioInput"
    ...
```

Each class that is a descendant of the `IPlugin` class needs a `CATEGORY` attribute defined. When you are writing your own Freeseer plugin, you often don't need to define a new category. You can extend one of the existing plugin classes and will not need to override the `CATEGORY` attribute.

If you are creating a new category, you will need to override the `CATEGORY` attribute and add the new category name and class name to the PluginManager's category filter in the form of a key-value pair, where the key is the plugin's category and the value is the plugin's classname.

```
class PluginManager(QtCore.QObject):
    ...
    self.plugmanc.setCategoriesFilter({
        IAudioInput.CATEGORY: IAudioInput,
        IAudioMixer.CATEGORY: IAudioMixer,
        IVideoInput.CATEGORY: IVideoInput,
```

```
        IVideoMixer.CATEGORY: IVideoMixer,
        IImporter.CATEGORY:   IImporter,
        IOutput.CATEGORY:     IOutput})
    self.plugmanc.collectPlugins()
    ...
```

Yapsy provides a number of useful decorators for its PluginManager which modify behaviour. Freeseer's plugin system uses the `ConfigurablePluginManager` which allows Freeseer to save and load the active plugins and their settings to a configuration file.

```
from yapsy.ConfigurablePluginManager import ConfigurablePluginManager
...

class PluginManager(QtCore.QObject):
    ...
    PluginManagerSingleton.setBehaviour([ConfigurablePluginManager])
    ...
```

Many of the Freeseer plugins, such as the video and audio plugins, use the `ConfigurablePluginManager` to save the active plugins.

Creating a Plugin

The basic steps for creating a new plugin are:

1. Write a plugin info file, `plugin_name.freeseer-plugin`, inside the appropriate directory within `src/freeseer/plugins/`. This file will hold metadata for the plugin and has the following format:

   ```
   [Core]
   Name = Plugin Name
   Module = plugin_module_or_directory

   [Documentation]
   Author = Your Name
   Version = Latest version of Freeseer that your plugin is compatible with
   Website = http://fosslc.org
   Description = Simple one-sentence plugin description
   ```

2. Create the plugin Python file(s)

   - If you are creating a single-file plugin, create a Python module with the same name as your plugin info file:

   ```
   plugin_name.freeseer-plugin
   plugin_name.py
   ```

   - If you are creating a multi-file plugin, your Python modules should be separated from your plugin info file:

   1. Create a plugin directory with the same name as your plugin info file (minus the extension).
   2. In the new plugin directory, create the file `__init__.py` and write your plugin class inside it. Your class should extend one of the `IBackendPlugin` subclasses (e.g. `IAudioInput`). Don't forget to override the class atribute `name` with your plugin's name.
   3. Add other useful plugin code in other modules if necessary. For example, if your plugin requires a GUI, create a module called `widget.py` inside your plugin's directory and import it inside your plugin's `__init__.py` module.

Accessing a Plugin

Any modules that need to access the plugins will need to import Freeseer's `PluginManager`:

```
from freeseer.framework.plugin import PluginManager
```

There are a number of ways to access the plugins via the `PluginManager`. You can iterate over all of the plugins (or all of the plugins in a given category) or you can access a specific plugin by its name.

While Yapsy provides methods for accessing plugins (e.g. `getAllPlugin()`, `getPluginsOfCategory()`, and `getPluginByName()`), the recommended way to access the plugins is to use the accessor methods provided by Freeseer's `PluginManager`:

```
get_plugin_by_name(name, category)
get_all_plugins()
get_plugins_of_category(category)
get_audioinput_plugins()
get_audiomixer_plugins()
get_videoinput_plugins()
get_videomixer_plugins()
get_importer_plugins()
get_output_plugins()
```

When you call any of the above accessor methods, you receive a `PluginInfo` object or a list of `PluginInfo` objects. Such an object contains meta information about the plugin. Each `PluginInfo` object has an attribute `plugin_object` which returns an instance of the plugin which you can then use.

For example:

```
plugman = PluginManager(config_dir)
plugin_info = plugman.get_plugin_by_name(name, category)
plugin = plugin_info.plugin_object
plugin.do_something()
```

As another example, here's a snippet of the Freeseer codebase where a class uses a plugin. It does so by creating an instance of the PluginManager and then calls the plugin by name, using the `plugin_object` attribute to access the plugin object:

```
from freeseer.framework.plugin import PluginManager
...

class QtDBConnector(object):

    def __init__(self, config_dir, ...):
        ...
        self.plugman = PluginManager(config_dir)
        ...

    ...

    def add_talks_from_rss(self, feed_url):
        """Adds talks from an RSS feed."""
        plugin = self.plugman.get_plugin_by_name("Rss FeedParser", "Importer")
        feedparser = plugin.plugin_object
        presentations = feedparser.get_presentations(feed_url)
        if presentations:
            for presentation in presentations:
                talk = Presentation(presentation["Title"],
                                    presentation["Speaker"],
```

```
                                presentation["Abstract"],  # Description
                                presentation["Level"],
                                presentation["Event"],
                                presentation["Room"],
                                presentation["Time"])
            self.insert_presentation(talk)
        else:
            log.info("RSS: No data found")
```

Code Review

We use Pull Requests on GitHub for code reviews and merging proposed changes.

Pull requests let you tell others about changes you've pushed to a GitHub repository. Once a pull request is sent, we can review the set of changes, discuss potential modifications, and you can even push follow-up commits if necessary (without having to close and re-open the pull request).

See also:

- Creating a pull request
- Merging a pull request (for anyone with push access to the destination repository)
- Closing a pull request

What we look for

in a pull request

- Descriptive title
- Summary describing the changes
- Points to and from the correct branches
  - From your development branch to Freeseer's master branch
- Reference any related issues or resources

in the code

- Code should follow our *Style Guide*
- Code is well documented
  - Documentation should also exist in our online documentation for any new features
- Logic of the code makes sense
- Code is efficient and readable
- Code is modular
  - Similar code should be put in functions
  - Functions should be small and focus on one thing
- Your code is thoroughly documented and uses docstrings where appropriate
- Your branch can be merged cleanly into master
  - No merge conflicts

  - Your branch includes the latest commits from master (rebase to avoid merge commits)
- Includes unit tests for the new code
- All unit tests pass

in the commits

- Each commit should represent one type of change
- Commit messages are as descriptive as possible
- Commit messages follow our formatting guidelines
- Squash related commits into a single commit

Tips

- **Open a Pull Request as early as possible**

  Pull requests are a great way to start a conversation of a feature or a work in progress, so send one as soon as possible—even before you are finished with the code. Your team can comment on the feature as it evolves, instead of providing all their feedback at the very end.

- **Pull Requests work branch to branch**

  If you have push access to Freeseer/freeseer, you don't need to fork it to work on a new feature. Create your topic branch on Freeseer/freeseer instead, then make a pull request in the same repository.

- **A Pull Request doesn't have to be merged**

  Pull requests are easy to make and a great way to get feedback and track progress on a branch. But some ideas don't make it. It's *okay* to close a pull request without merging.

- **Anyone can review code**

  Reviewing code isn't exclusive to active contributors. Anyone is welcome and encouraged to review code—the more the better!

4.1.3 Translator

We want to adapt Freeseer for as many non-native environments as possible. There should be no language barrier between Freeseer and our users. Help us in our localization efforts.

Add a Translation

1. Open the **Qt Linguist** tool – it should come with your installation of PyQt
2. Translation files are located in `freeseer/src/freeseer/frontend/qtcommon/languages/`
   - If a file for your language exists, continue to step 3
   - Otherwise, you'll need to update translation resources first
3. Using Qt Linguist, open translation (.ts) files for languages you wish to add
4. Once the translation is complete, send a pull request

See also:

Qt Linguist manual for translators

Update Translation Resources

For new translations to appear in Freeseer, you need to update the translation resources. This task is typically left to a developer, not a translator. If you feel uncomfortable doing these steps, please ask a developer to update the translation resources.

1. Update Translation Files

This step only needs to be completed if a developer wrote code that contains new translation strings in the user-interface. To update translation files:

```
$ cd freeseer/src/freeseer/frontend/qtcommon/languages
$ pylupdate4 freeseer.pro
```

The `freeseer.pro` file specifies which source files contain translation strings, as well as which translation files need to be updated and/or created. If you want to translate to a new language, add a new locale for that language.

2. Add Qt Translation Files to Freeseer-monitored List

Next, you need to update the list of monitored translations by editing `freeseer/src/freeseer/frontend/qtcommon/resource.qrc`.

Add the following line:

```
<file alias="languages/tr_LANGUAGE_LOCALE.qm">languages/tr_LANGUAGE_LOCALE.qm</file>
```

where *LANGUAGE* and *LOCALE* are specific to your translation. For example, for an American English translation:

```
<file alias="languages/tr_en_US.qm">languages/tr_en_US.qm</file>
```

3. Update Qt Resource Files

When translations are ready to be used, they need to be imported into Qt's resource files. We included a script to automate the process. Simply run:

```
$ cd freeseer/src/freeseer/frontend/qtcommon
$ make
```

You should now see your translations the next time you run Freeseer.

4.1.4 Designer

Whether you're a graphical designer, user interface designer, or user experience designer, we can use your help. Design is becoming increasingly important in open source projects, yet finding designers to contribute remains challenging.

Good design is hard, so we need your help. Critique our design, show us where we can improve, and help us make the improvements. Create wireframes, mock-ups, sketches, and prototypes.

Our goal is to make Freeseer a highly usable app that's simple enough for a first time user and advanced enough for power users.

4.1.5 Power User

Power users have installed and upgraded Freeseer, know its ins and outs, and have used it to record several talks. They can help with mailing list and IRC support, issue tracker management, documentation, and suggestions.

4.2 Basics

No matter which role you take on, these are some of the basics you'll most likely have to deal with.

4.2.1 Basics

Git & GitHub

Freeseer is hosted on GitHub, which uses Git. You'll generally encounter both while contributing.

Git allows many people to work on the same documents (e.g. source code) at the same time, without stepping on each other's toes. It's a *distributed version control system*. Git can be complicated for beginners or for people who have previously used a version control system that's different by design (e.g. Subversion).

No worries, there are tons of online resources. If you can't find what you're looking for, please ask us!

Before we begin, keep in mind that there is no *correct* way of using git. With git, you can often achieve the same results via different ways. How you use git is often determined by the guidelines of the project you're contributing to. The examples on this page are how we recommend you use git when contributing to Freeseer.

See also:

- Set up Git and your GitHub account → help.github.com
- Learn by doing → Try Git
- Learn by watching → Git Videos
- Learn by reading → Pro Git book (free)
- Forgetful? → Git Cheat Sheet
- Eclipse is your preferred IDE? → Eclipse Git Plugin
- Prefer GUIs over CLIs? → GitHub for Windows or GitHub for Mac

Forking and Cloning Freeseer

1. Go to the Freeseer repository on GitHub and click the fork button. This creates your own public copy of the project under your GitHub profile (github.com/username/freeseer). A fork allows you to easily use someone's project as a starting point for your own.

2. So far your fork only exists on GitHub. You'll need to clone it to your local machine to be able to work on the project.

```
$ git clone https://github.com/your_username/freeseer.git
```

3. Your cloned repository has a default remote named `origin` that points to your fork on GitHub, which can be used for pushing and pulling updates. But there is no remote that points to the original repository that you forked from. Add a remote named `upstream` to keep track of the original Freeseer repository.

```
$ cd freeseer
$ git remote add upstream https://github.com/Freeseer/freeseer.git
$ git remote -v # Lists your remotes, you should see origin and upstream
```

Tip: The name `upstream` is by convention. You can use whatever name you prefer (e.g. `mainstream` or `mothership`).

Basic Workflow

Whenever you're going to make a set of edits to the project, you should create a topic branch (also called a feature branch) for your changes. Your topic branch will usually be branched off of master.

Never make changes directly in the master branch. Your master branch should mirror upstream's master branch, try to keep them in sync. You'll use your local master branch to pull in changes from upstream.

1. Switch to the master branch and pull in the latest changes from upstream.

```
$ git checkout master
$ git pull upstream master
```

2. Create and checkout a new branch. Please follow our *naming guidelines*.

```
$ git checkout -b my-topic-branch
```

3. Start making your changes. Commit early and often.

```
$ git add modified_file
$ git commit -m "Add foo" # Omit the -m flag to write a more detailed commit message.
```

4. After your first few commits, push your topic branch to GitHub.

```
$ git push -u origin my-topic-branch # The next time you need to push, simply use git push
```

5. Go to GitHub and open a pull request from your topic branch to upstream's master branch.

   This allows members of the Freeseer organization to easily see updates made to your branch and perform code reviews as you make changes. So please **open a pull request as soon as possible!**

6. Rebase frequently to incorporate changes from upstream.

```
$ git checkout master
$ git pull upstream master
$ git checkout my-topic-branch
$ git rebase master
```

7. Push your commits to GitHub frequently. At a minimum, push your changes when you're done working for the day.
8. When you consider your work complete and ready to be merged, rebase any changes from upstream into your branch once more (see step 6).
9. Squash any dirty commits via an interactive rebase, so the remaining commits are meaningful and comprehensible. For example, squash commits that only fix a typo or whitespace, and rewrite poor commit messages.

```
$ git rebase -i master
```

10. Let others know you consider your work ready to be merged by leaving a comment in your pull request. *You may be asked to make some changes.*
11. When your pull request has been merged, celebrate, then clean up by deleting your local and remote topic branch.

```
$ git checkout master
$ git pull
$ git branch --delete my-topic-branch # Deletes the topic branch on your machine (can also use -
$ git push --delete my-topic-branch # Deletes the topic branch on your fork
```

Warning: Performing an interactive rebase (as in step 9) will rewrite history, and should therefore only be used on personal branches. Never rewrite history on branches that others are also working on.

Tip: If you rewrite history that's already been pushed, you'll need to force push the next time (`git push -f`). Try to avoid forced pushes by only editing commits that haven't been pushed yet. Use `git rebase -i HEAD~N` to edit the last *N* commits.

Workflow Diagram

A visual representation of what a contributor's workflow should look like.

Reference Issues in your Commit Messages

Note: We use a single issue tracker for all of our repositories: github.com/Freeseer/freeseer/issues

Similar to how GitHub allows you to reference issues and commits from a comment on GitHub.com, you can also reference issues from a commit message.

Tip: Referencing issues from your commit messages makes it easy to view more context and see which commits are related.

There are two ways to reference issues.

1. Short form: *#123* or *GH-123*
2. Long form: *user/repo#123*

You can reference issues that belong to different repositories on GitHub using the long form. This is called a cross-repo reference.

If you forked a repository, you can use the short form to reference issues belonging to the original repository.

To close an issue from a commit message [2], place a supported keyword directly in front of the reference. For example, "Close #123" or "Fix gh-123".

Supported keywords

- close
- closes
- closed
- fix
- fixes
- fixed
- resolve
- resolves
- resolved

You can also close multiple issues in a single commit message, and close issues cross-repo if you use the long form. [3]

Tip: GitHub is case-insensitive to commit messages.

See also:

Closing issues via commit messages

Dealing with Conflicts

You'll run into a merge conflict eventually. It's when something doesn't match up between the local and remote copy of a file. To be more precise, a merge conflict usually occurs when your current branch and the branch you want to merge into the current branch have diverged. That is, you have commits in your current branch which are not in the other branch, and vice versa.

The secret is to use `git mergetool`. Here's one way how you can resolve conflicts:

```
$ git fetch upstream
$ git rebase upstream/experimental current-local-branch
... CONFLICT: Merge conflict in <filenames>
```

Now you have 3 options:

1. Selectively choose which parts of a file to use (using an external visual diff & merge tool):

   ```
   $ sudo apt-get install meld  # Install Meld (or at http://meld.sourceforge.net)
   $ git mergetool -t meld  # Some alternatives are kdiff3, opendiff, diffmerge, etc.
   ... The visual merge tool is launched.
   ... It shows three versions of the file (local, failed merge, remote).
   ... You can easily choose code from any and all of them to resolve conflicts.
   ... Don't forget to save the file when you're done.
   ```

2. Ignore their changes, use your file:

[2] You can only close an issue from a commit message if you have push access to that repository. In other words, if you can close the issue from GitHub.com, you can also close it from a commit message.

[3] This is useful when closing an issue in Freeseer/freeseer from a commit that belongs to another repository under the Freeseer organization.

```
$ git checkout --ours <filename>
```

3. Ignore your changes, use their file:

```
$ git checkout --theirs <filename>
```

Once you've resolved all conflicts:

```
$ git add <filename>  # Or 'git add .' to mark all files as resolved
$ git rebase --continue
```

To abort the conflict merging process at any time:

```
$ git rebase --abort
```

Renaming your Branch

Want to use a better name for your branch?

Renaming a local branch:

```
$ git branch --move old-name new-name  # Short option is -m
```

Renaming a remote branch is more difficult because git doesn't support it. A workaround is to delete the branch and re-add it with the new name:

```
$ git push origin new-name
$ git push origin --delete old-name
```

Reporting Bugs & Requesting Features

1. Search We troubleshoot and discuss features in public. If you've found a bug or have an idea, take a few minutes to see if it's already been documented.

Search our *documentation*, *mailing list*, issue tracker, and IRC log.

2. Ask Contact us before opening a new issue, otherwise you risk it being closed for reasons such as it being a known issue or previously rejected idea.

Hop in our *IRC channel* or send an email to the *mailing list* and describe your problem or idea.

3. Open a new issue After searching and contacting us, open an issue if none exist and reference any existing related issues that you know of.

If you're a new contributor, please use one of the templates below.

Bug Report Template

For bug reports, describe step by step exactly what you did and what went wrong.

```
Steps to reproduce the problem:
1.
2.
3.


What is the expected behavior?
```

```
What went wrong? (Place any screenshots here)


Did this work before?
- Not applicable / I don't know
- Yes, this is a regression
- No, I think it never worked


Any other comments? (E.g. Freeseer version, Python version, operating system, error messages, etc.)
```

Or use this conciser template:

```
Steps:
1.
2.
3.

Expected:

Observed:

Notes:
```

Feature Request Template

```
Purpose of feature (pros, cons, use cases):


Describe the feature and its functionality:


Mockups / Screenshots / Examples:
```

Of course you can also argue feature removal.

4.3 Best Practices

Some tips and suggestions to get the most out of your contributions!

4.3.1 Best Practices

Don't Develop on Master Branch

The master branch contains stable code that's ready to ship. The master branch build should always be passing.

The maintenance branch (if it exists) contains patches for the latest release, and doesn't get any new features.

Use topic branches instead of working directly on master.

Name Your Branch After an Issue or Task

If an issue exists for the task you're going to work on, name your new branch after the issue # and description. For example, if you're working on Issue #100, a new logo, your branch name would be "100-new-logo".

If you'll be working on a task which no issue exists for, consider creating an issue for it. If you decide to go issue-less, at least give your branch a descriptive name that matches the task you'll be working on.

See also:

Could your branch name be improved? Rename your branch!

Start a New Task on a New Branch

Using git requires a certain mindset:

- **Branches** are **tasks**
- **Commits** are **subtasks**

Tasks are major changes to the codebase, such as a new feature. Tasks are usually projects of their own and require a large amount of work. A task can be broken down into subtasks. These are the smaller problems that need to be solved to make progress towards your larger task.

Each new task should have its own branch. Why?

- Your work is more organized (separate branches for separate tasks)
- Easier for everyone to see what task you're working on
- Reduces the risk of introducing new bugs
- Easier to isolate and fix new bugs
- Good for experimenting as nothing outside that branch is harmed

Properly Style Your Commit Messages

To help keep the style of our commit messages consistent and for easier viewing on GitHub, please write your commit messages in accordance with this style:

```
Capitalized and concise (50 chars or less) summary of your commit

More detailed explanatory text if necessary. Wrap at 72 characters.
Notice that the above summary message does not end with a period,
and there's a blank line between the summary and body text.

If the commit fixes an issue, start the summary line with "Fix",
followed by the issue number. E.g. "Fix #123 Add foo to bar".

- Bullet points (hyphens or asterisks) are allowed

- No ending period needed and wrap at 72 chars

- Put a space after bullet points and blank lines between them

- Use imperative, present tense: "fix", not "fixed" or "fixes"

- Add any references to related issues on GitHub if possible
```

```
Last paragraph should reference related issues and pull requests.
Fix #123
Close #321
Related to #404
```

If you can describe your commit with just a summary line, you may use git commit's message argument:

```
git commit -m "Summary of your commit (50 chars or less)"
```

CHAPTER 5

Release Engineering

When releasing a new Freeseer version use the following checklist.

1. Ensure your git repo is clean
2. Update resource files
3. • Navigate to `src/freeseer/frontend/qtcommon` and run `make`
4. `./bump_version.sh <new version>`
5. `git commit -asm "Release 3.0.0"`
6. `git push origin master`
7. Login to GitHub and navigate to a new release
8. Draft the release notes include new features and bugs fixed
9. Create python egg and source distribution:
10. • `python setup.py sdist`
11. • `python setup.py bdist_egg`
12. Push the source distribution to PyPi
13. Add the 2 binaries to the Release Draft
14. After peer review and release date agreed on Publish the release

Note: The commit message summary line should say "Bump version to release version 3.0.0" (update version as relevant).

Symbols

F