Exaile Documentation

Release 4.1.1+ac3fb49

Adam Olsen <arolsen@gmail.com>Johannes Sasongko <sasongk

Mar 28, 2021

Contents

Exaile is a music player with a simple interface and powerful music management capabilities. Features include automatic fetching of album art, lyrics fetching, streaming internet radio, tabbed playlists, smart playlists with extensive filtering/search capabilities, and much more.

Exaile is written using python and GTK+ and is easily extensible via plugins. There are over 50 plugins distributed with Exaile that include advanced track tagging, last.fm scrobbling, support for portable media players, podcasts, internet radio such as icecast and Soma.FM, ReplayGain, output via a secondary output device (great for DJs!), and much more.

CHAPTER 1

User's guide

The great thing about Exaile is that it's really simple to use, so we haven't written instructions on how to use it! ;)

1.1 Installation dependencies

The official installers for Windows and OSX should already come with/install the necessary dependencies for base functionality to work.

Note: Plugin dependencies should be listed in their description

1.1.1 Core dependencies

Essential:

- python3 >= 3.6
- python3-bsddb3
- gtk+ >= 3.22
- gstreamer (>= 1.14)
- gstreamer-plugins-base (>= 1.14)
- gstreamer-plugins-good (>= 1.14)
- python3-mutagen (>= 1.38)
- python3-dbus
- GI typelib files for GTK+, GStreamer (including gstreamer-plugins-base) and cairo and their python bindings * Packages on Debian and Ubuntu:
  - python3-gi >= 3.22

  - python3-gi-cairo
  - gir1.2-gtk-3.0
  - gir1.2-gstreamer-1.0
  - gir1.2-gst-plugins-base-1.0
  - Packages on Fedora: * python3-cairo * python3-gobject >= 3.22 * python3-gstreamer1

1.1.2 Optional dependencies

Translation:

- gettext

Documentation:

- sphinx
- sphinx_rtd_theme
- help2man

Device detection:

- udisks2

CD info: (TODO: This is currently broken on python3, see #608 and #652)

- cddb (python2), from http://cddb-py.sourceforge.net/

DAAP plugins (daapserver and daapclient):

- spydaap (python3), e.g. from https://github.com/exaile/spydaap or https://pypi.org/project/spydaap/
- python-zeroconf is an optional dependency of daapclient to enable share auto-discovery

Last.FM integration:

- pylast (python3)

Lyrics from lyricsmania.com (lyricsmania):

- lxml (python3)

Lyrics from lyrics.wikia.com (lyricwiki):

- Beautifulsoup4 (python3)
- lxml (python3)

Musicbrainz covers:

- musicbrainzngs (python3)

Podcast plugin:

- feedparser (python3)

Wikipedia info:

- webkit2gtk3 + its GI typelib

Xlib-based hotkeys:

- keybinder3 + its GI typelib

Scalable icons:

- librsvg2

Native Notifications:

- libnotify

Recording streams:

- streamripper

Moodbar plugin:

- moodbar from https://github.com/exaile/moodbar

BPM Counter plugin:

- bpmdetect from gstreamer1-plugins-bad

Test dependencies:

- mox3 (python3)
- pytest (python3)

1.2 Installation

1.2.1 Windows

On Windows, using the official Windows installation program is recommended. If you want to build your own installer, see *Windows Installer*.

1.2.2 Install on OSX

We are not currently distributing an installer for OSX. Instead, you must install Exaile via Homebrew. For more details, see the *homebrew-exaile <https://github.com/exaile/homebrew-exaile>* repo.

1.2.3 Linux/*nix

On *nix-based systems (but not OSX), run the following command from the source code directory to install:

```
$ make
# make install
```

The "make" step is optional and only compiles the modules and translations.

There is also an install_no_locale target if you wish to install without translations.

To uninstall exaile please run:

```
# make uninstall
```

from the same directory.

Environment Variables

Note: normally it should be enough to set `PREFIX`, and potentially `LIBINSTALLDIR` on a 64-bit system. The additional variables can provide further installation customization.

Environment variables that affect "make install":

LIBINSTALLDIR The full path to the lib directory. Default: `EPREFIX/lib`

DATADIR The full path to non-arcitecture dependent data files. Default: `PREFIX/share`

MANPREFIX The full path to the parent of the man page installation directory (same as system DATADIR) Default: `PREFIX/share`

EPREFIX The executable installation prefix. Default: `PREFIX`

PREFIX The main installation prefix. Default: `/usr/local` **Note**: If this default is changed, you may need to set the LIBINSTALLDIR or XDG_DATA_DIRS environment variables to include the changed path. See [1].

XDGCONFDIR The directory to install system-wide config files in, following xdg spec. Default: `/etc/xdg` **Note**: If this default is changed, you may need to set the XDG_CONFIG_DIRS environment variable to include the changed path. See [1].

DESTDIR Destination directory to put the file structure under. Mainly of use for packagers. Default: not set (`/`)

DEFAULTARGS Default arguments that the 'exaile' launcher script (installed to `$PREFIX/bin/exaile`) will pass to exaile on startup

Additional Resources: 1. XDG Base Directory Specification

1.3 Windows Notes

We're excited to announce that Exaile has official support for Windows platforms as of version 3.3.0, and is distributed in an official installer program.

Exaile (and its installer) has been tested on:

- Windows 7 x64 SP1
- Windows 8.1
- Windows 10
- Python 2.7.10 (32-bit and 64-bit)
- PyGObject 3.14 + GTK+ 3.14
- GStreamer 1.4.5

Exaile now requires GStreamer 1.x and GTK3. The best way to install these requirements on Windows is using the All-In-One PyGI/PyGObject for Windows Installer. When using this installer, you must select the following options:

- GTK 3.x
- GStreamer 1.4.5
- The gst-plugins options you desire (recommend installing them all)

1.3.1 Running Exaile

If you installed exaile using the Windows installer, you can find icons to launch Exaile in the Start Menu under "All Programs" -> "Exaile".

Alternatively, you can directly launch "exaile_win.py".

1.3.2 Bug Reporting

If exaile crashes, you should be able to find a logfile from the last time exaile was launched in the following directories:

- Windows XP: `%USERPROFILE%\Local Settings\Application Data\exaile\logs`
- Windows 7+: `%USERPROFILE%\AppData\Local\exaile\logs`

If you run into any bugs not mentioned in this document, please report them to us via GitHub. Please include any relevant logfile snippets.

1.4 OS X Notes

Exaile has experimental support for OS X as of version 3.4.0. There are a number of known issues (including crashes) with Exaile on OS X, but these are most likely issues with GTK's OS X support. We hope that as time goes on the most critical bugs will go away.

1.4.1 Requirements

We are not currently distributing an installer for OSX. Instead, you must install Exaile via Homebrew. For more details, see the *homebrew-exaile <https://github.com/exaile/homebrew-exaile>* repo.

1.4.2 Known issues

The OS X version is about as functional as the Windows version, so most things will work without any problems. The CD plugin and other device plugins will not work on OS X.

Transparency may not work.

1.5 Frequently Asked Questions

1.5.1 Error "no suitable plugin found" when playing a (.mp3, .m4a, etc)

Exaile 4.x currently uses GStreamer 1.x to decode/play audio files, and does not directly decode audio itself. For playback to work, you need to have the correct GStreamer plugins installed.

Note: For Linux users, you may find that other GStreamer programs can play a specific file type, but Exaile cannot. Check to make sure that the correct plugins are installed for GStreamer 1.x, as other players may be using GStreamer 0.10 instead.

1.5.2 File tags don't update when I change them using an external program

When setting up your collection, ensure that the 'monitored' and 'scan on startup' options are checked, otherwise Exaile may become out of sync with your collection if it is modified by external programs.

To detect that the file has changed, Exaile checks to see if the modification time of the file has changed. This makes rescans much quicker.

Some third-party taggers (notably EasyTag) have options where they do not update the modification time of the file when they change the contents of the file. In these cases, Exaile may not be able to detect that the file has changed. To remain compatible with Exaile (and other media players), you should configure your tagger to update the modification time.

Note: As of Exaile 3.4.2, there is a menu option called 'Rescan Collection (slow)' which will force a rescan of every file in your collections, regardless of whether the modification time has changed. This should detect any changes to your collection.

1.5.3 How do I enable output to a secondary soundcard?

A: Enable the 'preview device' plugin. You can change the secondary output device settings by editing the plugin's settings.

1.5.4 Output switches to my primary output when I disconnect my secondary output?!

This issue occurs with some types of audio sinks that are available from GStreamer. In particular, on many systems the PulseAudio system is configured to automatically fallback to the another output if a stream is playing and its output fails. Exaile's 'preview device' plugin has a hack that partially solves this, but to truly solve it you have to convince your system to not do this.

On a system with PulseAudio, you must edit `/etc/pulse/default.pa` and comment out the following line:

```
load-module module-rescue-streams
```

CHAPTER 2

Developer's guide

2.1 Getting Started

You want to hack on Exaile? Rock on! The documentation for getting started isn't as complete as we'd like (please help us improve it!), but this should help you on your way:

2.1.1 Setting up a Development Environment

Because Exaile is written in Python, if you can run Exaile on your machine, then you can easily modify the source code and run that too. The Exaile developers mostly run Exaile directly from the git checkout, without installing Exaile:

```
git clone https://github.com/exaile/exaile.git
cd exaile
# On Linux, Mac OS X or *BSD:
./exaile

# On Windows:
exaile.bat
```

If that works, then you're ready to go! If not, you need to install Exaile's various dependencies:

Linux

On Ubuntu 16.04 following apt-get command should install most of the needed dependencies:

```
sudo apt-get install \
  python3-mutagen \
  python3-gi \
  python3-gi-cairo \
  python3-dbus \
```

(continues on next page)

(continued from previous page)

```
  gir1.2-gtk-3.0 \
  gir1.2-gstreamer-1.0 \
  gir1.2-gst-plugins-base-1.0 \
  gstreamer1.0-plugins-base \
  gstreamer1.0-plugins-good \
  gstreamer1.0-plugins-ugly \
  gstreamer1.0-plugins-bad
```

Windows

First, install msys2. Then, open the MinGW32 shell window (look in the Start Menu for it), and run this monster (it may take awhile):

```
pacman -S \
  mingw-w64-i686-python3-gobject \
  mingw-w64-i686-python3-cairo \
  mingw-w64-i686-python3-pip \
  mingw-w64-i686-python3-bsddb3 \
  mingw-w64-i686-gtk3 \
  mingw-w64-i686-gdk-pixbuf2 \
  mingw-w64-i686-gstreamer \
  mingw-w64-i686-gst-plugins-base \
  mingw-w64-i686-gst-plugins-good \
  mingw-w64-i686-gst-plugins-bad \
  mingw-w64-i686-gst-libav \
  mingw-w64-i686-gst-plugins-ugly
```

Once that is complete, you'll want to install mutagen:

```
python3 -m pip install mutagen
```

And then you should be able to launch Exaile from the msys2 console:

```
cd exaile
python3 exaile_win.py
```

OSX

The Python GTK3 GStreamer SDK repo can be used to install an appropriate environment for OSX, and has instructions for setting it up:

- https://github.com/exaile/python-gtk3-gst-sdk/tree/master/osx_bundle

Other instructions

See the PyGObject Getting Started documentation for getting the core PyGObject stuff installed. Once you get that working, then you just need to use the appropriate package manager to install GStreamer and things should be good to go.

Once you get pygobject working, you will also want to install mutagen via pip:

```
python -m pip install mutagen
```

2.1.2 Useful documentation

Exaile is built upon Python, PyGObject, Gtk+, and GStreamer. Here is a bunch of documentation that you will find useful when working with these frameworks:

- Python 3
- PyGObject
- Python GI API Reference
- Python GTK+3 Tutorial
- ABI/API tracker for tracking incompatible changes in C/C++ ABI and API

2.1.3 Useful tools

- Glade is what we use to edit the 'ui' xml files that describe our UI layout.

> **Warning:** Glade historically has been very prone to crashing, so when using it save your work often!

2.1.4 Editor setup

Atom

I've found recent versions of Github's Atom editor to be very useful for Python development, I recommend installing the `autocomplete-python` and `linter-pyflakes` packages.

Eclipse + pydev

Pydev can be a bit tricky to set up correctly, see its documentation for details.

- Ensure you add the correct python interpreter in the project settings
- Add the root of the repository as a source directory

2.1.5 Running the tests

If you have pytest installed, then you can just run:

```
make test
```

2.2 Contributing to Exaile

The exaile team is always looking for others to help contribute to exaile in various ways:

- Bugfixes
- Documentation updates
- New features + plugins
- Translations on https://hosted.weblate.org/engage/exaile/

The best way to contribute the first three is to submit patches via pull request on GitHub.

If you think your bug report/request is being ignored, it probably isn't. All of the Exaile developers work on this project in their spare time, and so we don't always have time to work on specific problems. We *do* try to push good patches as soon as we can, however. Ping the bug report, or leave a message on #exaile if we haven't at least made an initial response, sometimes bug report emails can get lost in the noise or forgotten.

2.2.1 Translating Exaile

Translations for Exaile should be done on Exaile's Weblate project. If you are new to Weblate, you may find the Weblate translators guide useful.

Python string formatting

Python has two ways of specifying string formatting options.

With %. This method has several possible variations. Some examples:

- "Downloading %s" (a string)
- "Track number: %d" (an integer)
- "Size: %.2f MB" (a floating-point number, rounded to 2 decimal points)
- "Editing track %(current)d out of %(total)d" (two integers with disambiguating labels)

With {}. These are equivalent to the above examples:

- "Downloading {}"
- "Track number: {}"
- "Size: {:.2f} MB"
- "Editing track {current} out of {total}"

If you find two placeholders in one string with no labels to disambiguate them, for example if you see "The %s has %d items" or "Loading: {} out of {}", please report it as a bug. This is because in some languages it may be necessary to reorder elements of the string, which is impossible to do with both examples.

GTK+ keyboard mnemonics

An underscore (_) character in a GTK+ menu string indicates the keyboard mnemonic for that menu item. For example, the File menu is written as "_File" and the Open menu item is written as "_Open", which then allows the user to access the File→Open menu item by pressing `Alt+F, O`. You are encouraged to change these mnemonics to match existing conventions in your language and to avoid conflicting mnemonics within the same menu.

2.3 Debugging Exaile

Contents

2.3.1 What's an easy way to test stuff without wrecking my actual collection?

If you use the `--all-data-dir` option to Exaile, it will store all data for that execution of Exaile in that directory (collections, playlists, logs):

```
./exaile --all-data-dir=tmp
```

2.3.2 Debugging options for Exaile

See `--help` for more details, but there are a few useful options:

- `--debug` - Shows debug log messages
- `--eventdebug` - Enable debugging of xl.event. Generates lots of output
- `--eventdebug-full` - Enable debugging of xl.event. Generates LOTS of output
- `--threaddebug` - Adds the thread name to logging messages

2.3.3 Where can I find log files?

On Exaile 4, you can click the 'Open error logs' button in the 'Help' menu and it will open the directory where logs are stored.

On Linux/OSX:

- `~/.local/share/exaile/logs/` for Exaile 3.x+ releases

On Windows:

- `%APPDATA%\..\Local\exaile

2.3.4 Viewing stack traces when Exaile hangs

On Linux/OSX if you send SIGUSR2 to Exaile it will dump stacktraces of all current Python threads to stderr.

2.3.5 GStreamer Debugging Techniques

When tracking down GST issues, a useful thing to do is the following:

```
$ GST_DEBUG=3 ./exaile
$ GST_DEBUG="cat:5;cat2:3" .. etc.

$ GST_DEBUG="GST_STATES:4" ./exaile
```

`GST_DEBUG_NO_COLOR=1` is good if you're running exaile inside of pydev on eclipse.

Additional help about GStreamer debugging variables can be found in its Documentation

GST Bin Visualization

This is pretty cool, shows you the entire GST pipeline:

```
Gst.debug_bin_to_dot_file(some_gst_element, Gst.DebugGraphDetails.ALL, "filename")
```

Then if you run exaile like so:

```
GST_DEBUG_DUMP_DOT_DIR=foo ./exaile
```

It will dump a dot file that you can turn into an image:

```
dot -Tpng -oimage.png graph_lowlevel.dot
```

2.3.6 Using GDB to diagnose issues

Preparing GDB

Please make sure that you have installed debug symbols for all essential non-python packages listed in *Installation dependencies*. Python packages do not need debug symbols, because they ship both binary and source files already. Depending on the distribution you are using, you may obtain debug symbols in different ways.

- Fedora: Run `dnf debuginfo-install [packagename]` as root or with sudo. Fedora also ships a *C/C++ Debugger* with the Eclipse CDT (`eclipse-cdt`) package, which provides a useful GUI.
- Debian, Ubuntu, Linux Mint: Have a look at the wiki pages Backtrace and DebuggingProgramCrash
- Arch Linux

Basic Usage

GDB can be used to diagnose segfaults and other issues. To run GDB:

```
gdb --args python3 exaile.py --startgui <other arguments here>
```

Refer to the Python Documentation, but especially useful here are:

- `(gdb) py-bt` is similar to `(gdb) bt`, but it lists the python stack instead
- `(gdb) info threads`

Tips for debugging issues related to Gtk+ or GLib

Refer to the Gtk+ and GLib debugging documentation.

In particular, the GTK+ Inspector is very useful. On GTK 3.14+, hit CTRL-SHIFT-D or CTRL-SHIFT-I to bring up GtkInspector to help debug UI problems. If the hotkeys don't work, run Exaile with GTK_DEBUG=interactive. (On Gtk=3.18 this sometimes causes GtkDialogs to crash on closing.)

Enable diagnostic warnings

On GLib >= 2.46 you might want to set the `G_ENABLE_DIAGNOSTIC` environment variable to show deprecation warnings. They are disabled by default since 2.46 and sometimes on older versions. See this commit.

Eliminating Gtk-WARNING

1. run gdb with `G_DEBUG=fatal-warnings gdb --args python3 exaile --startgui`
2. run exaile from gdb with `run`
3. do whatever causes *Gtk-WARNING*. This will lead to a crash in exaile.
4. debug this crash with gdb

WARNING: On Linux, this will freeze your X server if the crash happens in a menu. This is due to X grabbing all input on open menus. When gdb stops exaile inside a menu it can't leave the input grab.

Prevent X server from freezing your Desktop when debugging exaile

Some recommend starting exaile on another X server or on a Wayland backend. One way to workaround this is to run exaile on a nested X server inside weston:

1. install weston
2. run `weston --modules=xwayland.so` (note: from now on all your Gtk+ 3.x applications will try to start inside weston due to preferring Wayland over X)
3. inside weston, run `env | grep DISPLAY` to figure out which X11 display to start exaile on
4. before running gdb, add `GDK_BACKEND=x11` and `DISPLAY=:1` (or whatever you got the step before) to its environment

To make Gtk+ 3.x applications not run inside weston but use your current X11 desktop session, run them with `GDK_BACKEND=x11` environment variable set.

Debugging segfaults (segmentation violations)

1. Open a terminal.
2. Use the `cd` command to change to the directory where you put Exaile source code or to its installation directory.
3. Run `gdb /usr/bin/python3`
4. In gdb, run `set logging on exaile-segfault.txt` to enable logging to that file.
5. In gdb, run `run ./exaile.py --startgui`. You might append other arguments if you need them.
6. Use Exaile as you did before and try to reproduce the problem. At some point, exaile might freeze. This is when gdb catched the segmentation fault.
7. In gdb, run `t a a py-bt` and `t a a bt full`. The first one will get python backtraces from all threads, the second one will get native (C/C++) stacktraces. You might need to type the return key a few times after each of these two commands to make gdb print all lines of the stack traces. This might take a while.
8. In gdb, type `quit` and press the enter key.
9. Please attach the file `exaile-segfault.txt` to a bug report at Github after you checked that it does not contain any private data. If you prefer to send the data encrypted, please feel free to encrypt them to the PGP key ID 0x545B42FB8713DA3B and send it to one of its Email addresses.

Debugging freezes

If Exaile freezes, follow the steps above for debugging segfaults but attach to the running instance instead.

1. Get the PID of Exaile. You may want to use `top`, `htop`, *KSysGuard* or *GNOME System Monitor* or a similar tool.
2. Follow the steps above, with one change: Instead of starting `run ./exaile.py --startgui`, run the `attach [pid]` command inside gdb to attach to the exaile instance with the PID you retrieved in the previous step.

Debugging ignored exceptions

Sometimes, especially when shutting down, Exaile may print a message like this:

```
Exception TypeError:  "'NoneType' object is not callable" in <object
repr() failed> ignored
```

You may see this output when the python runtime ran into an exception when calling *__del__* on an object or during garbage collection. This output is generated by `PyErr_WriteUnraisable` in python's `errors.c`. To debug it, attach gdb to Exaile or start Exaile in gdb and run `break PyErr_WriteUnraisable`. Instead of writing the above message, gdb should break at the specified function and you should be able to get a backtrace.

2.3.7 Other thoughts

Exaile is written using Gtk+, GStreamer, and Python. Any generally useful debugging tips that apply to those environments will often apply to Exaile also. Quod Libet is another audio player uses Gtk/GStreamer and Python, their development documentation also has useful debugging information:

- Quod Libet Useful Development Tools

2.4 Code guidelines

Page to hold style and practice guidelines for contributions to Exaile. Patches to make the existing core codebase follow these guidelines are always welcome and a good way to start learning about the internal workings of Exaile.

Note that this document will generally reflect the 'trunk' version of Exaile, and may not be fully applicable to stable releases. If in doubt, ask!

2.4.1 Basic Style

Exaile uses the black code formatter to enforce a consistent style across the project. You can run black like so:

```
make format
```

For things that the code formatter doesn't do for you, the following applies:

- In general, PEP 8 applies
- Keep imports on one line each to make sure imports cannot be missed:

```
# Not recommended
from gi.repository import Gtk, GLib, GObject

# Preferred
from gi.repository import Gtk
from gi.repository import GLib
from gi.repository import GObject
```

- Always write out variable names to keep them descriptive. Thus `notebook_page` is to be preferred over `nb`.
  - Exceptions:
    - Names which are prone to spelling mistakes like `miscellaneous` and `utilities`. Here `misc` and `util` are perfectly fine.
    - If a very-long-named (like fooooo.bar_baz_biz_boz) variable or function needs to be accessed by a large percentage of lines in a small space, it may be shortened as long as 1) the name it is shortened to is consistent across all uses of this shortcut, and 2) the shortcut is limited in scope to just the area where it is used repeatedly. If in doubt, do NOT use this exception.
- Try to group related methods within a class, this makes it easier to debug. If it's a particularly significant group of methods, mark them with a triple-comment at the beginning and end, like so:

```
### Methods for FOOBAR ###
## more-detailed description (if needed)
def meth1(self):
    ...

### End FOOBAR ###
```

- The closing triple-comment may be omitted if at the end of a class or if another triple-comment starter comes after it.
- If you need a collection of constants for some purpose, it is recommended to use the `enum` function from `xl.common` to construct one. The constant type should be UpperCamelCase, the possible values UPPERCASE:

```
from xl.common import enum

ActionType = enum(ADD='add', EDIT='edit', ...)

# ...

if action.type == ActionType.EDIT:
    # ...
```

2.4.2 Documentation

- Always add docstrings to your public classes, methods and functions.
- Follow the Sphinx format for documentation within docstrings.

2.4.3 Events and Signals

- Items internal to Exaile (ie. anything under `xl/`) should generally prefer `xl.event` over `GObject` signals. Items that tie deeply into the (GTK) UI should prefer `GObject` signals over `xl.event`.
- Keep in mind all events are synchronous - if your callback might take a while, run it in a separate thread.
- – Make sure that every access to GTK UI components is run in the GTK main thread. Otherwise unpredictable issues can occur including crashes due to cross-thread access. This can be accomplished by running the specific code through the GLib.idle_add function. Please use the function decorator `common.idle_add`. A typical mistake:

```
def __init__(self):
    """
        Set up a label in the GTK main thead and
        connect to the playback_track_start event
    """
    self.label = Gtk.Label()
    event.add_callback(self.on_playback_track_start, 'playback_track_start')

def on_playback_track_start(event, player, track):
    """
        Serious problem: this event is run in a
        different thread, a crash is likely to occur
    """
    self.label.set_text(track.get_tag_display('title'))
```

- Event names should be all lower-case, using underscores to separate words.
  - Names should be prefixed by the general name indicating the category or sender of the event. For example, events sent from `xl.player` start with a `playback_` prefix.
  - The remainder of the name should indicate what action just happened. eg. `playback_player_pause`.
  - The data sent in an event should be whatever piece (or pieces) of data are most relevant to the event. For example, if the event is signaling that a state has changed, the new state should be sent, or if the event indicates that an item was added, the new item should be sent.
- Callbacks for `GObject` and `xl.event` should always be named "`on_`" + the name of the event. This avoids confusion and draws a line between regular methods and signal/event callbacks.

- If you need to handle the same signal/event for multiple objects but differently (as in: different callbacks), include the name of the object in the callback name. Thus the event "`clicked`" for the `Gtk.Button` "`play_button`" would become "`on_play_button_clicked`". A small exception to this rule is when a word would be repeated. Thus "`on_play_button_press_event`" should be preferred over "`on_play_button_button_press_event`" for the "`button-press-event`" signal of the button.
- If you use Gtk.Builder for UI descriptions, apply the rules above, make the callbacks methods of your class and simply call `Gtk.Builder.connect_signals(self)`

2.4.4 Managed object access

- To keep classes interchangeable, try to make use of existing signals/events wherever possible. Avoid reaching deeply into property hierarchies under all circumstances. This is bound to break sooner than later.
- If you need access to the main *exaile* object, call `xl.main.exaile()`, if you need access to the main GUI object, call `xlgui.get_controller()`, for the main window `xlgui.main.mainwindow()`
- Many systems are already ported to singleton managers. Examples are `xl.covers` and `xlgui.icons`. Simply use their `MANAGER` property to access them.

2.4.5 GUI

- Use .ui files to define most widgets - reduces code clutter. A lot of basic structure can be easily prepared with the Glade interface designer, especially objects where cell renderers and models are involved.
- Try to avoid dialogs, as they are intrusive and users generally don't read them anyway. Inline alternatives like Gtk.InfoBar and its convenience wrapper `xlgui.widgets.dialogs.MessageBar` are much more effective.

2.4.6 Logging

- Messages should
  - Be short but descriptive.
  - Be proper English sentences, minus the period.
  - Happen after the thing they are logging, UNLESS the thing might take a while, in which case it may be printed before, with a confirmation after the action completes.
    - The tense of the message should match when it's sent - if after the action, use the past tense ("Logged into Audioscrobbler"), if before, use the present(?) tense ("Logging into audioscrobbler...").
    - Messages which are present tense may use an ellipsis ("...") to indicate the different state more clearly than by tense alone.
  - Not be given prefixes to identify module, as –debug will automatically add module names. It is acceptable to use related names in the message to increase clarity however. For example, "Logged into Audioscrobbler" is much clearer than "Logged in", but "Audioscrobbler: Logged in" is not acceptable.
- There are 4 standard logging levels built into Exaile, their names and purpose are as follows:
  - DEBUG - A significant internal event happened. Not shown by default.
  - INFO - A major but expected event happened.
  - WARNING - Something suboptimal happened. Exaile will continue to work properly but some features may be unavailable.

  - ERROR - A critical error occurred. Exaile was unable to perform a requested action and may be in an inconsistent state if the error was not fully handled.

- When writing messages, please run both with and without –debug to ensure it looks correct and does not duplicate the information provided by any other message.
- Be sparing in the use of logging messages, particularly non-DEBUG messages. Logging messages are not an alternative to inserting print statements when debugging!

2.4.7 Other

- If you create a new on-disk format, add a version flag to it. This makes forwards and backwards compatibility MUCH easier should the format ever need to change.

2.5 Plugin Development Guide

Note: these instructions always track current Exaile trunk, and may not be fully compatible with stable releases. It is recommended that you develop plugins against trunk, so that you can submit patches to trunk if need be during the creation of your plugin, and so that your plugin can easily be merged into trunk when it is ready.

2.5.1 Style

If you plan to submit your plugin for inclusion in Exaile, please read and follow the guidelines in the *Code guidelines*

2.5.2 Basic plugin structure

Plugins in Exaile 3.x+ are handled slightly differently than in the past. Each plugin has its own directory in `~/.local/share/exaile/plugins/`. In order for your plugin to be recognized as valid by Exaile, it needs to have at least two files in the plugin directory (`~/.local/share/exaile/plugins/myplugin/`):

- `__init__.py`
- `PLUGININFO`

The format of the `PLUGININFO` is as follows:

```
Version='0.0.1'
Authors=['Your Name <your@email.com>']
Name=_('Plugin Name')
Description=_('Something that describes your plugin. Also mention any extra␣
↪dependencies.')
Category=_('Development')
```

The following two attributes are optional:

- *Platforms* - A list of the platforms your plugin works on. If you have no specific requirements, omitting this argument or using an empty list is fine. The values of the list are the sys.platform value.
- *RequiredModules* - A list of additional modules required by your plugin. Modules that Exaile already require (e.g. mutagen) don't need to be specified. To specify GObject Introspection libraries, prefix it with `gi:`, e.g. `gi:WebKit2`.

Note: Name and Description are what show up in the plugin manager. Category is used to list your plugin alongside other plugins. Platforms and RequiredModules are used to filter out the plugin on inappropriate platforms.

Before Exaile 3.4, `__init__.py` was required to define at least two methods, `enable()` and `disable()`. However, Exaile 3.4 introduced a new way to write plugins which will eliminate a lot of unnecessary boilerplate for plugin authors. We will use this model below:

```
class MyPlugin:

    def enable(self, exaile):
        print('You enabled me!')

    def disable(self, exaile):
        print('I am being disabled')


plugin_class = MyPlugin
```

For many types of plugins, this might be enough. However, there are other optional methods you can define in your plugin object.

- `on_gui_loaded` - This will be called when the GUI is ready, or immediately if already done
- `on_exaile_loaded` - This will be called when exaile has finished loading, or immediately if already done
- `teardown` - This will be called when exaile is unloading

These methods may be necessary for your plugin because plugins can only access Exaile's infrastructure when Exaile itself finishes loading. The first `enable()` method is called when Exaile is partway through loading. But since we can't do anything until Exaile finishes loading, we can add `on_exaile_loaded` to our object that is called when Exaile finishes loading. Some plugins need to modify state earlier in the startup process, hence the need for this separation.

The `exaile` object in the above example is an instance of a class called Exaile, which is defined in `xl/main.py`. This class is a base for everything in the program.

You can get a handle on various objects in Exaile by looking at the members of this class.

2.5.3 Something (slightly) more useful

Here is an example of a plugin that will, when a track is played, show the track information in a `MessageDialog`. It demonstrates a callback on an event, and getting the Gtk.Window object of Exaile to use as a parent for a MessageBox.

The `PLUGININFO` is as follows:

```
Version='0.0.1'
Authors=['Me <me@internets.com>']
Name='Tutorial Plugin'
Description='Plugin to demonstrate how to make a plugin.'
```

and the `__init__.py` is as follows

```
'''
    This plugin will show an obnoxious Gtk.MessageDialog that
    won't disappear, when a track is played. The MessageDialog
    will contain the information of the currently playing track.
```

(continues on next page)

(continued from previous page)

```
'''

from xl import event
from gi.repository import Gtk

# The main functionality of each plugin is generally defined in a class
# This is by convention, and also makes programming easier
class TutorialPlugin:

    def enable(self, exaile):
        '''This method is called when the plugin is loaded by exaile'''

        # We need a reference to the main Exaile object in order to set the
        # parent window for our obnoxious MessageDialog
        self.exaile = exaile

    def disable(self, exaile):
        '''This method is called when the plugin is disabled. Typically it is used for
           removing any GUI elements that we may have added in _enable()'''
        self.show_messagebox("Byebye!")

    def on_exaile_loaded(self):
        '''Called when exaile is ready for us to manipulate it'''

        #The reason why we dont use show_messagebox here is it hangs the GUI
        #which means it would hang Exaile as soon as you restart, because all
        #enabled plugins are loaded on start.
        print('You enabled the Tutorial plugin!')

        # Add a callback for the 'playback_track_start' event.
        # See xl/event.py for more details.
        event.add_callback(self.popup_message, 'playback_track_start')


    def popup_message(self, type, player, track):
        # The Track object (defined in xl/track.py) stores its data in lists
        # Convert the lists into strings for displaying
        title = track.get_tag_display('title')
        artist = track.get_tag_display('artist')
        album = track.get_tag_display('album')
        message = "Started playing %s by %s on %s" % (title, artist, album)
        self.show_messagebox(message)

    def show_messagebox(self, message):
        # This is the obnoxious MessageDialog. Due to (something to do with threading?
↪)
        # it will steal, and never relinquish, focus when it is displayed.
        dialog = Gtk.MessageDialog(self.exaile.gui.main.window, 0,
                                   Gtk.MessageType.INFO, Gtk.ButtonsType.OK, message)
        dialog.run()
        dialog.destroy()


plugin_class = TutorialPlugin
```

Have a look in the comments for an explanation of what everything is doing.

2.5.4 Adding a track to the Playlist

This is relatively simple. A Playlist consists of the actual graphical representation of a playlist (see `xlgui/playlist.py`) and its underlying Playlist object (see `xl/playlist.py`). Any changes made to the underlying playlist object are shown in the graphical representation. We will be appending Track objects to this underlying playlist.

First you need to get a handle on the underlying Playlist:

```
playlist_handle = exaile.gui.main.get_selected_playlist().playlist
```

Then, you need to create a Track object (defined in `xl/track.py`). The method to do this from a local file versus a URL is slightly different.

For a local source:

```
from xl import trax
path = "/home/user/track.ogg" #basically, just specify an absolute path
myTrack = trax.Track(path)
```

For a url:

```
from xl import trax
url = "http://path/to/streaming/source"
myTrack = trax.get_tracks_from_uri(url)
```

You can set the track information like this:

```
myTrack.set_tags(title='Cool Track',
                 artist='Cool Person',
                 album='Cool Album')
```

Once you have a Track object, and a handle on the Playlist you would like to add the track to, you can proceed to add the track:

```
playlist_handle.add(myTrack)
```

Note that `get_tracks_from_uri()` returns a list, so you will need to use the method for adding multiple tracks if your Track object was created this way. You can also create your own list of Track objects and add them all in one go like this too:

```
playlist_handle.add_tracks(myTrack)
```

This is pretty much all you need to do to add a track to the playlist. An example in a plugin might be:

```
from xl import event, trax

class PlaylistExample:

    def enable(self, exaile):
        self.exaile = exaile

    def disable(self, exaile):
        pass

    def on_gui_loaded(self):
        self.playlist_handle = self.exaile.gui.main.get_selected_playlist().playlist
```

(continues on next page)

(continued from previous page)

```
        local_tr = self.create_track_from_path('/home/user/track.ogg')
        remote_tr = self.create_track_from_url('http://site.com/track.ogg')
        self.add_single_to_playlist(local_tr)
        self.add_multiple_to_playlist(remote_tr)

    def create_track_from_path(self, path):
        return trax.Track(path)

    def create_track_from_url(self, url):
        return trax.get_tracks_from_uri(url)

    def add_single_to_playlist(self, track):
        self.playlist_handle.add(track)

    def add_multiple_to_playlist(self, tracks):
        self.playlist_handle.add_tracks(tracks)


plugin_class = PlaylistExample
```

You can do more things when adding a track than simply specifying a track object to add, see the methods in the class Playlist (`xl/playlist.py`) for more details.

2.5.5 Adding another page to the left-hand Notebook

This is done pretty easily. Basically, you need to subclass `xlgui.panel.Panel` and register a provider advertising your panel.

The subclass needs to have two attributes:

- `ui_info` - This defines the location of the .glade file that will be loaded into the notebook page (This file must be in Gtk.Builder format, not glade format)
- `name` - This is the name that will show on the notebook page, such as "MyPlugin"

```
from xl import providers
from xlgui import panel

# Note: The following uses the exaile object from the enable() method. You
# might want to call this from the on_gui_loaded function of your plugin.
page = MyPanel(exaile.gui.main.window)
providers.register('main-panel', page)

# to remove later:
providers.unregister('main-panel', page)

class MyPanel(panel.Panel):

    #specifies the path to the gladefile (must be in Gtk.Builder format) and the name␣
↪of the Root Element in the gladefile
    ui_info = (os.path.dirname(__file__) + "mypanel_gladefile.glade",
↪'NameOfRootElement')

    def __init__(self, parent):
        panel.Panel.__init__(self, parent)
```

(continues on next page)

(continued from previous page)

```
        #This is the name that will show up on the tab in Exaile
        self.name = "MyPlugin"

        #typically here you'd set up your gui further, eg connect methods to signals␣
↪etc
```

That's pretty much all there is to it. To see an actual implementation, have a look at `xlgui/panel/collection.py` or take a look at the Jamendo plugin.

2.5.6 Setting the cover art for a track

This is done by subclassing `CoverSearchMethod` and adding and instance of the subclass the existing list. When Exaile plays a track with no cover, it uses all the methods in its `CoverSearchMethod` list to try and find a cover.

A `CoverSearchMethod` must define:

- `name` - The name of the `CoverSearchMethod`, used for removing it from the list once its been added
- `type` - The type of the `CoverSearchMethod` (local, remote)
- `find_covers(self, track, limit=-1)` - This is the method that is called by Exaile when it utilises the `CoverSearchMethod`. This method must return an absolute path to the cover file on the users harddrive.

Here is an example CoverSearchMethod (taken from the Jamendo plugin). It searches Jamendo for covers, downloads the cover to a local temp directory and returns the path to the downloaded cover.

```
import urllib.request
import hashlib
from xl.cover import CoverSearchMethod, NoCoverFoundException

class JamendoCoverSearch(CoverSearchMethod):
    name = 'jamendo'
    type = 'remote'

    def __init__(self):
        CoverSearchMethod.__init__(self)

    def find_covers(self, track, limit=-1):
        jamendo_url = track.get_loc_for_io()

        cache_dir = self.manager.cache_dir
        if (not jamendo_url) or (not ('http://' and 'jamendo' in jamendo_url)):
            raise NoCoverFoundException

        #http://stream10.jamendo.com/stream/61541/ogg2/02%20-%20PieRreF%20-
↪%20Hologram.ogg?u=0&h=f2b227d38d
        split=jamendo_url.split('/')
        track_num = split[4]
        image_url = jamapi.get_album_image_url_from_track(track_num)

        if not image_url:
            raise NoCoverFoundException

        local_name = hashlib.sha1(split[6]).hexdigest() + ".jpg"
        covername = os.path.join(cache_dir, local_name)
        urllib.request.urlretrieve(image_url, covername)
```

(continues on next page)

(continued from previous page)

```
        return [covername]
```

You can then add it to the list of `CoverSearchMethods` for Exaile to try like this:

```
exaile.covers.add_search_method(JamendoCoverSearch())
```

And remove it like this:

```
exaile.covers.remove_search_method_by_name('jamendo')
```

2.5.7 Make strings translatable

Every message should be written in English and should be translatable. The following example shows how you can make a string translatable:

```
from xl.nls import gettext as _
print(_('translatable string'))
```

2.5.8 Saving/Loading arbitrary settings

This is quite easy. It's probably quicker to just show some code instead of trying to explain it:

```
from xl import settings

#to save a setting:
setting_value = 'I am the value for this setting!'
settings.set_option('plugin/pluginname/settingname', setting_value)

#to get a setting
default_value = 'If the setting doesnt exist, I am the default value.'
retrieved_setting = settings.get_option('plugin/pluginname/settingname', default_
↪value)
```

That's all there is to it. There is a few restrictions as to the datatypes you can save as settings, see `xl/settings.py` for more details.

2.5.9 Searching the collection

The following method returns an list of similiar tracks to the current playing track:

```
exaile.dynamic.find_similar_tracks(exaile.player.current, 5) #the second optional␣
↪argument is the limit
```

This method returns an list of tuples, which consist of the match rate and the artist's name:

```
exaile.dynamic.find_similar_artists(exaile.player.current)
```

If you would like to search the collection for a specific artist, album or genre, you can use the following code:

```
from xl.trax import search

artist = 'Oasis'
tracks = [x.track for x in search.search_tracks_from_string(
          exaile.collection, ('artist=="%s"'%artist))]

genre = 'pop'
tracks = [x.track for x in search.search_tracks_from_string(
          exaile.collection, ('genre=="%s"'%genre))]

album = 'Hefty Fine'
tracks = [x.track for x in search.search_tracks_from_string(
          exaile.collection, ('album=="%s"'%album))]
```

You can search the collection also for different assignments, like the last played tracks, the most recently added tracks or the tracks, which were played most often. Here you see an example to display the most recently added tracks:

```
from xl.trax import search
from xl.trax.util import sort_tracks

tracks = [x.track for x in search.search_tracks_from_string(exaile.collection, ('!
↪%s==__null__' % '__last_played'))]
tracks = sort_tracks(['__last_played'], tracks, True) #sort the tracks by the last␣
↪playing
```

The other keywords are `__date_added` and `__playcount`

2.5.10 Exaile D-Bus

Here is a simple example how to use the D-Bus object:

```
#!/usr/bin/env python3

from io import BytesIO
import sys

import dbus
import Image

def test_dbus():
    bus = dbus.SessionBus()
    try:
        remote_object = bus.get_object("org.exaile.Exaile","/org/exaile/Exaile")
        iface = dbus.Interface(remote_object, "org.exaile.Exaile")
        if iface.IsPlaying():
            title = iface.GetTrackAttr("title")
            print('Title:', title)
            album = iface.GetTrackAttr("album")
            print('Album:', album)
            artist = iface.GetTrackAttr("artist")
            print('Artist:', artist)
            genre = iface.GetTrackAttr("genre")
            print('Genre:', genre)
            dbusArray = iface.GetCoverData()
            coverdata = bytes(dbusArray)
            if coverdata:
```

(continues on next page)

(continued from previous page)

```
                im = Image.open(BytesIO(coverdata))
                im.show()
        else:
            print("Exaile is not playing.")
    except dbus.exceptions.DBusException:
        print("Exaile is not running.")

if __name__ == "__main__":
    test_dbus()
```

Please check out `xl/xldbus.py` for further method signatures.

2.5.11 Playback events

Since playback events can occur far before the main GUI object or even the `exaile` object is loaded, connecting to them in advance is required. To do this, in your `__init__` method:

```
event.add_callback(self.on_playback_player_start, 'playback_player_start')
```

2.5.12 Distributing the Plugin

Create a Plugin Archive

Basically, you just need to tar up your plugin's directory, and rename the tarfile to <name_of_plugin_directory>.exz

You will need to develop your plugin with a similar hierarchy to the following:

```
root --
     \ -- __init__.py
     \ -- PLUGININFO
     \ -- data
       \ -- somefile.glade
       \ -- somefile.dat
     \ -- images
       \ -- somefile.png
```

The archive should be named with the extension *.exz*. The name of the plugin.exz file needs to match the name of the plugin directory.

So in the above example, you would need to call your plugin *root.exz* in order for it to be accepted by Exaile.

exz files can optionally be compressed, using either gzip or bzip2. the extension remains the same.

This is all you need to do to make a plugin archive.

2.5.13 Exaile API

Now you know the basics about programming plugins for Exaile, but there are many more useful classes you may need. You can get an overview about the classes and their use by going through the *Exaile API Docs*.

Building your own version of this documentation

You can use the Python package manager (pip) to install sphinx:

```
$ pip install sphinx

# or on windows
$ py -m pip install sphinx
```

Then you can run the following command in a terminal:

```
$ cd doc && make html
```

You'll find the documentation in `doc/_build/html`.

2.6 Windows Installer

Note: Windows installers are built by Appveyor automatically for every commit and pull request. Artifacts are currently retained for six months. You may find it easier to just download the installer from Appveyor instead of creating it yourself.

Go to https://ci.appveyor.com/project/ExaileDevelopmentTeam/exaile/history, click a commit, and select 'Artifacts'.

2.6.1 Install the SDK

You will need to have the SDK installed on your Windows machine. First clone the repo somewhere.

```
git clone https://github.com/exaile/python-gtk3-gst-sdk
```

Next install the SDK by running this from inside the tools/installer directory:

```
/path/to/python-gtk3-gst-sdk/win_installer/build_win32_sdk.sh
```

2.6.2 Build the installer

Build the installer by running this command from the tools/installer directory:

```
/path/to/python-gtk3-gst-sdk/win_installer/build_win32_installer.sh
```

2.7 Release process

This is an attempt to document what needs to be done in order to create a release for Exaile.

2.7.1 Step 0: Upgrading the Exaile SDK for Windows (if needed)

If you want to generate a new SDK, go to the exaile-sdk-win project on AppVeyor and click "New Build". Once the build is done, you can update the `sdk_ver` variable on `appveyor.yml` to the new SDK build number.

Note that new SDK versions can come with issues. It's better to do this step well in advance and test the result to make sure nothing breaks. In fact it's better to do this regularly, so that if something does break, we can revert to a not-too-old SDK version.

2.7.2 Step 1: Translations

Ensure that the translations from weblate are merged. Generally, this should happen automatically. It's probably easiest to check via the command line in your repo.

If you haven't already, add weblate to your git remotes:

```
$ git remote add weblate git://git.weblate.org/exaile.git
```

Check to see if the weblate repo has the same commits as the exaile repo (assuming that origin is pointing at the main exaile repo).

```
$ git fetch weblate
$ git fetch origin
$ git log -1 origin/master
$ git log -1 weblate/master
```

If they're equivalent, then we're all set. If not, then figure out what needs to be done to get them merged.

2.7.3 Step 2: Gather and update release notes

There's a lot of ways to go about this. I find that the easiest way to see what has changed is go to GitHub releases page, find the last release, and click on XXX commits since this release. Then you can browse the list of commits and pick out anything worth noting there.

If there is an actively-maintained changelog / release notes page, update it. This may include updating the release date, preferably in UTC.

2.7.4 Step 3: Tag the release locally

Make sure you have the correct thing checked out in your git tree, and then tag the release.

```
$ git tag -a RELEASE_VERSION
```

You can either add some release notes as the tag message or just write "Exaile RELEASE_VERSION".

2.7.5 Step 4: Update plugin versions (if needed)

If the PLUGININFO files still refer to the old version number, update them:

```
$ tools/plugin_tool.py fix
```

This currently must not be done from Windows because it will clobber the line separators.

Note that the new version number in the PLUGININFO files does not include any alpha/beta/rc label, so once you've done it for version a.b.c-alpha1 you don't need to do this step again for version a.b.c.

Commit the changes and re-tag the release:

```
$ git add plugins/*/PLUGININFO
$ git commit
$ git tag -d RELEASE_VERSION
$ git tag -a RELEASE_VERSION
```

2.7.6 Step 5: Push the tag

```
$ git push origin RELEASE_VERSION
```

Do not push to master before doing this; our auto-release setup only works when there is a new commit associated with a tag. If you've made this mistake, delete the tag and create an empty commit:

```
$ git tag -d RELEASE_VERSION
$ git push -d origin RELEASE_VERSION
$ git commit --allow-empty
```

then re-tag and re-push.

2.7.7 Step 6: Release the release

Once the tag is in the GitHub repository, GitHub Actions will build a source tarball and AppVeyor will build a Windows installer. They will create a draft release on GitHub containing those files. Edit the draft, paste in your release notes, then click 'Publish Release'.

Ideally, the release notes should include a checksum for each release artifact. This can be created (for the format we usually use) with

```
sha256sum --tag FILENAME
```

2.7.8 Final steps

Once the tag is built and released, you can push to the master branch.

Next, close out the milestone (if applicable) on GitHub.

2.7.9 Sending release notices

After a release, we should:

- Update website (hosted via GitHub Pages at https://github.com/exaile/exaile.github.io)
  - Update versions in `_config.yml`
  - Add a new post to `_posts`
- Send email to exaile-dev and exaile-users mailing lists with the release notes
- Update the channel topic on IRC (`/msg ChanServ topic #exaile ...`)

2.8 Exaile API Docs

Core:

2.8.1 Collection

Classes representing collections and libraries

A collection is a database of tracks. It is based on `TrackDB` but has the ability to be linked with libraries.

A library finds tracks in a specified directory and adds them to an associated collection.

Collections

`xl.collection.`**`get_collection_by_loc`**(*loc: str*) → Optional[xl.collection.Collection]
gets the collection by a location.

Parameters **`loc`** – Location of the collection

Returns collection at location or None

`class` `xl.collection.`**`Collection`**(*name*, *location=None*, *pickle_attrs=[]*)
Manages a persistent track database.

Simple usage:

```
>>> from xl.collection import *
>>> from xl.trax import search
>>> collection = Collection("Test Collection")
>>> collection.add_library(Library("./tests/data"))
>>> collection.rescan_libraries()
>>> tracks = [i.track for i in search.search_tracks_from_string(
...     collection, ('artist==TestArtist'))]
>>> print(len(tracks))
5
```

`add_library`(*library: xl.collection.Library*) → None
Add this library to the collection

Parameters **`library`** – the library to add

`close`()
close the collection. does any work like saving to disk, closing network connections, etc.

`freeze_libraries`() → None
Prevents "libraries_modified" events from being sent from individual add and remove library calls.

Call this before making bulk changes to the libraries. Call thaw_libraries when you are done; this sends a single event if the libraries were modified.

`get_libraries`() → List[xl.collection.Library]
Gets a list of all the Libraries associated with this Collection

`remove_library`(*library: xl.collection.Library*) → None
Remove a library from the collection

Parameters **`library`** – the library to remove

`rescan_libraries`(*startup_only=False*, *force_update=False*)
Rescans all libraries associated with this Collection

`serialize_libraries`()
Save information about libraries

Called whenever the library's settings are changed

stop_scan()
Stops the library scan

thaw_libraries() → None
Re-allow "libraries_modified" events from being sent from individual add and remove library calls. Also sends a "libraries_modified" event if the libraries have ben modified since the last call to freeze_libraries.

unserialize_libraries(*_serial_libraries*)
restores libraries from their serialized state.

Should only be called once, from the constructor.

class xl.collection.**CollectionScanThread**(*collection*, *startup_scan=False*, *force_update=False*)
Scans the collection

on_scan_progress_update(*type*, *collection*, *progress*)
Notifies about progress changes

run()
Runs the thread

stop()
Stops the thread

Libraries

class xl.collection.**Library**(*location: str*, *monitored: bool = False*, *scan_interval: int = 0*, *startup_scan: bool = False*)
Scans and watches a folder for tracks, and adds them to a Collection.

Simple usage:

```
>>> from xl.collection import *
>>> c = Collection("TestCollection")
>>> l = Library("./tests/data")
>>> c.add_library(l)
>>> l.rescan()
True
>>> print(c.get_libraries()[0].location)
./tests/data
>>> print(len(list(c.search('artist="TestArtist"'))))
5
>>>
```

add(*loc: str*, *move: bool = False*) → None
Copies (or moves) a file into the library and adds it to the collection

delete(*loc: str*) → None
Deletes a file from the disk

> **Warning:** This permanently deletes the file from the hard disk.

get_location() → str
Gets the current location associated with this Library

Returns the current location

get_monitored() → bool
Whether the library should be monitored for changes

get_rescan_interval() → int

Returns the scan interval in seconds

monitored
Whether the library should be monitored for changes

rescan(*notify_interval: Optional[int] = None*, *force_update: bool = False*)
Rescan the associated folder and add the contained files to the Collection

set_location(*location: str*) → None
Changes the location of this Library

Parameters **location** – the new location to use

set_monitored(*monitored: bool*) → None
Enables or disables monitoring of the library

Parameters **monitored** (*bool*) – Whether to monitor the library

set_rescan_interval(*interval: int*) → None
Sets the scan interval in seconds. If the interval is 0 seconds, the scan interval is stopped

Parameters **interval** – scan interval in seconds

update_track()
Rescan the track at a given location

Parameters

- **gloc** (Gio.File) – the location
- **force_update** – Force update of file (default only updates file when mtime has changed)

returns: the Track object, None if it could not be updated

class xl.collection.**LibraryMonitor**(*library*)
Monitors library locations for changes

2.8.2 Common utilities

General functions and classes shared in the codebase

General functions

xl.common.**order_poset**(*items*)

Parameters **items** (list of *PosetItem*) – poset to order

Filesystem

xl.common.**open_file**(*path*)
Opens a file or folder using the system configured program

xl.common.**open_file_directory**(*path_or_uri*)
Opens the parent directory of a file, selecting the file if possible.

xl.common.**walk**(*root*)
Walk through a Gio directory, yielding each file

Files are enumerated in the following order: first the directory, then the files in that directory. Once one directory's files have all been listed, it moves on to the next directory. Order of files within a directory and order of directory traversal is not specified.

Parameters **root** – a `Gio.File` representing the directory to walk through

Returns a generator object

Return type `Gio.File`

xl.common.**walk_directories**(*root*)
Walk through a Gio directory, yielding each subdirectory

Parameters **root** – a `Gio.File` representing the directory to walk through

Returns a generator object

Return type `Gio.File`

Decorators

xl.common.**threaded**(*func*)
A decorator that will make any function run in a new thread

Parameters **func** – the function to run threaded

xl.common.**synchronized**(*func*)
A decorator to make a function synchronized - which means only one thread is allowed to access it at a time.

This only works on class functions, and creates a variable in the instance called _sync_lock.

If this function is used on multiple functions in an object, they will be locked with respect to each other. The lock is re-entrant.

xl.common.**profileit**(*func*)
Decorator to profile a function

xl.common.**classproperty**(*function*)
Decorator allowing for class property access

xl.common.**cached**(*limit*)
Decorator to make a function's results cached does not cache if there is an exception.

Note: This probably breaks on functions that modify their arguments

Classes

exception xl.common.**VersionError**(*message*)

class xl.common.**LimitedCache**(*limit*)
Bases: `collections.abc.MutableMapping`

Simple cache that acts much like a dict, but has a maximum # of items

class xl.common.**TimeSpan**(*span*)
Calculates the number of days, hours, minutes, and seconds in a time span

days = 0
number of days

hours = 0
number of hours

minutes = 0
number of minutes

seconds = 0
number of seconds

class xl.common.**MetadataList**(*iterable=[]*, *metadata=[]*)
Like a list, but also associates an object of metadata with each entry.

(get|set|del)_meta_key are the metadata interface - they allow the metadata to act much like a dictionary, with a few optimizations.

List aspects that are not supported:

- sort
- comparisons other than equality
- multiply

class xl.common.**ProgressThread**
A basic thread with progress updates. The thread should emit the progress-update signal periodically. The contents must be number between 0 and 100, or a tuple of (n, total) where n is the current step.

run()
Override and make sure that the 'progress-update' signal is emitted regularly with the progress

stop()
Stops the thread

class xl.common.**PosetItem**(*name*, *after*, *priority*, *value=None*)

2.8.3 Cover

Provides the base for obtaining and storing covers, also known as album art.

Cover Manager

xl.covers.**MANAGER = <xl.covers.CoverManager object>**
The singleton *CoverManager* instance

class xl.covers.**CoverManager**(*location*)
Handles finding covers from various sources.

find_covers(*track*, *limit=-1*, *local_only=False*)
Find all covers for a track

Parameters

- **track** – The track to find covers for
- **limit** – maximum number of covers to return. -1=unlimited.
- **local_only** – If True, will only return results from local sources.

get_cover(*track*, *save_cover=True*, *set_only=False*, *use_default=False*)
get the cover for a given track. if the track has no set cover, backends are searched until a cover is found or we run out of backends.

Parameters

- **track** – the Track to get the cover for.
- **save_cover** – if True, a set_cover call will be made to store the cover for later use.
- **set_only** – Only retrieve covers that have been set in the db.
- **use_default** – If True, returns the default cover instead of None when no covers are found.

get_cover_data(*db_string*, *use_default=False*)
Get the raw image data for a cover.

Parameters

- **db_string** – The db_string identifying the cover to get.
- **use_default** – If True, returns the default cover instead of None when no covers are found.

get_cover_for_tracks(*tracks*, *db_strings_to_ignore*)
For tracks, try to find a cover Basically returns the first cover found :param tracks: list of tracks [xl.trax.Track] :param db_strings_to_ignore: list [str] :return: GdkPixbuf.Pixbuf or None if no cover found

get_db_string(*track: xl.trax.track.Track*) → Optional[str]
Returns the internal string used to map the cover to a track

Parameters **track** (*xl.trax.Track*) – the track to retrieve the string for

Returns the internal identifier string

get_default_cover()
Get the raw image data for the cover to show if there is no cover to display.

load()
Load the saved db

on_provider_added(*provider*)
Called when a new provider is added

Parameters **provider** (*object*) – the new provider

on_provider_removed(*provider*)
Called when a provider is removed

Parameters **provider** (*object*) – the removed provider

remove_cover(*track*)
Remove the saved cover entry for a track, if it exists.

save()
Save the db

set_cover(*track*, *db_string*, *data=None*)
Sets the cover for a track. This will overwrite any existing entry.

Parameters

- **track** – The track to set the cover for
- **db_string** – the string identifying the source of the cover, in "method:key" format.

- **data** – The raw cover data to store for the track. Will only be stored if the method has use_cache=True

set_preferred_order(*order*)
Sets the preferred search order

Parameters **order** – a list containing the order you'd like to search first

Cover Search Methods

class xl.covers.**CoverSearchMethod**
Base class for creating cover search methods.

Search methods do not have to inherit from this class, it's intended more as a template to demonstrate the needed interface.

find_covers(*track*, *limit=-1*)
Find the covers for a given track.

Parameters

- **track** – The track to find covers for.
- **limit** – Maximal number of covers to return.

Returns A list of strings that can be passed to get_cover_data.

get_cover_data(*db_string*)
Get the image data for a cover

Parameters **db_string** – A method-dependent string that identifies the cover to get.

name = 'base'
A name uniquely identifing the search method.

use_cache = True
If true, cover results will be cached for faster lookup

class xl.covers.**TagCoverFetcher**
Cover source that looks for images embedded in tags.

class xl.covers.**LocalFileCoverFetcher**
Cover source that looks for images in the same directory as the Track.

2.8.4 Events

Provides a signals-like system for sending and listening for 'events'

Events are kind of like signals, except they may be listened for on a global scale, rather than connected on a per-object basis like signals are. This means that ANY object can emit ANY event, and these events may be listened for by ANY object.

Events should be emitted AFTER the given event has taken place. Often the most appropriate spot is immediately before a return statement.

xl.event.**log_event**(*evty*, *obj*, *data*)
Sends an event.

Parameters

- **evty** (*string*) – the *type* or *name* of the event.

- **obj** (*object*) – the object sending the event.
- **data** (*object*) – some data about the event, None if not required

xl.event.**add_callback**(*function*, *evty=None*, *obj=None*, **args*, ***kwargs*)
Adds a callback to an event

You should ALWAYS specify one of the two options on what to listen for. While not forbidden to listen to all events, doing so will cause your callback to be called very frequently, and possibly may cause slowness within the player itself.

Parameters

- **function** (*callable*) – the function to call when the event happens
- **evty** (*string*) – the *type* or *name* of the event to listen for, eg *tracks_added*, *cover_changed*. Defaults to any event if not specified.
- **obj** (*object*) – the object to listen to events from, e.g. *exaile.collection* or *xl.covers.MANAGER*. Defaults to any object if not specified.
- **destroy_with** – (keyword arg only) If specified, this event will be detached when the specified Gtk widget is destroyed

Any additional parameters will be passed to the callback.

Returns a convenience function that you can call to remove the callback.

xl.event.**remove_callback**(*function*, *evty=None*, *obj=None*)
Removes a callback. Can remove both ui and non-ui callbacks.

The parameters passed should match those that were passed when adding the callback

2.8.5 Formatter

Provides an extensible framework for processing and preparation of data for display in various contexts.

Formatters

class xl.formatter.**Formatter**(*format*)
A generic text formatter based on a format string

By default the following parameters are provided to each identifier:

- **prefix, suffix: a string to put before or after the formatted string if that string is not empty**
  - Whitespace will be not be touched and transferred as is
  - The characters `,`, `}` and = need to be escaped like `\,`, `\}` and `\=` respectively
- `pad`: desired length the formatted string should have, will be achieved using the `padstring`
- **padstring: a string to use for padding, will be repeated as often as possible to achieve the desired length specified**
  - Example: `${identifier:pad=4, padstring=XY}` for *identifier* having the value *a* will become *XYXa*

extract()
Retrieves the identifiers and their optional parameters

Example of the returned dictionary:

```
extractions = {
    'identifier1': (
        'identifier1', {}),
    'identifier2:parameter': (
        'identifier2', {'parameter': True}),
    'identifier3:parameter=argument': (
        'identifier3', {'parameter': 'argument'})
}
```

Returns the extractions

Return type dict

format(**args*)

Returns a string by formatting the passed data

Parameters **args** – data to base the formatting on

Returns the formatted text

Return type string

class xl.formatter.**ProgressTextFormatter**(*format*, *player*)

A text formatter for progress indicators

format(*current_time=None*, *total_time=None*)

Returns a string suitable for progress indicators

Parameters

- **current_time** (*float*) – the current progress, taken from the current playback if not set
- **total_time** (*float*) – the total length of a track, taken from the current playback if not set

Returns The formatted text

Return type string

class xl.formatter.**TrackFormatter**(*format*)

A formatter for track data

format(*track*, *markup_escape=False*)

Returns a string for places where track data is presented to the user

Parameters

- **track** (*xl.trax.Track*) – a single track to take data from
- **markup_escape** (*bool*) – whether to escape markup-like characters in tag values

Returns the formatted text

Return type string

class xl.formatter.**TagFormatter**(*name*)

A formatter provider for a tag of a track

format(*track*, *parameters*)

Formats a raw tag value. Accepts optional parameters to manipulate the formatting process.

Parameters

- **track** (*xl.trax.Track*) – the track to get the tag from
- **parameters** (*dictionary*) – optionally passed parameters

Returns the formatted value

Return type string

class xl.formatter.**TrackNumberTagFormatter**
Bases: xl.formatter.NumberTagFormatter

A formatter for the tracknumber of a track

class xl.formatter.**DiscNumberTagFormatter**
Bases: xl.formatter.NumberTagFormatter

A formatter for the discnumber of a track

class xl.formatter.**ArtistTagFormatter**
Bases: *xl.formatter.TagFormatter*

A formatter for the artist of a track

format(*track*, *parameters*)
Formats a raw tag value

Parameters

- **track** (*xl.trax.Track*) – the track to get the tag from
- **parameters** – optionally passed parameters Possible values are:
  - compilate: Allows for proper handling of compilations, either via albumartist tag, a fallback value, or simply all artists

Returns the formatted value

Return type string

class xl.formatter.**LengthTagFormatter**
Bases: xl.formatter.TimeTagFormatter

A formatter for the length of a track

format(*track*, *parameters*)
Formats a raw tag value

Parameters

- **track** (*xl.trax.Track*) – the track to get the tag from
- **parameters** (*dictionary*) – Verbosity of the output, possible values for "format" are:
  - short: "1:02:42"
  - long: "1h, 2m, 42s"
  - verbose: "1 hour, 2 minutes, 42 seconds"

Returns the formatted value

Return type string

static format_value(*value*, *format='short'*)
Formats a length value

Parameters

- **value** (*float*) – the length in seconds
- **format** (*string*) – verbosity of the output, possible values are:
  - short: "1:02:42"
  - long: "1h, 2m, 42s"
  - verbose: "1 hour, 2 minutes, 42 seconds"

Returns the formatted value

Return type string

class xl.formatter.**RatingTagFormatter**
Bases: *xl.formatter.TagFormatter*

A formatter for the rating of a track

Will return glyphs representing the rating like

class xl.formatter.**LastPlayedTagFormatter**
Bases: xl.formatter.DateTagFormatter

A formatter for the last time a track was played

Templates

class xl.formatter.**ParameterTemplate**(*template*)
An extended template class which additionally accepts parameters assigned to identifiers.

This introduces another pattern group named "parameters" in addition to the groups created by string.Template

Examples:

- `${foo:parameter1}`
- `${bar:parameter1, parameter2}`
- `${qux:parameter1=argument1, parameter2}`

2.8.6 Metadata

xl.metadata.**formats = {'669': <class 'xl.metadata.mod.ModFormat'>, 'aac': <class 'xl.meta**
dictionary mapping extensions to Format classes.

xl.metadata.**get_format**(*loc: str*) → Optional[xl.metadata._base.BaseFormat]
get a Format object appropriate for the file at loc. if no suitable object can be found, None is returned.

Parameters **loc** – The location to read from as a Gio URI (from Track.get_loc_for_io())

Format Objects

exception xl.metadata.**NotWritable**

exception xl.metadata.**NotReadable**

class xl.metadata.**BaseFormat**(*loc*)
Base class for handling loading of metadata from files.

subclasses using mutagen should set MutagenType and overload the _get_tag, _set_tag, and _del_tag methods as needed.

subclasses not using mutagen should leave MutagenType as None

read_all()
Reads all non-blacklisted tags from the file.

Blacklisted tags include lyrics, covers, and any field starting with __. If you need to read these, call read_tags directly.

read_tags(*tags*)
get the values for the specified tags.

returns a dict of the found values. if no value was found for a requested tag it will not exist in the returned dict.

Parameters **tags** – a list of exaile tag names to read

Returns a dictionary of tag/value pairs.

write_tags(*tagdict*)
Write a set of tags to the file. Raises a NotWritable exception if the format does not support writing tags.

When calling this function, we assume the following:

- tagdict has all keys that you wish to write, keys are exaile tag names or custom tag names and values are the tags to write (lists of unicode strings)
- if a value is None, then that tag will be deleted from the file
- Will not modify/delete tags that are NOT in tagdict
- Will not write tags that start with '__'

Parameters **tagdict** – A dictionary of tag/value pairs to write.

2.8.7 Player

Allows for playback and queue control

xl.player.**PLAYER = <xl.player.player.ExailePlayer object>**
This is the player object that everything in Exaile interacts with to control audio playback. The player object controls a playback engine, which actually controls audio playback. Nothing in this object should be specific to a particular engine. Examples of engines could be GStreamer, Xine, etc. Currently only the GStreamer engine is actually implemented.

All public functions are assumed to be called from the Glib main thread, or bad things will happen. This includes most engine functions, with one or two noted exceptions.

The player singleton of ExailePlayer for playback control

ExailePlayer.**play**(*track*, *start_at=None*, *paused=False*)
Starts the playback with the provided track or stops the playback it immediately if none

Parameters

- **track** (*xl.trax.Track*) – the track to play or None
- **start_at** – The offset to start playback at, in seconds

- **paused** – If True, start the track in 'paused' mode

Note: The following *events* will be emitted by this method:

- *playback_player_start*: indicates the start of playback overall
- *playback_track_start*: indicates playback start of a track

---

ExailePlayer.**stop**()
Stops the playback

Note: The following *events* will be emitted by this method:

- *playback_player_end*: indicates the end of playback overall
- *playback_track_end*: indicates playback end of a track

ExailePlayer.**pause**()
Pauses the playback if playing, does not toggle it

Returns True if paused, False otherwise

Note: The following *events* will be emitted by this method:

- *playback_player_pause*: indicates that the playback has been paused
- *playback_toggle_pause*: indicates that the playback has been paused or resumed

---

ExailePlayer.**unpause**()
Resumes the playback if it is paused, does not toggle it

Returns True if paused, False otherwise

Note: The following *events* will be emitted by this method:

- *playback_player_resume*: indicates that the playback has been resumed
- *playback_toggle_pause*: indicates that the playback has been paused or resumed

---

ExailePlayer.**toggle_pause**()
Toggles between playing and paused state. Only valid when playback is not stopped.

Returns True if toggled, false otherwise

Note: The following *events* will be emitted by this method:

- *playback_toggle_pause*: indicates that the playback has been paused or resumed

---

ExailePlayer.**seek**(*value*)
Seek to a position in the currently playing stream

Parameters **value** (*int*) – the position in seconds

ExailePlayer.**get_position**()
Gets the current playback position of the playing track

Returns the playback position in nanoseconds

Return type int

ExailePlayer.**get_time**()
Gets the current playback time

Returns the playback time in seconds

Return type float

ExailePlayer.**get_progress**() → float
Gets the current playback progress

Returns the playback progress as [0..1]

ExailePlayer.**set_progress**(*progress*)
Seeks to the progress position

Parameters **progress** (*float*) – value ranged at [0..1]

ExailePlayer.**get_volume**()
Gets the current user volume

Returns the volume percentage

Type int

ExailePlayer.**set_volume**(*volume*)
Sets the current user volume

Parameters **volume** (*int*) – the volume percentage

ExailePlayer.**modify_volume**(*diff*)
Changes the current user volume

Parameters **diff** – the volume difference (pos or neg) percentage units

ExailePlayer.**get_state**()
Gets the player state

Returns one of *playing*, *paused* or *stopped*

Return type string

ExailePlayer.**is_playing**()
Convenience method to find out if the player is currently playing

Returns whether the player is currently playing

Return type bool

ExailePlayer.**is_paused**()
Convenience method to find out if the player is currently paused

Returns whether the player is currently paused

Return type bool

ExailePlayer.**is_stopped**()
Convenience method to find out if the player is currently stopped

Returns whether the player is currently stopped

Return type bool

xl.player.**QUEUE = <xl.player.queue.PlayQueue object>**
Manages the queue of songs to be played

The content of the queue are processed before processing the content of the assigned playlist.

When the remove_item_when_played option is enabled, the queue removes items from itself as they are played.

When not enabled, the queue acts like a regular playlist, and moves the position as tracks are played.

In this mode, when a new track is queued, the position is set to play that track, and play will continue with that track until the queue is exhausted, and then the assigned playlist will be continued.

TODO: Queue needs to be threadsafe!

The queue singleton of `PlayQueue`

class xl.player.queue.**PlayQueue**(*player*, *name*, *location=None*)
Bases: *xl.playlist.Playlist*

Manages the queue of songs to be played

The content of the queue are processed before processing the content of the assigned playlist.

When the remove_item_when_played option is enabled, the queue removes items from itself as they are played.

When not enabled, the queue acts like a regular playlist, and moves the position as tracks are played.

In this mode, when a new track is queued, the position is set to play that track, and play will continue with that track until the queue is exhausted, and then the assigned playlist will be continued.

TODO: Queue needs to be threadsafe!

current_playlist
The playlist currently processed in the queue

get_current()
Gets the current track
Returns the current track
Type *xl.trax.Track*

get_current_position()
Retrieves the current position within the playlist
Returns the position
Return type int

get_next()
Retrieves the next track that will be played. Does not actually set the position. When you call next(), it should return the same track.

This exists to support retrieving a track before it actually needs to be played, such as for pre-buffering.
Returns the next track to be played
Return type *xl.trax.Track* or None

is_play_enabled()
Returns True when calling play() will have no effect

next(*autoplay=True*, *track=None*)
Goes to the next track, either in the queue, or in the current playlist. If a track is passed in, that track is played
Parameters
- **autoplay** (*bool*) – play the track in addition to returning it
- **track** (*xl.trax.Track*) – if passed, play this track

Note: The following *events* will be emitted by this method:
- *playback_playlist_end*: indicates that the end of the queue has been reached

---

play(*track=None*)
Starts queue processing with the given track preceding the queue content
Parameters **track** (*xl.trax.Track*) – the track to play

prev()
Goes to the previous track

queue_length()
Returns the number of tracks left to play in the queue's internal playlist.

set_current_playlist(*playlist*)
Sets the playlist to be processed in the queue
Parameters **playlist** (*xl.playlist.Playlist*) – the playlist to process

---

Note: The following *events* will be emitted by this method:
- *queue_current_playlist_changed*: indicates that the queue playlist has been changed

---

set_current_position(*position*)
Sets the current position within the playlist
Parameters **position** (*int*) – the new position

2.8.8 Playlist

Provides the fundamental objects for handling a list of tracks contained in playlists as well as methods to import and export from various file formats.

Playlists

class xl.playlist.**Playlist**(*name*, *initial_tracks=[]*)
Basic class for handling a list of tracks

EVENTS: (all events are synchronous)

- **playlist_tracks_added**
  - fired: after tracks are added
  - data: list of tuples of (index, track)
- **playlist_tracks_removed**
  - fired: after tracks are removed
  - data: list of tuples of (index, track)
- playlist_current_position_changed
- playlist_shuffle_mode_changed
- playlist_random_mode_changed
- playlist_dynamic_mode_changed

append(*other*)
Appends a single track to the playlist

Prefer extend() for batch updates, so that playlist_tracks_added is not emitted excessively.

Parameters **other** – a *xl.trax.Track*

clear()
Removes all contained tracks

clear_shuffle_history()
Clear the history of played tracks from a shuffle run

count(*other*)
Returns the count of contained tracks

Returns the count

Return type int

current_position
The position within the playlist (int)

dirty
Whether the playlist was changed or not (boolean)

dynamic_mode
The current dynamic mode (string)

extend(*other*)
Extends the playlist by another playlist

Parameters **other** – list of *xl.trax.Track*

get_current()
Retrieves the track at the current position

Returns the track

Return type *xl.trax.Track* or None

get_current_position()
Retrieves the current position within the playlist

Returns the position

Return type int

get_dynamic_mode()
Retrieves the current dynamic mode

Returns the dynamic mode

Return type string

get_repeat_mode()
Retrieves the current repeat mode

Returns the repeat mode

Return type string

get_shuffle_history()
Retrieves the history of played tracks from a shuffle run

Returns the tracks

Return type list

get_shuffle_mode()
Retrieves the current shuffle mode

Returns the shuffle mode

Return type string

get_spat_position()
Retrieves the current position within the playlist after which progressing shall be stopped

Returns the position

Return type int

index(*item*, *start=0*, *end=None*)
Retrieves the index of a track within the playlist

Returns the index

Return type int

load_from_location(*location*)
Loads the content of the playlist from a given location

Parameters **location** (*string*) – the location to load from

name
The playlist name (string)

next()
Progresses to the next track within the playlist and takes shuffle and repeat modes into account

Returns the new current track

Return type *xl.trax.Track* or None

pop(*i=-1*)
Pops a track from the playlist

Parameters **i** (*int*) – the index

Returns the track

Return type *xl.trax.Track*

prev()
Progresses to the previous track within the playlist and takes shuffle and repeat modes into account

Returns the new current track

Return type *xl.trax.Track* or None

randomize(*positions=None*)
Randomizes the content of the playlist contrary to shuffle which affects only the progressing order

By default all tracks in the playlist are randomized, but a list of positions can be passed. The tracks on these positions will be randomized, all other tracks will keep their positions.

Parameters **positions** (*iterable*) – list of track positions to randomize

repeat_mode
The current repeat mode (string)

save_to_location(*location*)
Writes the content of the playlist to a given location

Parameters **location** (*string*) – the location to save to

set_current_position (*position*)
Sets the current position within the playlist

Parameters **position** (*int*) – the new position

set_dynamic_mode (*mode*)
Sets the current dynamic mode

Parameters **mode** (*string*) – the new dynamic mode

set_repeat_mode (*mode*)
Sets the current repeat mode

Parameters **mode** (*string*) – the new repeat mode

set_shuffle_mode (*mode*)
Sets the current shuffle mode

Parameters **mode** (*string*) – the new shuffle mode

set_spat_position (*position*)
Sets the current position within the playlist after which progressing shall be stopped

Parameters **position** (*int*) – the new position

shuffle_mode
The current shuffle mode (string)

shuffle_mode_names = ['Shuffle _Off', 'Shuffle _Tracks', 'Shuffle _Albums', '_Random']
Titles of the valid shuffle modes (list of string)

shuffle_modes = ['disabled', 'track', 'album', 'random']
Valid shuffle modes (list of string)

sort (*tags*, *reverse=False*)
Sorts the content of the playlist

Parameters

- **tags** (*list of strings*) – tags to sort by
- **reverse** (*boolean*) – whether the sorting shall be reversed

spat_position
The position within the playlist after which to stop progressing (int)

Playlist Converters

class xl.playlist.**FormatConverter** (*name*)
Base class for all converters allowing to import from and export to a specific format

export_to_file (*playlist*, *path*, *options=None*)
Export a playlist to a given path

Parameters

- **playlist** (*Playlist*) – the playlist
- **path** (*string*) – the target path
- **options** (PlaylistExportOptions) – exporting options

import_from_file(*path*)
Import a playlist from a given path

Parameters **path** (*string*) – the source path

Returns the playlist

Return type *Playlist*

name_from_path(*path*)
Convenience method to retrieve a sane name from a path

Parameters **path** (*string*) – the source path

Returns a name

Return type string

class xl.playlist.**M3UConverter**
Bases: *xl.playlist.FormatConverter*

Import from and export to M3U format

class xl.playlist.**PLSConverter**
Bases: *xl.playlist.FormatConverter*

Import from and export to PLS format

class xl.playlist.**ASXConverter**
Bases: *xl.playlist.FormatConverter*

Import from and export to ASX format

class xl.playlist.**XSPFConverter**
Bases: *xl.playlist.FormatConverter*

Import from and export to XSPF format

2.8.9 Providers & Services

A generic framework for service providers, recommended to be used whenever there are multiple ways of accomplishing a task or multiple sources can offer the required data.

xl.providers.**MANAGER**
Singleton instance of the *ProviderManager*

xl.providers.**register**(*servicename*, *provider*, *target=None*)
Registers a provider for a service. The provider object is used by consumers of the service.

Services can be targeted for a specific use. For example, if you have a widget that uses a service 'foo', if your object can perform a service only for a specific type of widget, then target would be set to the widget type.

If you had a service that could perform 'foo' for all widgets, then target would be set to None, and all widgets could use your service.

It is intended that most services should set target to None, with some narrow exceptions.

Parameters

- **servicename** (*string*) – the name of the service [string]
- **provider** (*object*) – the object that is the provider [object]
- **target** (*object*) – a specific target for the service [object]

xl.providers.**unregister**(*servicename*, *provider*, *target=None*)
Unregisters a provider.

Parameters

- **servicename** (*string*) – the name of the service
- **provider** (*object*) – the provider to be removed
- **target** (*object*) – a specific target for the service [object]

xl.providers.**get**(*servicename*, *target=None*)
Returns a list of providers for the specified servicename.

This will return providers targeted for a specific target AND providers not targeted towards any particular target.

Parameters

- **servicename** (*string*) – the service name to get providers for
- **target** (*object*) – the target of the service

Returns list of providers

Return type list of objects

xl.providers.**get_provider**(*servicename*, *providername*, *target=None*)
Returns a single identified provider

This will return a provider either targeted for the specific target or a provider not targeted towards any particular target.

Parameters

- **servicename** (*string*) – The service name to get the provider for
- **providername** (*string*) – The provider name to identify the provider
- **target** (*object*) – the target of the service

Returns a provider or None

Return type object

class xl.providers.**ProviderManager**
The overall manager for services and providers for them

class xl.providers.**ProviderHandler**(*servicename*, *target=None*, *simple_init=False*)
Base class to handle providers for one specific service including notification about (un)registration

2.8.10 Settings

Central storage of application and user settings

xl.settings.**MANAGER**
Singleton instance of the *SettingsManager*

xl.settings.**get_option**(*name*, *default*)

xl.settings.**set_option**(*option*, *value*, *save=True*)
Set an option (in `section/key` syntax) to the specified value

Parameters

- **option** (*string*) – the full path to an option
- **value** (*any*) – the value the option should be assigned

- **save** – If True, cause the settings to be written to file

class xl.settings.**SettingsManager**(*location=None*, *default_location=None*)
Bases: configparser.RawConfigParser

Manages Exaile's settings

get_option(*option: str*, *default: Any = None*) → Any
Get the value of an option (in section/key syntax), returning *default* if the key does not exist yet

Parameters

- **option** – the full path to an option
- **default** – a default value to use as fallback

Returns the option value or *default*

has_option(*option*)
Returns information about the existence of a particular option

Parameters **option** (*string*) – the option path

Returns whether the option exists or not

Return type bool

remove_option(*option*)
Removes an option (in section/key syntax), thus will not be saved anymore

Parameters **option** (*string*) – the option path

save()
Save the settings to disk

set_option(*option*, *value*, *save=True*)
Set an option (in section/key syntax) to the specified value

Parameters

- **option** (*string*) – the full path to an option
- **value** (*any*) – the value the option should be assigned
- **save** – If True, cause the settings to be written to file

2.8.11 Trax

Provides the base for creating and managing Track objects.

Tracks

class xl.trax.**Track**(*uri=None*, *scan=True*, *_unpickles=None*)
Represents a single track.

exists()
Returns whether the file exists This can be very slow, use with caution!

get_loc_for_io()
Gets the location as a full uri.

Safe for IO operations via gio, not suitable for display to users as it may be in non-utf-8 encodings.

get_rating()
Returns the current track rating as an integer, as determined by the `rating/maximum` setting.

Return type int

get_tag_display(*tag*, *join=True*, *artist_compilations=False*, *extend_title=True*) → Union[str, List[str]]
Get a tag value in a form suitable for display.

Parameters

- **tag** – The name of the tag to get
- **join** – If True, joins lists of values into a single value.
- **artist_compilations** – If True, automatically handle albumartist and other compilations detections when tag==”albumartist”.
- **extend_title** – If the title tag is unknown, try to add some identifying information to it.

get_tag_raw(*tag*, *join=False*)
Get the raw value of a tag. For non-internal tags, the result will always be a list of unicode strings.

Parameters

- **tag** – The name of the tag to get
- **join** – If True, joins lists of values into a single value.

Returns None if the tag is not present

get_tag_sort(*tag*, *join=True*, *artist_compilations=False*, *extend_title=True*)
Get a tag value in a form suitable for sorting.

Parameters

- **tag** – The name of the tag to get
- **join** – If True, joins lists of values into a single value.
- **artist_compilations** – If True, automatically handle albumartist and other compilations detections when tag==”albumartist”.
- **extend_title** – If the title tag is unknown, try to add some identifying information to it.

get_type()
Get the URI schema the file uses, e.g. file, http, smb.

list_tags()
Returns a list of the names of all tags present in this Track.

read_tags(*force=True*, *notify_changed=True*)
Reads tags from the file for this Track.

Parameters **force** – If not True, then only read the tags if the file has be modified.

Returns False if unsuccessful, and a Format object from *xl.metadata* otherwise.

set_loc(*loc*, *notify_changed=True*)
Sets the location.

Parameters **loc** – the location, as either a uri or a file path.

set_rating(*rating*)
Sets the current track rating from an integer, on the scale determined by the `rating/maximum` setting.

Returns the scaled rating

set_tag_raw(*tag*, *values*, *notify_changed=True*)
Set the raw value of a tag.

Parameters

- **tag** – The name of the tag to set.
- **values** – The value or values to set the tag to.
- **notify_changed** – whether to send a signal to let other parts of Exaile know there has been an update. Only set this to False if you know that no other parts of Exaile need to be updated.

Note: When setting more than one tag, prefer set_tags instead

Warning: Covers and lyrics tags must be set via set_tag_disk

Returns True if changed, False otherwise

set_tags(*notify_changed=True*, ***kwargs*)
Set multiple tags on a track.

Parameters **notify_changed** – whether to send a signal to let other parts of Exaile know there has been an update. Only set this to False if you know that no other parts of Exaile need to be updated.

Prefer this method over calling set_tag_raw multiple times, as this method will be more efficient.

Warning: Covers and lyrics tags must be set via set_tag_disk

Returns Set of tags that have changed

write_tags()
Writes tags to the file for this Track.

Returns False if unsuccessful, and a Format object from *xl.metadata* otherwise.

xl.trax.**is_valid_track**(*location*)
Returns whether the file at the given location is a valid track

Parameters **location** (*string*) – the location to check

Returns whether the file is a valid track

Return type boolean

xl.trax.**get_uris_from_tracks**(*tracks*)
Returns all URIs for tracks

Parameters **tracks** (list of *xl.trax.Track*) – the tracks to retrieve the URIs from

Returns the uris

Return type list of string

xl.trax.**get_tracks_from_uri**(*uri*)
Returns all valid tracks located at uri

Parameters **uri** (*string*) – the uri to retrieve the tracks from

Returns the retrieved tracks

Return type list of *xl.trax.Track*

xl.trax.**sort_tracks**(*fields: Iterable[str], items: Iterable[T], trackfunc: Optional[Callable[[T], xl.trax.track.Track]] = None, reverse: bool = False, artist_compilations: bool = False*) → List[T]
Sorts tracks.

Parameters

- **fields** – tag names to sort by
- **items** – the tracks to sort, alternatively use *trackfunc*
- **trackfunc** – function to get a *Track* from an item in the *items* iterable
- **reverse** – whether to sort in reversed order

xl.trax.**sort_result_tracks**(*fields*, *trackiter*, *reverse=False*, *artist_compilations=False*)
Sorts SearchResultTracks, ie. the output from a search.

Same params as sort_tracks.

xl.trax.**get_rating_from_tracks**(*tracks*)
Returns the common rating for all tracks or simply 0 if not all tracks have the same rating. Same goes if the amount of tracks is 0 or more than the internal limit.

Parameters **tracks** (*iterable*) – the tracks to retrieve the rating from

Track Database

Track databases are a simple persistence layer to hold collections of Track objects.

class xl.trax.**TrackDB**(*name: str = ''*, *location: str = ''*, *pickle_attrs: List[str] = []*, *loadfirst: bool = False*)
Manages a track database.

Allows you to add, remove, retrieve, search, save and load Track objects.

Parameters

- **name** – The name of this *TrackDB*.
- **location** – Path to a file where this *TrackDB* should be stored.
- **pickle_attrs** – A list of attributes to store in the pickled representation of this object. All attributes listed must be built-in types, with one exception: If the object contains the phrase 'tracks' in its name it may be a list or dict of *Track* objects.
- **load_first** – Set to True if this collection should be loaded before any tracks are created.

add(*track: xl.trax.track.Track*) → None
Adds a track to the database of tracks

Parameters **track** – The *xl.trax.Track* to add

add_tracks(*tracks: Iterable[xl.trax.track.Track]*) → None
Like add(), but takes a list of *xl.trax.Track*

load_from_location(*location=None*)
Restores *TrackDB* state from the pickled representation stored at the specified location.

Parameters **location** (*string*) – the location to load the data from

remove(*track: xl.trax.track.Track*) → None
Removes a track from the database

Parameters **track** – the *xl.trax.Track* to remove

remove_tracks(*tracks: Iterable[xl.trax.track.Track]*) → None
Like remove(), but takes a list of *xl.trax.Track*

save_to_location(*location=None*)
Saves a pickled representation of this *TrackDB* to the specified location.

Parameters **location** (*string*) – the location to save the data to

Searching

class xl.trax.**TracksMatcher**(*search_string*, *case_sensitive=True*, *keyword_tags=None*)
Holds criteria and determines whether a given track matches those criteria.

xl.trax.**search_tracks**(*trackiter, trackmatchers: Collection[xl.trax.search.TracksMatcher]*)
Search a set of tracks for those that match specified conditions.

Parameters

- **trackiter** – An iterable object returning Track objects
- **trackmatchers** – A list of TrackMatcher objects

xl.trax.**search_tracks_from_string**(*trackiter*, *search_string*, *case_sensitive=True*, *keyword_tags=None*)
Convenience wrapper around search_tracks that builds matchers automatically from the search string.

Arguments have the same meaning as the corresponding arguments on on *search_tracks* and *TracksMatcher*.

2.8.12 D-Bus

D-Bus interface for playback control, data query and others

Access through the `/org/exaile/Exaile` object which implements the `org.exaile.Exaile` interface

org.exaile.Exaile Interface

xl.xldbus.**DbusManager**
alias of `mocks.`

GUI:

2.8.13 Icons & Images

Provides methods for convenient icons and image handling

Icon management

xlgui.icons.**MANAGER**
Singleton instance of the *IconManager*

class xlgui.icons.**IconManager**
Provides convenience functions for managing icons and images in general

add_icon_name_from_directory(*icon_name*, *directory*)
Registers an icon name from files found in a directory

Parameters

- **icon_name** (*string*) – the name for the icon
- **directory** (*string*) – the location to search for icons

Returns filesystem location of the highest-quality icon of this name, or None if not found

Return type Optional[str]

add_icon_name_from_file(*icon_name*, *filename*, *size=None*)
Registers an icon name from a filename

Parameters

- **icon_name** (*string*) – the name for the icon
- **filename** (*string*) – the filename of an image
- **size** (*int*) – the size the icon shall be registered for

add_icon_name_from_pixbuf(*icon_name*, *pixbuf*, *size=None*)
Registers an icon name from a pixbuf

Parameters

- **icon_name** (*string*) – the name for the icon
- **pixbuf** (GdkPixbuf.Pixbuf) – the pixbuf of an image
- **size** (*int*) – the size the icon shall be registered for

pixbuf_from_icon_name()
Generates a pixbuf from an icon name

Parameters

- **icon_name** – an icon name
- **size** – the size of the icon, will be tried to converted to a GTK icon size

Returns the generated pixbuf

pixbuf_from_rating(*rating*, *size_ratio=1*)
Returns a pixbuf representing a rating

Parameters **rating** (*int*) – the rating

Returns the rating pixbuf

Return type GdkPixbuf.Pixbuf

Utilities

class xlgui.icons.**ExtendedPixbuf**(*pixbuf*)

A `GdkPixbuf.Pixbuf` wrapper class allowing for interaction using standard operators

Thus you can do the following:

- `pixbuf1 + pixbuf2` (horizontally appends `pixbuf2` to `pixbuf1`)
- `pixbuf * 5` (multiplies the content of `pixbuf`)
- `pixbuf1 & pixbuf2` (simple composition of `pixbuf2` on `pixbuf1`, the desired alpha value has to be included in the pixbufs themselves)
- `pixbuf1 < pixbuf2`, `pixbuf1 > pixbuf2` (compares the pixbuf dimensions)
- `pixbuf1 == pixbuf2` (compares the pixel data, use the *is* operator to check for identity)

Even more is possible with the provided verbose methods

add_horizontal(*other*, *spacing=0*)

Horizontally appends a pixbuf to the current

Parameters

- **other** (`GdkPixbuf.Pixbuf`) – the pixbuf to append
- **spacing** (*int*) – amount of pixels between the pixbufs

Returns a new pixbuf

Return type *ExtendedPixbuf*

add_vertical(*other*, *spacing=0*)

Vertically appends a pixbuf to the current

Parameters

- **other** (`GdkPixbuf.Pixbuf`) – the pixbuf to append
- **spacing** (*int*) – amount of pixels between the pixbufs

Returns a new pixbuf

Return type *ExtendedPixbuf*

composite_simple(*other*)

Composites a pixbuf on the current pixbuf at the location (0, 0)

Parameters **other** (`GdkPixbuf.Pixbuf`) – the pixbuf to composite

Returns a new pixbuf

Return type *ExtendedPixbuf*

move(*offset_x*, *offset_y*, *resize=False*)

Moves the content of the current pixbuf within its boundaries (clips overlapping data) and optionally resizes the pixbuf to contain the movement

Parameters

- **offset_x** (*int*) – the amount of pixels to move in horizontal direction
- **offset_y** (*int*) – the amount of pixels to move in vertical direction
- **resize** (*bool*) – whether to resize the pixbuf on movement

Returns a new pixbuf

Return type *ExtendedPixbuf*

multiply_horizontal(*multiplier*, *spacing=0*)
Horizontally multiplies the current pixbuf content

Parameters

- **multiplier** (*int*) – How often the pixbuf shall be multiplied
- **spacing** (*int*) – amount of pixels between the pixbufs

Returns a new pixbuf

Return type *ExtendedPixbuf*

multiply_vertical(*multiplier*, *spacing=0*)
Vertically multiplies the current pixbuf content

Parameters

- **multiplier** (*int*) – How often the pixbuf shall be multiplied
- **spacing** (*int*) – amount of pixels between the pixbufs

Returns a new pixbuf

Return type *ExtendedPixbuf*

xlgui.icons.**extended_pixbuf_new_from_file**(*filename*)
Returns a new *ExtendedPixbuf* containing an image loaded from the specified file

Parameters **filename** (*string*) – the name of the file containing the image to load

Returns a new pixbuf

Return type *ExtendedPixbuf*

CHAPTER 3

Support

The following are ways you can get support or questions to answers you might have about using or developing Exaile.

3.1 Bugs/Enhancements

If you believe you have found a bug, you can file bugs at Exaile's github issue tracker You will need to register for a github account before you can post a bug report.

If you have an idea for an enhancement you can also file that at the github issue tracker. However, unless it's something really simple, if you're not willing to work on it then chances are it won't get implemented.

3.2 Mailing lists

We run mailing lists for various purposes.

- exaile-users - A list for people who use Exaile. Open to discussion between users to communicate and help each other. If you're having trouble using Exaile, ask here.
- exaile-devel - Our primary list for general development discussion. Questions about the code, plugin development, packaging, etc. should go here.

3.3 IRC

Exaile developers can be reached on #exaile channel on Freenode. The channel is not very active, but if you stick around long enough someone will probably answer your question – think in terms of email response time.

Note: Response time in #exaile is typically measured in days. If you ask a question and leave in an hour, you probably won't get an answer to your question. If you stay there, someone will eventually answer it! Or use email.

- *User's guide*
- *Developer's guide*
- search

Python Module Index

x

Index

N

O

P

Q

R

S

T

U

V

W

X