Linux desktop app guide Documentation

Thomas Kluyver & contributors

Dec 13, 2018

Contents:

Desktop Linux can be a confusing world, with different desktops and distributions offering their own frameworks for application authors. This guide explains the pieces you need to make an application that works for most desktop Linux users.

Much of this also applies to BSD and other open source platforms, but we'll refer the 'Linux desktop' for convenience.

CHAPTER 1

User Interface options

Most of *Penguin Carpentry* is about integrating a desktop application you've already written. But if you're just getting started, one of the first things you'll need to figure out is what to use for the user interface.

As with many things in the open source world, there are many different options. This is an overview of a few of the main ones.

1.1 Desktop style: GTK or Qt

If you want a traditional desktop style application - buttons, menus, toolbars, and so on, the main options are **GTK** and **Qt**. They're written in C and C++ respectively, so you can make very efficient apps, but they also have bindings to higher-level languages like Python.

If your app will also run on Windows/Mac, go with Qt; its cross platform support is stronger than GTK. If you're writing just for Linux, it's mostly a matter of taste. GTK apps look more native in the GNOME desktop, whereas Qt apps fit better in KDE, but both toolkits work with either desktop.

See also:

Qt 5 documentation

GTK+ 3 reference manual

Python GTK+ 3 tutorial

1.2 Web tech: Electron or browser

Electron is a framework to create desktop applications using HTML and Javascript. You can easily create attractive, cross platform applications using familiar web development techniques, and a lot of new applications have been written with it. But Electron apps tend to use a lot of memory, so there's a pushback from some users.

A lightweight alternative for some applications is to run a small web server on localhost and use a browser to display an HTML interface. It's hard to integrate this nicely with the desktop, though, and you need to pay attention to security so that other websites open in the browser can't touch it.

See also:

Electron home page

1.3 Game style

Game interfaces tend to be drawn with a different set of tools. High-performance graphics will use **OpenGL** or its replacement, **Vulkan**, but there are many frameworks and engines built around these to provide more convenient APIs, such as **SDL** and **pygame**.

Note: I don't know this area well! If you can expand and improve this section, please contribute.

CHAPTER 2

Launching your application

2.1 Command line

Most applications on Linux can be launched from the command line, even if they don't show output in the terminal. To allow this, you need an executable file in any of the directories listed in the `PATH` environment variable. These will often end in `bin`, such as `/usr/local/bin`.

Executable files

On Unix, an executable file is one with the 'execute bit' set in its permissions. You can make a file executable with this command:

```
chmod +x path/to/file
```

The file should be either a compiled 'binary', or a text script. Scripts need to start with a 'shebang', a line that identifies the interpreter to run the script with. For example:

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

The command is the filename. By convention, it should be lower case. If you want to use more than one word, separate them with hyphens, e.g. `chromium-browser`.

2.2 Desktop launcher

To add your application to the desktop launcher or applications menu, you need a desktop entry file, with a `.desktop` extension. It contains something like this:

```
[Desktop Entry]
Version=1.0
Type=Application
```

(continues on next page)

(continued from previous page)

```
# The name will be displayed
Name=Inkscape
# Translations are possible: they're used depending on the system locale
Name[hi]=

# See the icon section for how this is looked up.
Icon=inkscape

# The command to launch your application. %F is for file paths to open.
Exec=inkscape %F

# File types it can open; see file associations.
MimeType=image/svg+xml;...
```

Your application's desktop entry is also used for *file association*.

There are a number of other optional fields you can use. See the links below for more information.

These desktop files are placed in an `applications` subdirectory of each *XDG data directory*. If you have to put it in place yourself, the normal locations are:

- Per-user: `~/.local/share/applications`
- System: `/usr/local/share/applications`

See also:

GNOME developer guide to desktop files

Desktop Entry Specification

2.3 File associations

To use your application for opening files from the file manager, specify the details in your *desktop file*. In the `Exec` field, make sure your launch command includes a placeholder like `%F`:

```
Exec=inkscape %F
```

`%F` may be replaced by one or more file paths. Use lowercase `%f` if it can only handle one path per command. `%U` and `%u` are similar, but they pass URLs. Local files have URLs starting `file://`, but the platform might also pass HTTP or FTP URLs to your application.

Then use the `MimeType` field to specify what MIME types it handles:

```
MimeType=image/svg+xml;image/x-eps;
```

There may be other applications that support the same MIME type. It's normally up to the user to pick the default application for a file type, but if you have a good reason to change it, you can use a `mimeapps.list` file as described in the MIME associations specification.

See also:

MIME application associations specification

If the file format you want to open isn't already defined on the system, you'll need to define a new MIME type for it.

2.3.1 Define a MIME type

A MIME type is meant to be a unique name for a file format, like `image/png` or `text/x-makefile`. For new MIME types, the recommended format is `application/vnd.`*`org_name`*`.`*`app_name`*, filling in the organisation name and app or format name as appropriate (e.g. Libreoffice ODT files are `application/vnd.oasis.opendocument.text`). You can add `+json` or `+xml` to the end if your file format is based on one of these generic data formats.

MIME types are added to the system with XML files like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<mime-info xmlns="http://www.freedesktop.org/standards/shared-mime-info">
  <mime-type type="application/vnd.acme.frobulate">
      <comment>Frobulate file</comment>
      <glob pattern="*.frobulate"/>
  </mime-type>
</mime-info>
```

The `<glob>` tag specifies a file extension for files with this MIME type. Other fields can distinguish different file types sharing the same extension, but it's best to pick a unique extension. There's no need to limit the extension to three letters.

The filename of this XML file should start with the vendor name, e.g. `acme-frobulate.xml`. Call `xdg-mime install acme-frobulate.xml` to install it. This will copy it into a directory such as `/usr/local/share/mime/packages`, and rebuild the MIME database from all of these XML source files.

See also:

Shared MIME-info database specification

CHAPTER 3

Icons

You probably want to install at least one icon for your application's *desktop launcher*. If you're *defining a file type* to be used with your application, you'll want an icon for that too.

3.1 Icon file formats

Icons on Linux are stored in standard image formats: PNG for bitmap icons, and SVG for vector icons. You can provide either format, or both.

If you use bitmap icons, you'll need to provide several files with different resolutions. The most important one for applications and file types is a 48 × 48 pixel square, and 16, 32 and 64 pixel square shapes are also common. You can provide any other sizes you like, and you can also make double resolution icons for high-DPI displays - e.g. a double resolution icon for a 32 pixel square would have 64 × 64 pixels, and may be visually simpler than a standard 64 × 64 pixel icon.

If a desktop wants to show an icon at a size for which it doesn't have a file, it will scale another size up or down. It may not look perfect, but the icon is still clearly recognisable.

3.2 File locations

Icons are organised under an `icons` subdirectory of each *XDG data directory*. So the default locations are:

- Per-user: `~/.local/share/icons`
- System: `/usr/local/share/icons`

Within each `icons` directory are a number of theme directories. The only one you need to care about as an application author is `hicolor`; this is used whenever the user's preferred theme doesn't have a particular icon. You can investigate the others if you're keen enough to make several different versions of each icon.

Within each theme are directories for different icon sizes in pixels, e.g. `48x48`. Vector icons have a separate directory called `scalable` at this level.

Within each size directory are different categories. The two most relevant for application authors are `apps` for application icons, and `mimetypes` for file types. Finally, these category directories hold the icon files.

So the path of an installed icon file looks something like this:

```
/usr/share/icons/hicolor/48x48/apps/firefox.png
```

See also:

Icon theme specification

Icon naming specification

CHAPTER 4

Notifications

Use desktop notifications to tell the user about something happening while your app is in the background, e.g. a new message arriving. On some desktops, notifications can also have clickable buttons.

Use notifications carefully, because they can easily distract a user who's trying to focus. The desktop may have controls to disable all notifications from an application, but you could also offer the user more fine-grained controls.

Wrapper libraries like *libnotify* make it easy to send notifications with a couple of lines of code. If you need to dig deeper, look at the specification linked below.

See also:

Desktop notifications in Arch Linux wiki (with examples in many programming languages)

Desktop notifications specification

CHAPTER 5

Storing data

When the user deliberately saves or exports a file, they'll normally select where it should go. Data that you want to store 'invisibly' is divided into a few categories:

- **Cache**: Data that can be regenerated or redownloaded if necessary
- **Configuration**
- **Runtime**: Transient objects your application uses while it's running
- **Application data**: Everything else

See also:

XDG base directory specification

5.1 Cache

Cached data is stored in a single folder, controlled by an environment variable. It's a good idea to make a subdirectory for your application.

`XDG_CACHE_HOME`
A path to a directory where cache data should be stored. If empty or unset, the default is `~/.cache`.

5.2 Configuration & application data

Both configuration and application data have an ordered list of directories to look in. If you're storing data at runtime, you should write to the appropriate 'home' location, which is also the highest priority location to look for files. The other locations allow config/data files to be installed systemwide.

If your application will need more than a single file in either of these categories, it's a good idea to use a subdirectory to group together the files it uses.

XDG_CONFIG_HOME
A path to a directory where config data should be stored. If empty or unset, the default is `~/.config`.

XDG_CONFIG_DIRS
A colon-separated list of directories to look for config files, in addition to `XDG_CONFIG_HOME`. If empty or unset, the default is `/etc/xdg`.

XDG_DATA_HOME
A path to a directory where application data should be stored. If empty or unset, the default is `~/.local/share`.

XDG_DATA_DIRS
A colon-separated list of directories to look for application data files, in addition to `XDG_DATA_HOME`. If empty or unset, the default is `/usr/local/share/:/usr/share/`.

5.3 Runtime data

Runtime data is stored in a single directory. This is meant for communication and synchronisation between different components of your application. It has some special guarantees and limitations:

- The directory must support special filesystem objects like named pipes and unix sockets.
- Each user has their own runtime directory, and it is only accessible to them.
- Runtime data is not shared between computers, whereas your home directory may be shared using something like NFS.
- The directory is deleted when the user logs out, and files not modified in six hours may be removed even while logged in (set the *sticky bit* to prevent that).
- It may be kept purely in memory, so don't try to store lots of data here.

XDG_RUNTIME_DIR
The path of a directory to store runtime data. There is no default; if it's not set, the desktop has not set up a runtime directory.

CHAPTER 6

Indices and tables

- genindex
- search

Index

D

E

P