Canonical

Developer Guide: IoT Graphics with Ubuntu Frame

April 2023

Ubuntu Frame explained

Ubuntu Frame is the foundation for embedded displays. It provides a reliable, secure and easy way to embed your applications into a kiosk-style, IoT device, or digital signage solution. With Ubuntu Frame, the graphic application you choose or design gets a fullscreen window, a dedicated set of windows behaviours, input from touch, keyboard and mouse without needing to deal with the specific hardware, on-screen keyboard, remote desktop, and more.

Together with Ubuntu Core, Ubuntu Frame provides all the infrastructure you need to securely deploy and maintain graphic applications on edge devices. And while Ubuntu Core maximises performance and security of your apps, Ubuntu Frame is compatible with any Linux operating system that supports snaps.

This developer guide will show you how to deploy your graphic application that supports the Wayland protocol to work with Ubuntu Frame and Ubuntu Core. This guide is for developers looking to build kiosks, digital signage solutions, infotainment systems, IoT devices or any other applications that require a graphic interface running on a screen.

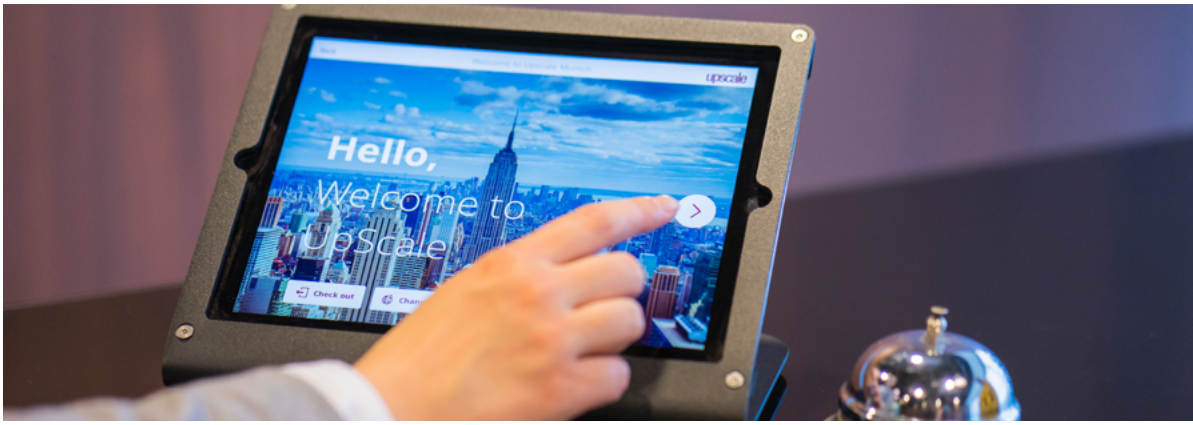
We will cover:

1. Setting up the tools and environment required to package and deploy your application on your desktop
2. Testing if an application works with Ubuntu Frame on your desktop
3. Troubleshooting some common issues
4. Packaging the app as a snap and testing whether the snap works on your desktop
5. Packaging the snap for an IoT device and testing it on the device

If you want to learn how to install pre-built applications such as wpe-webkit-mir-kiosk, mir-kiosk-kodi, or Scummvm, follow their official installation and configuration guides.

Note: This guide will not cover how to build an application using a toolkit that supports Wayland (there are many). And while it is possible to package X11-based applications to work on Ubuntu Core, this guide will not cover this either. We will also not cover how to upload your snap to the snap store, nor building custom Ubuntu Core images with pre-configured snaps. That is documented on snapcraft.io/docs.

If you are new to Ubuntu Core, we recommend reading our getting started document. If you want to learn about building custom Ubuntu Core images, you could find information on the snapcraft docs.



Requirements

Developers use different tools and processes to build their graphic applications. For the purpose of this guide, we assume that you have an application that supports the Wayland protocol that you can test on your Linux-based desktop.

It is possible to work in a container or on a different computer (if snapd and X forwarding work well enough). But those options are outside the current scope.

For some of the later steps, you will need an Ubuntu One account. This will let you enable `remote-build` on your Launchpad account and publish on the Snap Store.

Setting up your test environment

Ubuntu Frame provides a tool for developers to simulate how their end application will look and respond in your development environment. So, you don't need to work directly on your target device to perform the first design and usability iterations.

Open a terminal window and type:

```
sudo snap install ubuntu-frame
```

Snapcraft is a command-line utility for building snaps. This software allows users to build their own applications or software packages, and then publish them to the Snap Store.

In the same terminal window type:

```
sudo snap install snapcraft --classic
```

If you don't have git installed, now is a good time to install it (on Ubuntu, use the command `sudo apt install git`).

Checking your application works with Ubuntu Frame

There can be problems with both getting your application to work well with Ubuntu Frame and getting your application to work in a snap. To avoid confusion, we recommend first testing your application with Ubuntu Frame before packaging it as a snap. In this section, you will test your application, explore some common issues you might run into, and learn how to fix them.

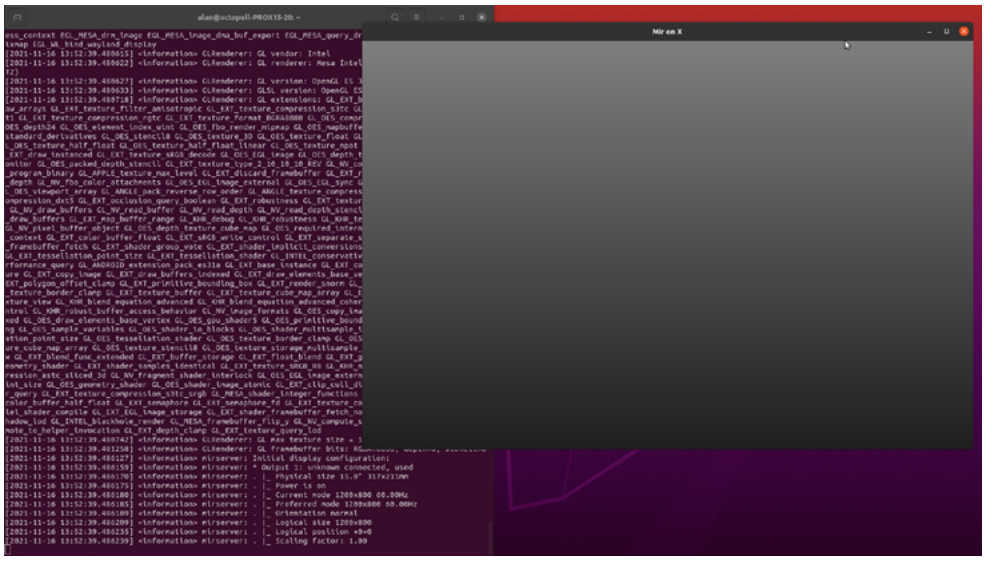
Enabling Ubuntu Frame simulator

The key thing to know about connecting an app to Ubuntu Frame is that the connection is controlled by the `WAYLAND_DISPLAY` environment variable. You need to set that to the same value for both processes and, because you may be using a Wayland-based desktop environment (which uses the default of `wayland-0`), set it to `wayland-99`.

In a terminal window (that you'll use for this section and the next section) type:

```
export WAYLAND_DISPLAY=wayland-99
ubuntu-frame&
```

You should see a "Ubuntu Frame" window containing a graduated grey background. Mir is the library on which Ubuntu Frame is based, and for testing your app in a window, you should use it's "Ubuntu Frame" simulator.



Testing your application on Ubuntu Frame simulator

You can use Electron, Flutter, Qt, or any other toolkit or programming language to develop your graphic application. There is no sole path for checking all of them. Instead, this guide will use some example applications using GTK, QT, and SDL2.

The examples used here are game applications, such as Mastermind, Neverputt, and Bomber. We've chosen these applications as they are easily installable and are designed to work without a full desktop session. But they can be replaced by your kiosk application, industrial GUI, smart fridge GUI, digital sign and more.

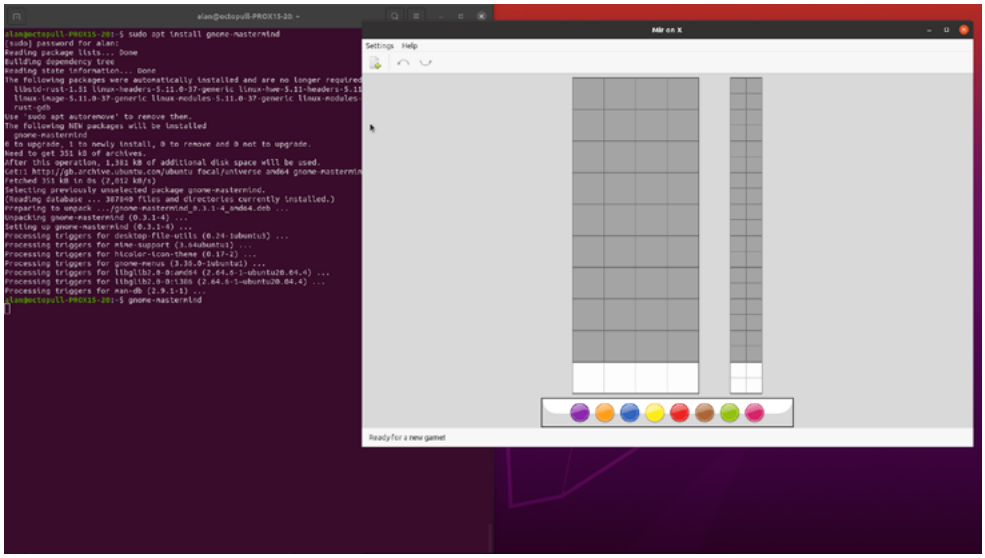
So, the first step is to download these applications and execute them. If you are able to see it on the “Ubuntu Frame” window, then you know that your application works with Ubuntu Frame.

Now, still in the same terminal window, type:

GTK Example: Mastermind

```
sudo apt install gnome-mastermind
gnome-mastermind
```

Now Frame’s “Mir on X” window should contain the “Mastermind” game.



If your application doesn’t appear or look right at this stage, then this is the time to work out the fix, before packaging as a snap. You will see some examples of the problems you might encounter in the following examples.

Close Mastermind (`Ctrl-C`) and try the next example:

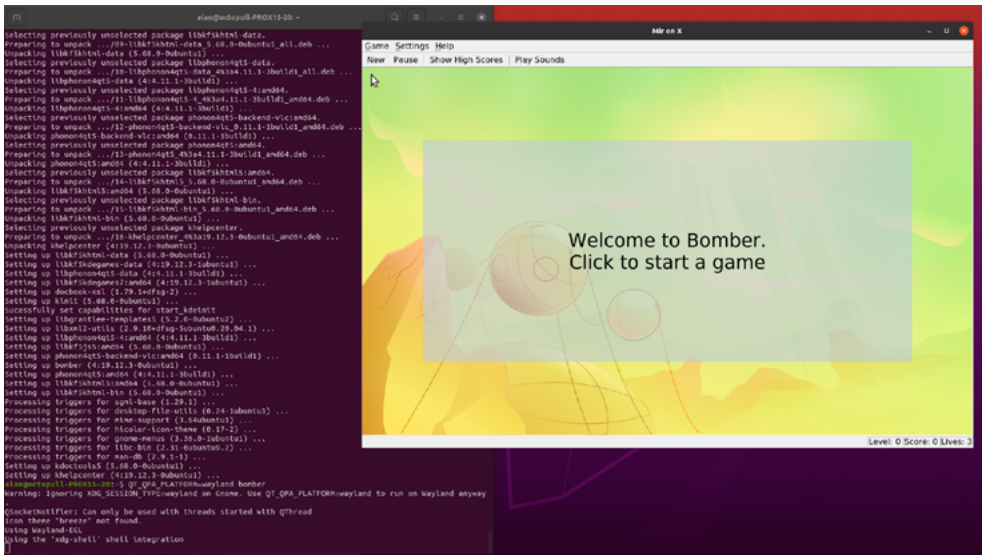
Qt Example: Bomber

```
sudo apt install bomber
bomber
```

What you see now is that Bomber appears in its own window, not in the Ubuntu Frame window. The problem is that Qt does not default to using Wayland and that `QT_QPA_PLATFORM` needs to be set to get that behaviour. Close it and try again with:

```
QT_QPA_PLATFORM=wayland bomber
```

Now Frame’s “Mir on X” window should contain the “Bomber” game as shown in the next image.



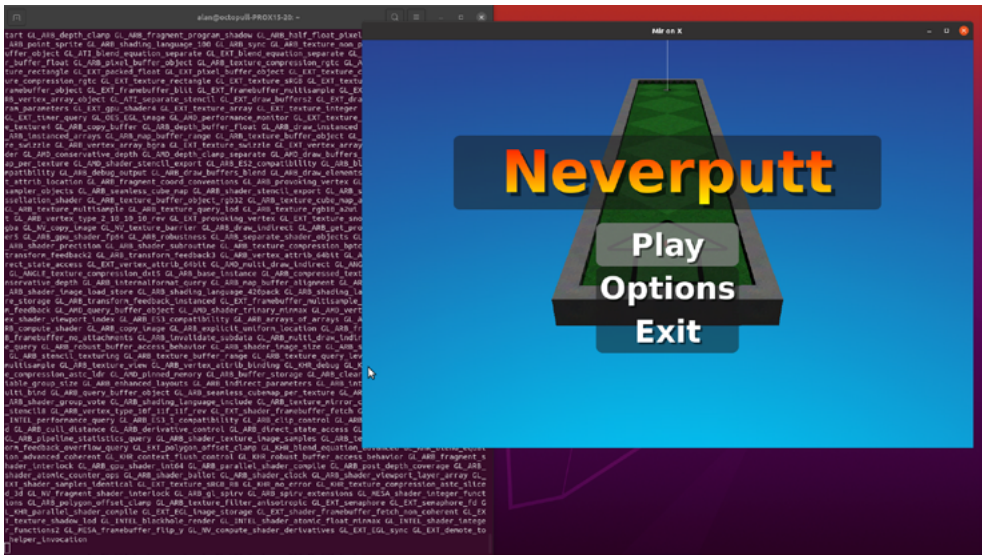
You will incorporate setting `QT_QPA_PLATFORM` in the "environment" of the snap recipe later. Close that (`Ctrl-C`) and try the next example.

SDL2 Example: Neverputt

```
sudo apt install neverputt
SDL_VIDEODRIVER=wayland neverputt
```

Now Frame's "Mir on X" window should contain the "Neverputt" game. Note that SDL2 does not default to using Wayland and that `SDL_VIDEODRIVER` needs to be set to get that behaviour.

The other thing you'll see is that the game doesn't fill the display. That's because the application doesn't understand Ubuntu Frame telling it to fill the screen. This is a problem with some applications and, in this case, can be fixed by editing `~/.neverball/neverballrc` [sic] to say "`fullscreen 1`" and restarting the game. (The same file works for neverball, but you may need to figure out the right configuration file for your application.) This is what you will see:



You will incorporate setting `SDL_VIDEODRIVER` and the `neverballrc` setting in the snap recipe later. You can now close the app (`Ctrl-C`).

Packaging your application as a Snap

Now that you know how to confirm that an application is working with Ubuntu Frame, the next step is to use snap packaging to prepare the application for use on an IoT device. As before, this section will show you how to package your application together with some issues you might find and their troubleshoot. Snap packaging for IoT graphics

For use with Ubuntu Core, your application needs to be packaged as a snap. This will also allow you to leverage Over The Air updates, automatic rollbacks, delta updates, update semantic channels, and more. If you don't use Ubuntu Core, but instead another form of Linux, we recommend you use snaps to get many of these advantages.

There's a lot of information about packaging snaps online, and the purpose here is not to teach about the snapcraft packaging tool or the Snap Store. We will, instead, focus on the things that are special to IoT graphics.

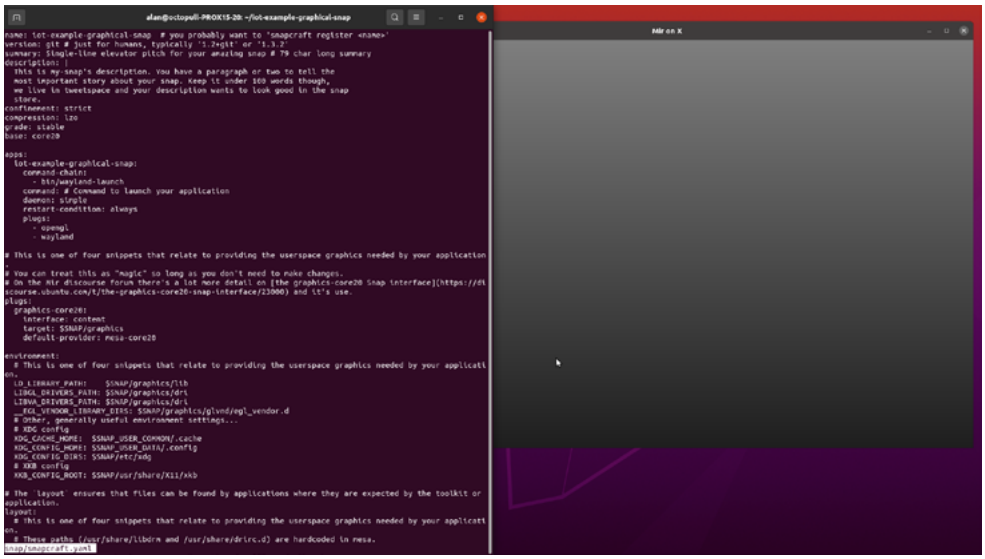
Much of what you find online about packaging GUI applications as a snap refers to packaging for desktop. Some of that doesn't apply to IoT as Ubuntu Core and Ubuntu Server do not include everything a desktop installation does and the snaps need to run as daemons (background services) instead of being launched in a user session. In particular, you should ignore various Snapcraft extensions that help writing snap recipes that integrate with the desktop environment (e.g. using the correct theme).

Writing snap recipes without these extensions is not difficult as we'll illustrate for each of the example programs used in the previous section.

First, you will clone a repository containing a generic Snapcraft recipe for IoT graphics. In the same terminal window you opened at the start of the last section, type:

```
git clone https://github.com/MirServer/iot-example-graphical-snap.git
cd iot-example-graphical-snap
```

If you look in `snap/snapcraft.yaml`, you'll see a generic "snapcraft recipe" for an IoT graphics snap. This is where you will insert instructions for packaging your application. This is how the .yaml file looks like:



The customised snapcraft recipe for each example described in this guide (i.e. GTK, Qt and SDL2) is on a corresponding branch in this repository:

```
$ git branch -a
* master
  remotes/origin/GTK3-mastermind
  remotes/origin/HEAD -> origin/master
  remotes/origin/Qt5-bomber
  remotes/origin/SDL2-neverputt
  remotes/origin/master
```

Once you have the customised snapcraft recipe you can snap your example applications.

GTK Example: Mastermind

Switch to the GTK example branch. Then use `snapcraft` to build the snap:

```
git checkout GTK3-mastermind
snapcraft
```

Snapcraft is the packaging tool used to create snaps. We are not going to explore all its options here but, to avoid confusion, note that when you first run `snapcraft`, you will be asked "Support for 'multipass' needs to be set up. Would you like to do it now? [y/N]:", answer "yes".

After a few minutes, the snap will be built with a message like:

```
Snapped iot-example-graphical-snap_0+git.7c73b6a_amd64.snap
```

You can then install and run the snap:

```
sudo snap install --dangerous *.snap
snap run iot-example-graphical-snap
```

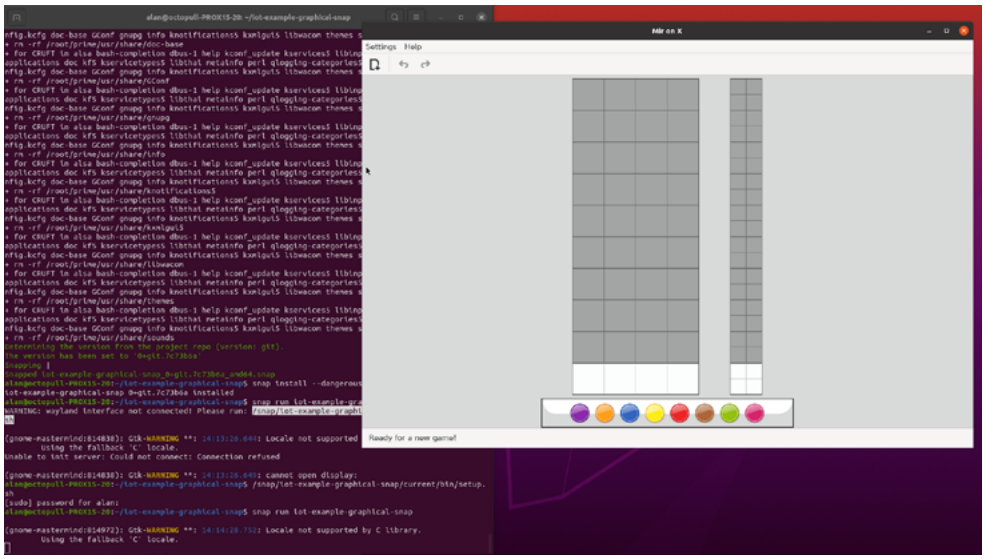
The first time you run your snap with Ubuntu Frame installed, you are likely to see a warning:

```
WARNING: wayland interface not connected! Please run: /snap/iot-example-
graphical-snap/current/bin/setup.sh

(gnome-mastermind:814838): Gtk-WARNING **: 14:13:26.644: Locale not
supported by C library.
    Using the fallback 'C' locale.
Unable to init server: Could not connect: Connection refused

(gnome-mastermind:814838): Gtk-WARNING **: 14:13:26.649: cannot open
display:
```

The first WARNING is the key to the problem and comes from one of the scripts in the generic recipe. While developing your snap (that is, until your snap is uploaded to the store and any necessary "store assertions" granted), connecting any "interfaces" your snap uses needs to be done manually. As the message suggests, there's a helper script for this. Run it and try again:

```
/snap/iot-example-graphical-snap/current/bin/setup.sh
snap run iot-example-graphical-snap
```

Now Frame's "Mir on X" window should contain the "Mastermind" game.



Close that (`Ctrl-C`) and try the next example:

Qt Example: Bomber

To avoid confusion, delete the `.snap` file created with the previous example:

```
rm *.snap
```

Now switch to the Qt `first-try` example branch. Then build, install, and run the snap:

```
git checkout Qt5-bomber-first-try
snapcraft
sudo snap install --dangerous *.snap
snap run iot-example-graphical-snap
```

If you've been paying attention, you'll notice that the branch name `Qt5-bomber-first-try` is not what you might expect. This is to show you the sort of problem you might encounter:

```
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_
PLATFORM=wayland to run on Wayland anyway.
QSocketNotifier: Can only be used with threads started with QThread
"Session bus not found\nTo circumvent this problem try the following
command (with Linux and bash)\nexport $(dbus-launch)"
```

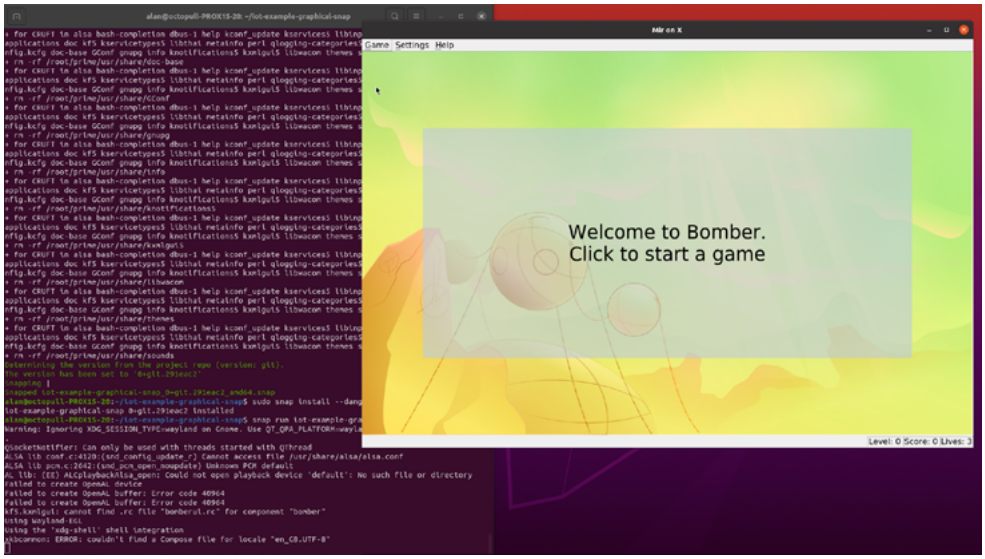
This didn't work as the bomber application requires a DBus "session bus". To solve this issue, you can provide one within the snap using `dbus-run-session`. You can see exactly how this is done by comparing branches:

```
git diff Qt5-bomber-first-try Qt5-bomber
```

Now switch to the Qt example branch. Then build, install and run the snap:

```
git checkout Qt5-bomber
snapcraft
sudo snap install --dangerous *.snap
snap run iot-example-graphical-snap
```

Now Frame's "Mir on X" should contain the "Bomber" game.



Close that (`Ctrl-C`) and try the next example:

SDL2 Example: Neverputt

To avoid confusion, delete the `.snap` file created with the previous example:

```
rm *.snap
```

Now switch to the SDL2 example branch. Then build, install, and run the snap:

```
git checkout SDL2-neverputt
snapcraft
sudo snap install --dangerous *.snap
snap run iot-example-graphical-snap
```

Now Frame's "Mir on X" window should contain the "Neverputt" game.

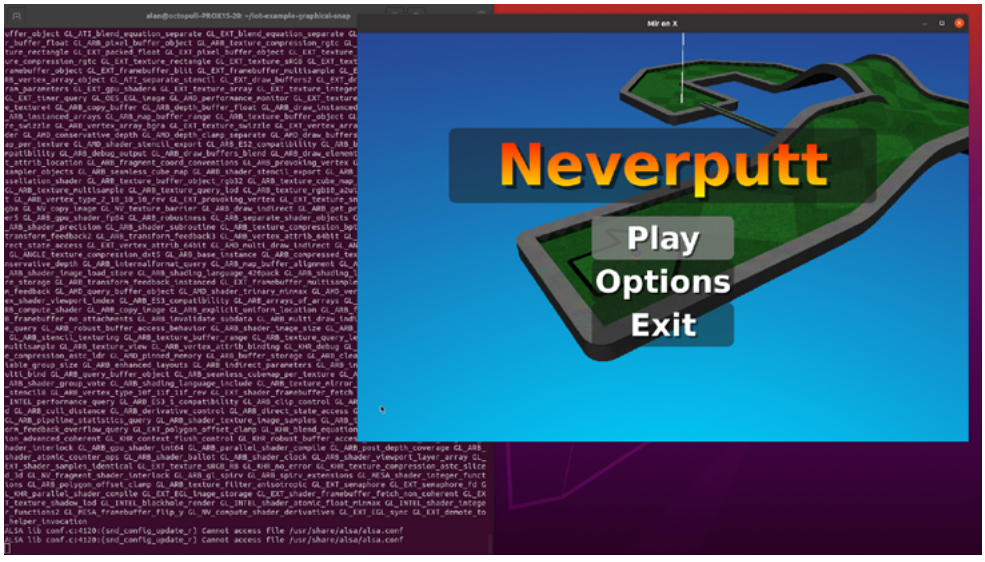
But you are likely to see a warning as Neverputt uses additional plugs that have not been connected:

```
WARNING: hardware-observe interface not connected! Please run: /snap/
iot-example-graphical-snap/current/bin/setup.sh
WARNING: joystick interface not connected! Please run: /snap/iot-example-
graphical-snap/current/bin/setup.sh
ALSA lib conf.c:4120:(snd_config_update_r) Cannot access file /usr/share/
alsa/alsa.conf
Failure to initialize SDL (Could not initialize UDEV)
```

You have warnings requesting that the setup script is run. This is because the Neverputt snap needs some interfaces that you haven't connected yet, and this could be the case with other applications that you build. Identifying the interfaces needed by your application and adding them to the `snapcraft.yaml` recipe is one of the things you need to do at this stage.

Run the setup script to connect the missing interfaces, and try again:

```
/snap/iot-example-graphical-snap/current/bin/setup.sh
snap run iot-example-graphical-snap
```



Close that (`Ctrl-C`) and close the Ubuntu Frame window. Your application has been successfully snapped.

Packaging your own application

When packaging an application there are many issues to address: what needs to be in the snap, how does the runtime environment need to be configured and what interfaces are needed.

You might get some inspiration from the examples we've given. You can see the customisation used in each example using `git diff` for example:

```
git diff master SDL2-neverputt
```

You'll see, for example, the `SDL_VIDEODRIVER` settings and the `neverballrc` file in this example. But we can't go into all the possibilities and tools in this guide. There are helpful docs and forums on the Snapcraft website.

Building for and installing on a device

So far you explored the process for testing if your snapped application will work on Ubuntu Frame only using your desktop. While this accelerates the development process, you still need to consider the final board for your edge device. A lot of edge devices don't use the **amd64** architecture that is typical of development machines. Therefore, in this section, you will see how to build for and install on a device if your device uses a different architecture, using the SDL2 Neverputt application as an example. You will also see how to troubleshoot some common issues.

Leveraging Snapcraft remote build tool

The simplest way to build your snap for other architectures is:

```
snapcraft remote-build
```

This uses the Launchpad build farm to build each of the architectures supported by the snap. This requires you to have a Launchpad account and to be okay uploading your snap source to a public location.

Once the build is complete, you can scp the .snap file to your IoT device and install using `--dangerous`.

For the sake of this guide, we are using a VM set up using the approach described in Ubuntu Core: Preparing a virtual machine with graphics support. Apart from the address used for scp and ssh this is the same as any other device and makes showing screenshots easier.

```
scp -P 10022 *.snap <username>@localhost:~
ssh -p 10022  <username>@localhost
snap install ubuntu-frame
snap install --dangerous *.snap
```

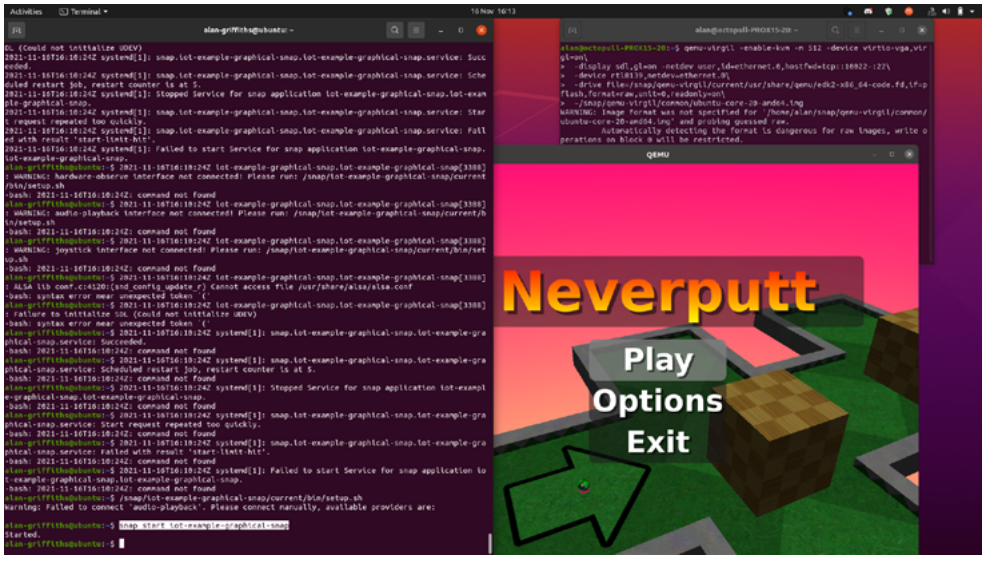
You'll see the Ubuntu Frame grayscale background once that instals, but (if you've been following the steps precisely) you won't see Neverputt start:

```
$ snap logs -n 30 iot-example-graphical-snap
...
2021-11-16T16:10:24Z iot-example-graphical-snap.iot-example-graphical-
snap[3388]: WARNING: hardware-observe interface not connected! Please
run: /snap/iot-example-graphical-snap/current/bin/setup.sh
2021-11-16T16:10:24Z iot-example-graphical-snap.iot-example-graphical-
snap[3388]: WARNING: audio-playback interface not connected! Please run:
/snap/iot-example-graphical-snap/current/bin/setup.sh
2021-11-16T16:10:24Z iot-example-graphical-snap.iot-example-graphical-
snap[3388]: WARNING: joystick interface not connected! Please run: /
snap/iot-example-graphical-snap/current/bin/setup.sh
2021-11-16T16:10:24Z iot-example-graphical-snap.iot-example-graphical-
snap[3388]: ALSA lib conf.c:4120:(snd_config_update_r) Cannot access file /
usr/share/alsa/alsa.conf
2021-11-16T16:10:24Z iot-example-graphical-snap.iot-example-graphical-
snap[3388]: Failure to initialize SDL (Could not initialize UDEV)
2021-11-16T16:10:24Z systemd[1]: snap.iot-example-graphical-snap.iot-
example-graphical-snap.service: Succeeded.
2021-11-16T16:10:24Z systemd[1]: snap.iot-example-graphical-snap.iot-
example-graphical-snap.service: Scheduled restart job, restart counter
is at 5.
2021-11-16T16:10:24Z systemd[1]: Stopped Service for snap application iot-
example-graphical-snap.iot-example-graphical-snap.
2021-11-16T16:10:24Z systemd[1]: snap.iot-example-graphical-snap.iot-
example-graphical-snap.service: Start request repeated too quickly.
2021-11-16T16:10:24Z systemd[1]: snap.iot-example-graphical-snap.iot-
example-graphical-snap.service: Failed with result ‘start-limit-hit’.
2021-11-16T16:10:24Z systemd[1]: Failed to start Service for snap
application iot-example-graphical-snap.iot-example-graphical-snap.
...
```

All these WARNING messages give the clue: you’re still developing the snap and interfaces are not yet being connected automatically. So, connect the missing interfaces and manually start the daemon:

```
/snap/iot-example-graphical-snap/current/bin/setup.sh
snap start iot-example-graphical-snap
```

You should see Neverputt starting.



Conclusion

From testing to deployment, this guide shows you how to use Ubuntu Frame to deploy your graphic applications. It covers topics as seting up the tools and environment on your desktop, testing if an application works with Ubuntu Frame, and packaging the application as a snap for an IoT device. It also included some issues you can encounter working with your applications and how to troubleshoot them.

We have also shown all the steps needed to get your snap running on a device. The rest of your development process is the same as for any other snap: uploading the snap to the store and installing on devices from there. There are just a few parting things to note:

Now that the Snap interfaces are configured, the application will automatically start when the system (re)starts.

Once you've uploaded the snap to the store, you can request store assertions to auto-connect any required interfaces. Alternatively, if you are building a Snap Appliance, then you can connect the interfaces in the "Gadget Snap".

For more information about Ubuntu Frame please visit our website.

You may also consider reading the following materials:

- Read Ubuntu Frame's documentation
- Where does Ubuntu Frame work?
- How to set up digital signage on Ubuntu Frame
- How to use remote assistance with Ubuntu Frame
- How to use the Ubuntu Frame diagnostic feature
- How to enable on-screen keyboard support in Ubuntu Frame

Need help in getting to market? Contact us.

Canonical