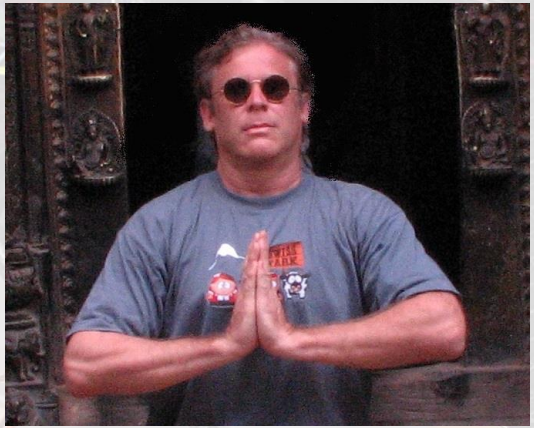
Getting Started with PX4

For Contributors

Mark West

PX4 Community Volunteer



Format

A word about format

Limited Time + Big Subject = Compression

Some Subjects Skipped = Look in Appendix

Demo Videos Clipped = Look in Appendix

Who is this for?

Anybody

Levels:

- L1: Operate a PX4 vehicle
- L2: Build a PX4 vehicle
- L3: Build the source
- L4: Modify the source
- L5: Contribute

L1: Operate

You want to operate a drivable unit

See Appendix : Operate Vehicle

L2: Build a Vehicle

You want to build a drivable unit

See Appendix : Build Vehicle

L3: Build the Source

You want to build the image from source

- Big step up from L2
- Choose
  - Toolchain
  - Container

L3: Build the Source : Toolchain

All the tools you need to build an image / exe

Compilers / Linkers / Tools / Etc

Installation:

Manual : If needed
Convenience Scripts : Preferred

L3: Build the Source : Container

Container?

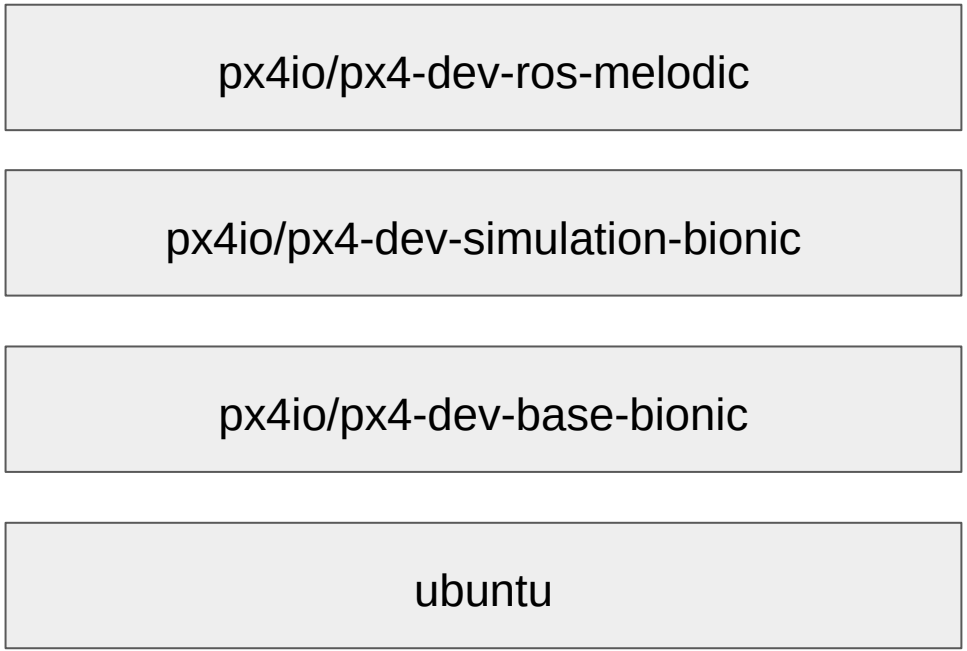
Like a better VM, with toolchain installed

Easy to install, easy to update, no interaction

Image = container snapshot

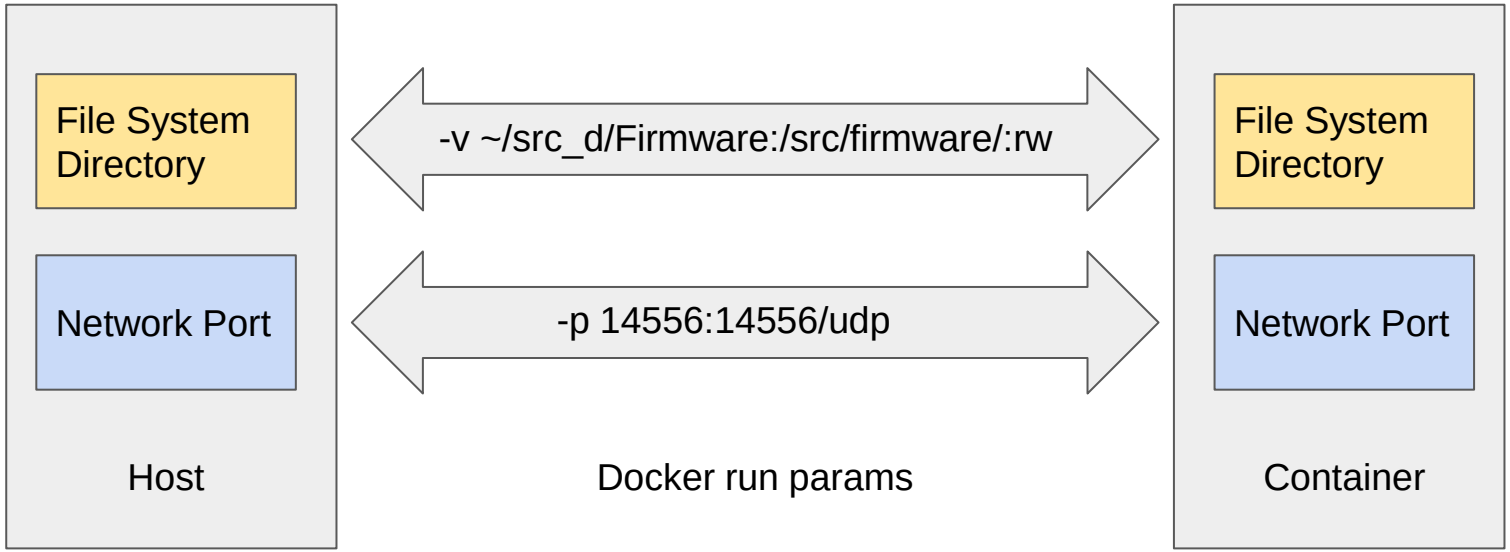
L3: Build the Source : Container

Container images built in layers



L3: Build the Source : Container

Containers are isolated, all interaction is explicit



L3: Build the Source : Toolchain or Container?

Container if possible

Toolchain if no suitable container

* But you can make your own containers

L3: Build the Source : Toolchain Demo

Build image for SITL Simulation

1) Install Toolchain
2) Git PX4 Source
3) Build PX4
4) Takeoff

L3: Build the Source : Container Demo

Build image for SITL Simulation

1) Install Docker
2) Git PX4 Source
3) Locate Container
4) Run Container
5) Build PX4
6) Build available on host computer

L4: Modify the Source

Next step up in complexity

Why? Fix bugs, add features

Loop: Edit, Build, Debug

IDE: Visual Studio, Eclipse, QT Creator, etc.

L4: Modify the Source : Build and Debug Demo

Build and Debug with an IDE

1) Install IDE
2) Git PX4 Source
3) Open Folder
   Select Kit and Variant
4) Build
5) Debug
   Breakpoint and Step

L4: Modify the Source : More Debugging

Big Subject

1) Simulation (HITL, SITL)
2) Console
3) Logs
4) JTAG

L5: Contribute

Scenario: You want to contribute to the community

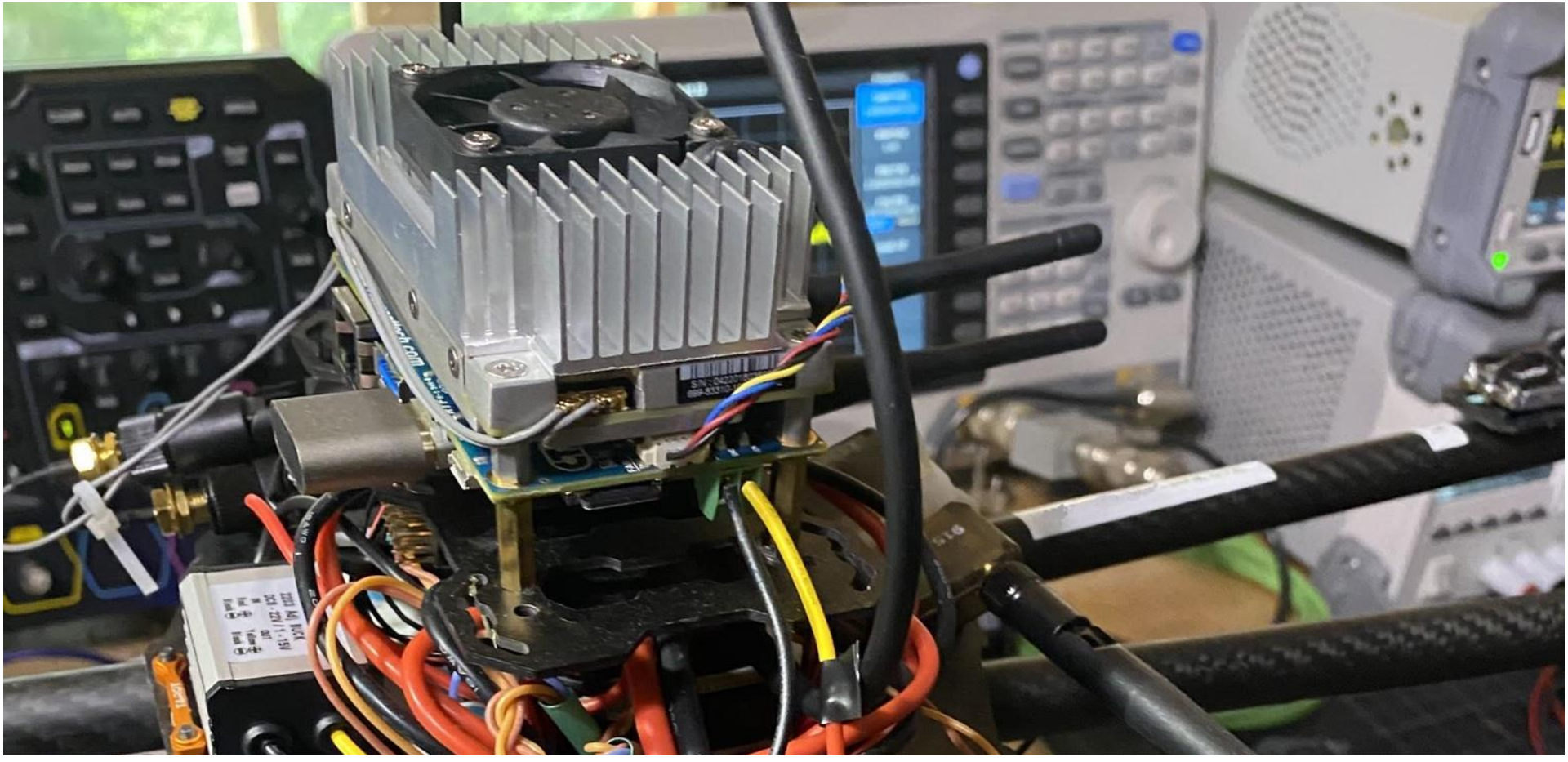
- Add Features, Fixes, Content
- Test Flights
- Report Bugs, Debug, Analyze
- See Appendix : PX4 Community

Thanks for Watching

Lots of content in the appendix. The answers to all of your questions are there :-).

Mark West
mark@telemething.com

Appendix



Appendix : What is PX4

-Software

PX4 is flight control software which supports a number of platforms: Fixed wing, rotor, rover, and submarine.
See https://docs.px4.io/master/en/airframes/airframe_reference.html

BSD open source, created by Lorenz Meier and team in 2008 while at ETH. Hosted on Github.
Anyone is free to obtain, modify, and deploy PX4.
See https://auterion.com/the-history-of-pixhawk/

-Hardware

Pixhawk standard: community support, full compatibility.
Manufacturer supported: shared support, variations in compatibility.
Unsupported: manufacturer only support, variation in compatibility."

See https://px4.io/ecosystem/compatible-hardware/

Appendix : PX4 Community

Dronecode : https://www.dronecode.org/

PX4 is part of Dronecode, and Dronecode is part of the Linux Foundation. Dronecode's mission is to support open source drone software

Website: https://px4.io/

PX4 main site

Slack : https://slack.px4.io/

70 channels, something for everyone. Good, but messages time out and disappear quickly on free license.

Forum : https://discuss.px4.io/

Traditional message board. Messages don't expire.

DevCall : https://dev.px4.io/master/en/contribute/dev_call.html

Standard Scrum, moves quickly, not for chit-chat. A good place for newcomers to observe and get a feel for what's what and look for opportunities to contribute. Hosted on Zoom

Appendix : Operate Vehicle

Scenario: You want to operate a PX4 vehicle. It is already built and can (probably) maneuver at some level

Tasks: Pair Radio, Configure, Tune (maybe). See https://docs.px4.io/v1.9.0/en/getting_started/

In addition to the **vehicle** you will need a **radio controller** and optionally **QGroundControl** software

- Radio Controller

A radio controller is the handheld unit which you use to communicate with the mobile frame during field operations.

While it is possible to do without an RC (by use of control from an onboard companion computer), you will probably need or want to use a radio controller.

A radio controller communicates with the flight controller via a transceiver which is physically connected with the flight controller. The controller and on-board transceiver communicate two way, exchanging commands and telemetry.

Prior to flight the controller and transceiver must be paired to one another.
See https://docs.px4.io/master/en/getting_started/rc_transmitter_receiver.html

Appendix : Operate Vehicle

- QGroundControl

QGroundControl is a ground station, which runs on a computer separate from the PX4 mobile platform. You can use it to perform pretty much any task you need to configure the flight controller and most other hardware mounted on the mobile platform. In addition to configuration the FC can also be used for monitoring and mission tasks.

There are several operations which need to be performed prior to flight. The standard procedure is to connect the flight controller (on the mobile frame) to a computer via USB and then use QGroundControl to perform those operations. Some typical things to perform: airframe selection, sensor orientation calibration, radio channel mapping.

See http://qgroundcontrol.com/

Appendix : Build Vehicle

Scenario: You want to build a PX4 vehicle.

You will acquire physical components and assemble them.

See https://docs.px4.io/v1.9.0/en/assembly/

A PX4 flight controller pretty much always comes delivered with PX4 installed.

If you want to change that version then follow the instructions at https://docs.qgroundcontrol.com/en/SetupView/Firmware.html

Appendix : Toolchain

- Toolchain

A toolchain is all the stuff you'll need to build for a particular target: Win, Mac, Linux. We'll use Ubuntu to build for simulation on Ubuntu.

Several convenience scripts, use is recommended is possible.

Sometimes the scripts don't meet at requirements, for example : If work with ROS using some openCV features which must be built (not in a distrib), you will need to build ROS up from source too, therefore can't use the ROS convenience script. The same goes for Gazebo. Some people find that they run into so many problems with conflicting OpenCV that they always just install the whole chain manually, despite the extra installation and maintenance issues.

For guidance on both convenience scripts and manual installation see
https://dev.px4.io/master/en/setup/dev_env.html

Appendix : Toolchain Demo

Here we demonstrate building an image for SITL simulation

1) Install the toolchain
   see https://dev.px4.io/v1.9.0/en/setup/dev_env_linux_ubuntu.html

2) Create a directory for the PX4 source, then Git

```
mkdir src
cd src
git clone https://github.com/PX4/Firmware.git --recursive
```

3) Navigate to the Firmware dir, build PX4

```
cd Firmware
make px4_sitl jmavsim
```

4) If all goes well a JMavSim will appear and the console will open the PX4 shell. Give the shell a moment to initialize, then takeoff.

```
commander takeoff
```

5) The build process commences. When complete, the build output will be present under ~/src/Firmware/build/"

Appendix : Containers

Docker is a platform for the management of containers. There are other types of containers but Docker is the default for non enterprise applications. See https://www.docker.com/

A container behaves much like a VM (isolation), but is usually smaller and more efficient. They do this with namespaces and cgroups

Containers are in very heavy use in the big clouds and are a popular way of distributing dev environments. They allow you as a consumer to plug and play very sophisticated environments which will not conflict with each other or the host.

You can elect to use PX4 containers instead of installing and maintaining a PX4 toolchain (recommended). See https://github.com/PX4/containers/

Containers build upon other containers, each layer contains all aspects of parent layers. An important part of working with containers is figuring out which one you want to use. For example: the px4-dev-ros-melodic containers adds ROS to px4-dev-simulation-bionic, which itself adds simulation to px4-dev-base-bionic.

All interaction of the container with the host is prohibited, except as explicitly enabled by command line parameters. For example: it is very common for a directory in the user domain on the host be mapped to a directory on the container through the '-v' parameter, and it is very common to open a network port via the '-p' parameter.

Appendix : Container Build Demo

1) Install Docker

See https://www.docker.com/

2) Create a directory for the PX4 source, then Git

```
mkdir src_d
cd src_d
git clone https://github.com/PX4/Firmware.git --recursive
cd Firmware
```

3) Add current user to Docker group

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

Log out/in

4) Select which container you wish to use

Go to https://github.com/PX4/containers/, select your container

We'll use **px4-dev-simulation-bionic**, click on that link

Notice the Docker Pull Command, you can run this command on your command line to download and install the container, but this isn't required. Just note the name of the container: **px4io/px4-dev-simulation-bionic**.

Appendix : Container Build Demo

5) Run the container

```
sudo docker run -it --privileged \
    --env=LOCAL_USER_ID="$(id -u)" \
    -v ~/src_d/Firmware:/src/firmware/:rw \
    -v /tmp/.X11-unix:/tmp/.X11-unix:ro \
    -e DISPLAY=:0 \
    -p 14556:14556/udp \
    --name=px4sim px4io/px4-dev-simulation-bionic bash
```

Notice PX4 source dir: '~/src_d', container name: 'px4sim', and image name: px4io/px4io/px4-dev-simulation-bionic. These may likely be unique to your instance.

At this point the container images will download, install, and run. Your prompt will change, which indicates that you are now running in the container

6) Navigate to the share on the container, build PX4

```
cd src/firmware
 make px4_sitl_default
```

The conventional build process commences. When complete, the build directory will be present on the share on the host!. You can exit the container or leave it running.
On the host (not in the container) navigate to 'src_d', you will see the build directory and output have been created.

Appendix : IDE Build Debug Demo

Part 1: Building

1) Install VSCode. See https://code.visualstudio.com/Download

2) Create a directory for the PX4 source, then Git

```
mkdir src_vsc
cd src_d
git clone https://github.com/PX4/Firmware.git --recursive
```

3) Open VSC, open folder **~/src_vsc/Firmware**.

A lot of config happens, wait, don't break out. Breaking out early will require VSC restart

If 'This workspace has extension recommendations' prompt appears in lower right, click 'Show Recommendations', select all except 'CMake'.More config happens, wait, don't break out.

4) If **[PX4 detect]** is not active kit, change to **[PX4 detect]**. Wait for config to finish.

5) Select build variant, we'll use [**px4_sitl]** here. Wait for config to finish.

6) Click **Build**. A standard build ensues. If problems happen then messages will be seen. Sometimes the **problems tab** is the best view, sometimes the **output tab** is the best view. You can double click in either, you may be taken to the problem code.

Appendix : IDE Build Debug Demo

Part 2: Debugging

1) Set a breakpoint at **~/src-vsc/Firmware/platforms/posix/src/p44/common/main.cpp** in **main()** on an executable line.

2) Click the **Debug icon** in the left horizontal toolbar. Click on the **Start Debugging green triangle** in the upper left dropdown.
   * if VSCode brings up dialog with "Errors exist after running preLaunchTask 'gazebo.iris', and 'terminal' tab shows 'Could not open file[/home/<you>/src_vsc/Firmware/Tools/sitl_gazebo/worlds/iris.world]'
     grab a copy from /home/<you>/src/Firmware/Tools/sitl_gazebo/worlds/iris.world. Then restart VSC.

3) If all is well you should see a) a build, b) a startup sequence, c) code halting on the breakpoint. At this point you can single step, add and remove breakpoints, inspect variables, etc.

Appendix : More Debugging

Debugging / running / analysis is a huge area. We provide some references to help you get started.

Simulation: https://dev.px4.io/master/en/simulation/

Simulation : Airsim: https://github.com/Microsoft/AirSim/blob/master/docs/px4_setup.md

Debug Console: https://dev.px4.io/master/en/debug/consoles.html

Logging: https://dev.px4.io/master/en/log/logging.html

JTAG Debugging: https://dev.px4.io/master/en/debug/swd_debug.html

J-Link Visual Studio Code: https://wiki.segger.com/J-Link_Visual_Studio_Code

Babel UAVCAN adapter: https://zubax.com/products/babel

Zubax DroneCode Probe: https://shop.titaneliteinc.com/index.php?route=product/product&product_id=1294